AUTONOMOUS STRATEGY PLANNING UNDER UNCERTAINTY

by

Alp Sardağ

BS in Computer Engineering, Boğaziçi University, 1996

MS in Systems and Control Engineering, Boğaziçi University, 1998

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

Graduate Program in Computer Engineering

Boğaziçi University

2006

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to these people who made this study possible. First, thank you to all pioneers, McCallum, Thrun, Duda, Hart, and others that have inspired many researchers like me. Next, to Prof. H. Levent Akın who was always there when I needed guidance with his knowledge and patience, and because he has always been a great advisor. He supported all the ideas and guided me to the solution by asking the necessary questions.

I wish to thank to Prof. Ethem Alpaydın who always supported the ideas, helped to adapt the classical POMDP problems used in this research to continuous case, and to Assoc. Prof. Yağmur Denizhan for great courses which helped much in the mathematical background of this thesis.

A special thank to my wife, Leyla Sardağ, for her valuable assistance and moral support. She showed great patience throughout this study, and made it possible for me to carry on when I got exhausted.

And finally thanks to all my precious AILAB friends for always listening with patience during the AILAB seminars.

ABSTRACT

AUTONOMOUS STRATEGY PLANNING UNDER UNCERTAINTY

A real world environment is often partially observable for agents either because of noisy sensors or incomplete perception. Autonomous strategy planning under uncertainty has two major challenges. The first one is autonomous segmentation of the state space for a given task, and the second, emerging complex behaviors, that deal with each state segment. This thesis proposes three new approaches, namely ARKAQ-Learning, KAFAQ-Learning and KBVI, that handle both challenges by utilizing combinations of various techniques. ARKAQ makes use of ART2-A Networks augmented with Kalman Filters and Q-Learning. KAFAQ is a finite state automaton using Kalman filters and Q-Learning. KBVI uses Monte Carlo methods and introduces a new technique to calculate Q-values for continuous domains.

All are online algorithms with relatively low space and time complexity. The algorithms were run for some well-known Partially Observable Markov Decision Process problems, where the problem of representing the value function is more difficult than the discrete case because inputs are continuous distributions. The algorithms could reveal the hidden states, mapping non-Markovian observations to internal belief states, and also could construct an approximate optimal policy on the internal belief state space.

ÖZET

BELİRSİZLİK ALTINDA ÖZERK STRATEJİ GELİŞTİRME

Gerçek dünya, bu ortamdaki etmenler tarafından gürültülü algılayıcılar ya da eksik algı nedeniyle kısmi olarak gözlemlenebilir. Belirsizlik altında özerk strateji geliştirmenin karşısında iki büyük engel vardır. Verilen bir görev için sürekli durum uzayının özerk olarak bölgelere ayrılmasıve bu bölgeler üzerinde amaca yönelik karmaşık davranışların ortaya çıkarılması. Bu tezde isimleri ARKAQ-Öğrenme, KAFAQ-Öğrenme ve KBVI olan ve çeşitli tekniklerin bir araya getirilmesinden oluşan üç yeni yaklaşım önerilmektedir. ARKAQ-Öğrenme yapısında Kalman filtreleme özelliği eklenmiş, ART2-A ağı ve Q-Öğrenme metodları kullanılmıştır. KAFAQ-Öğrenme, Kalman filtreleme ve Q-Öğrenme yöntemlerini kullanan bir sonlu durum makinasıdır. KBVI ise Monte Carlo metodlari kullanmakta ve sürekli durum ortamlarında Q-değerlerinin hesaplanması için yeni bir teknik ortaya koymaktadır.

Bütün yordamlar gerçek zamanlıdır ve göreceli olarak düşük yer ve zaman karmaşıklıkları vardır. Yordamlar iyi bilinen Kısmen Gözlemlenebilir Markov Karar Süreç problemleri üzerinde uygulanmıştır. Burada sürekli dağılımlar kullanıldığı için değer fonksiyonunun gösterimi daha zorlaştırılmıştır. Yordamlar Markov olmayan gözlemleri iç inanç durumları ile ilişkilendirerek saklı durumları ortaya çıkarabilmiş ve iç inanç durum uzayı üzerinde yaklaşık olarak en iyi bir davranış politikası oluşturabilmişlerdir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS/ABBREVIATIONS

a	Action
A	Kalman state transition model
b	Belief
B	Kalman control-input model
h	History
H	Kalman observation model
o	Observation
p	Probability
P	Uncertainty value
Q	Q-value or Kalman process noise
R	Reward or Kalman measurement noise
s	State
T	State Transition Function
V	Long term value of each state or belief
$\boldsymbol{\gamma}$	Vigilance parameter or learning factor
$\boldsymbol{\pi}$	Policy
$1-NN$	One Nearest Neighbour
ARKAQ	Q-learning technique using ART and Kalman filter
ART	Adaptive Resonance Theory
KAFAQ	Q-learning technique using Kalman filters and generating an FSA policy
KBVI	Kalman based value iteration technique.
LTM	Long Term Memory
MDP	Markov Decision Process
POMDP	Partially Observable Markov Decision Process
STM	Short Term Memory

1. INTRODUCTION

The problem of planning under uncertainty has received significant attention in the scientific community over the past few years. It is now well recognized that considering uncertainty during planning and decision-making is imperative for the design of robust autonomous systems. This is particularly crucial in robotics, where the ability to interact effectively with real-world environments is a prerequisite for success.

In the early days of AI, planning was restricted to simple tasks in unrealistic environments; actions had few and predictable effects, and could be combined sequentially to satisfy the desired goal. This gave rise to a rich and successful set of approaches that could handle planning problems of increasing complexity, including the ability to satisfy multiple goals, handle time constraints, quickly re-plan, and so on. However, these methods generally relied on the assumption that the true state of the world could be sensed exactly and reliably.

While this assumption is reasonable in some highly structured domains, this is clearly not the case in many real-world problems. For example, significant research on natural language dialogue systems has sought to devise techniques for recovering state information through conversing with a person (Boyce, 2000). Similarly, in robotics, sensor limitations are pervasive and the seemingly simple problem of recovering the state from sensor measurements is the key subject of research in entire research programs.

One framework for modeling is the *Markov process*. The Markov process describes how the state of the world evolves, and rests upon two key assumptions. The first assumption is that the world can be described probabilistically. That is, each possible state of the world can be identified, and there exist well-defined, stationary probability distributions that describe how the world can change as a consequence of each action. The second assumption is known as the *Markov assumption*, where the future is assumed to be independent of the past conditioned upon knowledge of the

present. If a robot knows its current state, then nothing about how it arrived at its current state contains further information that can help predict what can happen next. (Note that the Markov assumption says nothing about how the current state is known, and the history of the robot may have a great deal to say about how the state evolved.) Once the current state is known, the past is of no predictive value.

The practical limitation of the Markov decision process is that the exact state of the world is assumed to be known at all times. A *policy* dictates the action to take from each state, but it is not always possible to know what state the world is in. In most real world situations, the true state of the world is not directly observable. For example, the absolute position of a robot can often not be known. In such situations, the state is described as *hidden*. Instead of directly observing the state of the world, the agent receives observations (e.g., sensor measurements) that permit possible states to be inferred. But, these observations rarely allow the agent to infer the current state uniquely and correctly. The concept of partial observability describes this scenario in which the state is not known.

An autonomous learner agent must reveal a strategy to reach a goal, i.e. it has to decide on a sequence of actions. This would be relatively easy if the environment was fully observable, that is the state of the agent could be identified exactly. However, in most real world environments, not only the environment is partially observable but also the agent perception is coupled with noise.

This research proposes policy learning methods for an agent with incomplete and unreliable sensor-input while acquiring adequate state recognition mechanism according to the interaction with its environment.

Converting the perceptual sensor inputs to sufficiently accurate state recognition, to decide on the next action, is not trivial for embodied agents such as robots. Therefore, in order to realize autonomous behaviors without embedding some prior knowledge by a human designer, the agent has to construct this conversion mechanism through the interaction with the environment. In order to realize this, it is necessary

to solve the following essential problems:

- Estimation of the current state using sensory inputs that lack accuracy owing to locality and noise,
- Segmentation of continuous and multidimensional sensor-space based on the significance for the ongoing task.
- Assign an optimal action to each state that will help the agent to accomplish its goal.

Partially Observable Markov Decision Process (POMDP) serves as a natural model for planning in such environments where effects of actions are non-deterministic and consequently, the state of the world is not known with certainty. Unfortunately, standard methods for planning, inference and learning with POMDP have been shown to be PSPACE-complete (Papadimitriou and Tsitsiklis, 1987), and their existence for infinite-horizon POMDPs — undecidable (Madani et al., 1999). Because of the intractability of current solution algorithms, especially those that use dynamic programming to construct (approximately) optimal value functions (Smallwood and Sondik, 1973, Cassandra et al., 1997), the application of POMDPs remains limited to very small number of problems.

There are two competing views of why POMDPs are generally so computationally intractable, as well as the appropriate ways of overcoming this intractability. These views can be referred to as *the curse of dimensionality* and *the curse of history*. The curse of dimensionality refers to the fact that, for a POMDP with n states, the optimal POMDP policy must be computed over an $n-1$ dimensional belief space, a substantially difficult task. The curse of history refers to the fact that each belief is in essence a sufficient statistic over a history of actions and observations. The number of different such histories grows exponentially with the planning horizon, leading to computational intractability. There is a tension between these two confounding factors, in that algorithms that overcome one curse often suffer from to the second. The algorithms presented in this thesis overcome both curses.

Another major flaw of the current POMDP solution methods is the assumption that the agent's world model is known. The process of world model extraction and representation is often very time consuming and contains a chunk of idealized assumptions. For these reasons, in this research we assumed that the world model is not known. This assumption reduces the burden and responsibility of the user of our algorithms who needs an approximate policy for its agent, but at the same time making the problem extremely difficult to solve for us.

Moreover, in this research, the agent operates over continuous state space and the perceived sensor inputs yield continuous observation values. In this case the agent's belief simplex dimensionality is infinite, since the agent does not have an infinite memory, we suggest a parametric representation of the belief state. This parametric representation will save us from the curse of dimensionality. To overcome the curse of history we will use *point backups* that represent the policy over the belief simplex. The points are generated through the exploration of the agent not like other point based methods where points are chosen randomly. Another central idea of this research is that probabilistic state estimation can be integrated into decision making for real world problems tractably while still producing good policies if compact representations of the probabilistic estimates are used.

This research suggests three algorithms that can be used to solve continuous state and observation space POMDPs. Although, these algorithms are based on the assumptions of continuous state and observation space POMDPs, we will show that *Kalman based value iteration* (KBVI) can handle also discrete state POMDP problems. Although continuous state and observation space assumption models better the real world, it lacks some important features of the discrete case such as the representation of the policy as piecewise linear convex manner over the belief simplex. This feature loss makes impossible to extend the algorithms designed for discrete case to continuous case. As mentioned, the impossibility in the extraction of a realistic world model is a major cause that prevents the algorithms to used in real world. Therefore, we designed our algorithms on the assumption that the underlying world dynamics are not known. Since the problem now seems like impossible to solve, no previous work exists so far

that attempted to solve POMDP problems under these assumptions.

In chapter 2, we will begin by describing Markov processes more formally, and examine how conventional value iteration is used to find policies for POMDPs. We will also examine prior methods in solving various kinds POMDPs. In chapter 3, we will develop the ARKAQ-learning algorithm, and evaluate its performance on some well-known POMDP problems. In chapter 4, we will develop the KAFAQ-learning algorithm, and will detail why it is better than ARKAQ-learning in solving continuous state POMDP problems. We will also run KAFAQ-learning on the same experiments as ARKAQ-learning. In chapter 5, we will present KBVI-learning. In chapter 6, we will examine the differences between ARKAQ and KAFAQ-learning and give a comparison. Finally, in chapter 7, we conclude with a summary of the work, and a discussion of some of the more interesting results. We discuss some of the unresolved problems of planning under uncertainty and give future work.

2. BACKGROUND

Since the real world is continuous, it can be best modeled by continuous domain POMDPs instead of discrete ones as has been the conventional approach until recently. Solving continuous domain POMDPs in an efficient way, in other words autonomously generating an approximate or exact optimal policy is very important for real world applications. In order to decide an optimal action for the current state, the agent must have a way of choosing among possible actions. In *reinforcement learning* (Sutton and Barto, 1998) the agent is rewarded or punished by a reinforcement signal depending on the decided action. This serves as a tool to generate the optimal policy. All of our algorithms use the reinforcement learning framework.

At each decision step, the agent needs first to compute the current belief. The POMDP framework uses the Bayes filter to update the internal belief of the agent. In the continuous case POMDP, we will represent the belief state by Gaussian distributions. As a natural result of this representation, we will use Kalman filters (Kalman, 1960) to update the agent's current belief.

In our research, the approximately optimal policy is represented by the approximately optimal action shared for all spatially connected belief points. To group spatially connected belief points sharing a common best action, we used clustering algorithms such as *One Nearest Neighbour (1-NN)* and *Adaptive Resonance Theory (ART)* networks (Carpenter and Grossberg, 1987a).

Sometimes the agent may be kidnapped and put to an arbitrary location. In such cases multi-hypothesis tracking is necessary, and as one may expect, the current belief is represented by a multi-modal Gaussian.

2.1. Reinforcement Learning

Reinforcement learning is the process of learning to behave optimally with respect to some scalar feedback value through trial-and-error interactions with a dynamic environment over a period of time (Sutton et al., 1999). The agent may or may not have *a priori* knowledge on the real environment it interacts with.

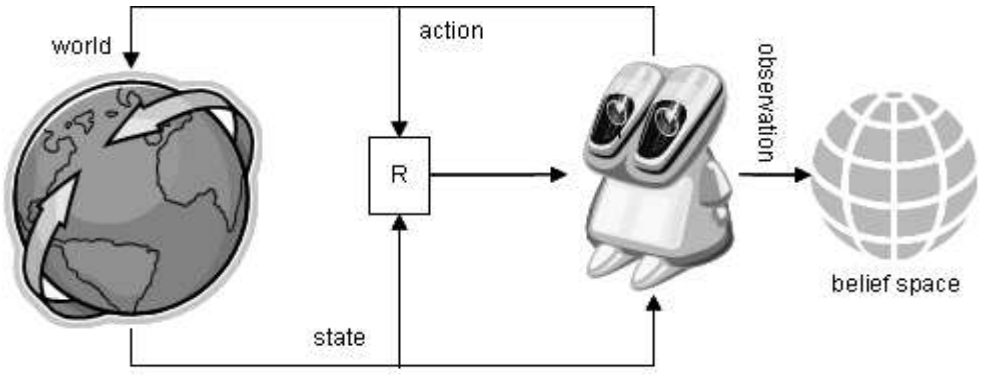

Figure 2.1. The reinforcement learning agent.

In the standard reinforcement learning model, an agent interacts with the environment through its sensor suite, and its actions. Once given some indication of the current state $s(t)$ of the environment at time t as an input as shown in Figure 2.1, the agent then chooses the action $a(t)$ according to some rule, often called a *policy*, denoted by π. This action changes the state of the environment. The effectiveness of the action taken and the resulting state transition is communicated to the agent through a scalar value $r(t)$, called the *reinforcement signal*, sometimes described as *reward* or *cost* or *feedback*. The agent behavior should choose actions that tend to increase the long run sum of values of the reinforcement signal. It can learn to act optimally over time by trial-and-error, guided by different kind of algorithms that will be mentioned later.

Reinforcement learning is different from supervised learning (Russell and Norvig, 2003) in many ways. The most important difference is that there are no input/output pairs. Instead, after choosing an action in an uncertain state the agent is told the reward but not the correct action. The agent should explore the environment about the possible states, actions and rewards to act to find optimal policy. Another difference is that the evaluation of the system is often concurrent with learning.

Reinforcement learning is a difficult problem because the learning system may perform an action and not be told whether that action was good or bad. For example, a learning auto-pilot program might be given control of a simulator and told not to crash. It will have to make many decisions each second and then, after acting on thousands of decisions, the aircraft might crash. What should the system learn from this experience? Which of its many actions were responsible for the crash? Credit assignment to individual actions is the problem that makes reinforcement learning difficult. However, there is a solution to this problem, which is based on dynamic programming (Berry and Fristedt, 1985), and it involves two basic principles. First, if an action causes something bad to happen immediately, such as crashing the plane, then the system learns not to do that action in that situation again. So whatever action the system performed one millisecond before the crash, it will avoid doing in the future. But that principle does not help for all the earlier actions, which did not lead to immediate disaster.

The second principle is that if all the actions in a certain situation lead to bad results, then that situation should be avoided. So if the system has experienced a certain combination of altitude and airspeed many different times, whereby trying a different action each time, and all actions led to something bad, then it will learn that the situation itself is bad. This is a powerful principle, because the learning system can now learn how to fly without crashing. In the future, any time it chooses an action that leads to this particular situation, it will immediately learn that particular action is bad, without having to wait for the crash.

By using these two principles, a learning system can learn to fly a plane, control a robot, or do similar tasks. It can first learn on a simulator, then fine tune on the actual system. This technique is generally referred to as dynamic programming, and a deeper analysis will reveal below how dynamic programming can generate the optimal value function.

2.1.1. Evaluation Metrics for Optimal Policy

For any reinforcement learning algorithm, the implementer should consider the model of optimality. There are generally three models of evaluating the optimality of a policy.

In the *finite-horizon* model the agent should optimize its expected reward given in Equation (2.1) for the next k steps.

$$R = E\left(\sum_{t=0}^{k} r(t)\right) \tag{2.1}$$

This type of optimality measure is used when the agent's lifetime is known in advance, and the policy is non-stationary; that is, it changes over time.

The *discounted infinite-horizon* (Bonet and Geffner, 2000, Boutilier et al., 1999) model takes into account the long run reward of the agent but future rewards are geometrically discounted according to discount factor γ, where $0 \leq \gamma < 1$.

$$R = E\left(\sum_{t=0}^{\infty} \gamma^t\, r(t)\right) \tag{2.2}$$

Many interpretations for γ are possible such as the probability of living another step or as a mathematical trick to put a bound on the infinite sum. This model is more mathematically tractable especially in the *finite-transient* case and is widely used (Sondik, 1978).

Another criterion for evaluating optimality is the average reward model, where the agent optimizes its long run average reward.

$$R = \lim_{k\to\infty} E\left(\frac{1}{k}\sum_{t=0}^{k} r(t)\right) \tag{2.3}$$

In general, the choice of the optimality model and parameters influence the policy considerably, so it must be chosen carefully. In KBVI algorithm, we used the combination

finite-horizon and discounted infinite-horizon model which is given in Equation (2.4).

$$\lim_{k\to\infty} E\left(\sum_{t=0}^{k} \gamma^t r(t)\right) \tag{2.4}$$

2.1.2. Evaluation of the Performance of the Learning Algorithms

It is important to be able to evaluate the quality of any learning algorithm. One can use several measures:

- Convergence to optimal value at some point,
- The time it takes to converge to optimal value,
- Regret, the expected decrease in reward gained due to executing the learning algorithm instead of behaving optimally from the very beginning. (Berry 1985)

We have evaluated our algorithms according to all the criteria given above and the results are given in section 7.

2.1.3. Exploration vs. Exploitation

The major difference between a supervised learner and a reinforcement learner agent is that, the latter must explore its environment in order to find the best policy. The fundamental question in reinforcement learning research is: How do we devise an algorithm that will efficiently find the optimal value function? Bellman (Bellman, 1957) showed that the optimal value function is a solution to the set of equations defined by the Bellman equation.[1] Here, the process of learning was described as the process of improving an approximation of the optimal value function by incrementally finding a solution to this set of equations. One should note that the Bellman equation is defined over all of the state space. The optimal value function satisfies this equation for all s in state space. This requirement introduces the need for exploration. Exploration is defined as intentionally choosing to perform an action that is not considered "best" for

[1]Detailed explanation of this equation will be given in MDP Algorithms part of this section.

the express purpose of acquiring knowledge of unseen (or little seen) states. In order to identify a sub-optimal approximation, the state space must be sufficiently explored. Our strategy was to evenly choose each possible action by assigning equal selection probabilities to each action.

The n-armed bandit problem (Sutton, 1988), which has been the subject of a many studies in the statistics and applied mathematics literature, is a very simple case of the exploration problem (Berry 1985). The agent is in a room with a collection of n gambling machines, each one called "one-armed bandit". The agent is permitted a fixed number of pulls; any arm may be pulled on each turn. The machines do not require a deposit to play; the only cost is in wasting a pull playing a sub optimal machine. When arm_i is pulled, $machine_i$ pays off one or zero according to some underlying probability parameter, p_i, where payoffs are independent events and the p_i s are unknown. What should the agent strategy be?

This problem illustrates the fundamental trade off between exploitation and exploration. The gambler must choose which arm to play on each successive trial: the one that has paid off best or maybe one that has not been tried. The n-armed bandit problem is a formal model for real problems in many vitally important areas. Each arm corresponds to an action and the payoff from pulling the arm corresponds to the benefits obtained from taking the action. Exploration is risky, expensive and has uncertain payoffs; on the other hand, failure to explore at all means that one never discovers any actions that may be worthwhile.

2.1.4. Immediate vs. Delayed Reward

If the reward is issued as soon as the action is performed, this is called an *immediate reward*. If the reward is issued once at the end of a session, this is a *delayed reward*. For example, learning tic-tac-toe is handled by delayed reward where the criterion is winning the game or not, and learning ping-pong requires immediate rewards since the focus is on not dropping the ball at each turn. Likewise, in this research, depending on the problem definition the agent is rewarded immediately or the reward is delayed.

By the definition, the tiger problem (Cassandra, 1999) is an immediate reward case, the Littman's one dimensional maze (Cassandra 1999) is a delayed reward case.

2.1.5. Markov Decision Process (MDP)

MDP is one of the modeling techniques for a reinforcement learner agent. However, Markov model assumes that the underlying world state is completely known by the agent. This is not realistic for many environments. Even for a human agent, it is not possible to see his/her back which invalidates the complete observation of the underlying state of the Markov model. Although it is not suitable for modeling environments in our research, discussion of this model will help better understand the partially observable models that will be discussed later. Below the formal definition of the MDP is given (Bellman 1957).

An MDP consists of a 4-tuple (S, A, T, R) where,

- S is the set of states,
- A is the set of actions,
- R is the reward function $S \times A \rightarrow R$,
- T is the state transition function $S \times A \rightarrow \pi(S)$, where a member of $\pi(S)$ is a probability distribution over the set S.

The state transition function, probabilistically describes the next state of the environment as a function of its current state and the agent's action.

2.1.6. MDP Algorithms

If we have a method of determining the long-term value of acting in each state, then an agent that knows the state can act optimally by the definition of a Markov process. Dynamic Programming (Bellman 1957), allows us to determine the long-term value $V(S)$ for each state S. Dynamic Programming is summarized by the Bellman

Equation, where $0 \leq \gamma < 1$.

$$V(s) = \max_a \left(R(s,a) + \gamma \sum_{S' \in S} T(s,a,s') V(s') \right), \forall s \in S \tag{2.5}$$

This is an iterative procedure for determining, the optimal value of $V(S)$, $V^*(S)$. In the limit, as the number of iterations goes to infinity, $V(S)$ converges to $V^*(S)$. Given $V^*(S)$, we can specify the optimal policy as

$$\pi^*(s) = \arg\max_a \left(R(s,a) + \gamma \sum_{S' \in S} T(s,a,s') V^*(s') \right) \tag{2.6}$$

One way to find an optimal policy is to find the optimal value function. It can be determined by a simple iterative algorithm called *value iteration* that can be shown to converge to the correct V^* values (Bellman 1957, Bertsekas, 1987). The basis of *value iteration* is as follows. If it is assumed that the function approximator used to represent V^* is a lookup table (each state has a corresponding element in the table whose entry is the approximated state value), then one can find the optimal value function by performing sweeps through state space, updating the value of each state according to Equation (2.5) until a sweep through state space is performed in which there are no further changes to state values (the state values have converged) which corresponds to evaluating Equation (2.5) until the maximum difference between two successive value functions is less than a small value, ε. This is known as *ε-convergence*, and forms a policy from Equation (2.6). Equation (2.5) has complexity order $O(|A||S|^2)$ for each iteration, and becomes intractable for very large state spaces. In addition, the transition probabilities, T, may not always be available. These two observations motivate Monte-Carlo methods for computing V^*. These methods learn by interacting with the world and gathering experience about the long-term rewards from each state.

Thus far it has been assumed that our function approximator is a lookup table, which is normally the case in classical dynamic programming. However, this assumption severely limits the size and complexity of the problems that can be solved. Many real-world problems have extremely large or even continuous state spaces. In practice,

it is not possible to represent the value function for such problems using a lookup table. Hence, an extension to classical value iteration is to use a function approximator that can generalize and interpolate values of states never before seen. For example, one might use a neural network for the approximation V^*. The resulting method is referred to as a *residual gradient algorithm* (Baird, 1995) because gradient descent is performed on the mean squared Bellman residual.

2.1.7. Q-learning

Q-learning is another extension to traditional dynamic programming (value iteration) that solves the problem of learning value functions *Q(S, A): S×A→R,* which represent the value of taking action A in state S and then acting optimally. It is summarized by the following update rule (Sutton 1999), which introduces a learning rate $0 \leq \alpha_t < 1$.

$$Q_t(s,a) = Q_t(s,a) + \alpha_t \left(r_{t+1} + \gamma \max_{a'} Q_{t+1}(s',a') - Q_t(s,a) \right) \tag{2.7}$$

Equation (2.7) states that the value $Q_t(s,a)$ should be updated in the direction of the error denoted by

$$\left(r_{t+1} + \gamma \max_{a'} Q_t(s',a') - Q_t(s,a) \right) \tag{2.8}$$

Another kind of Q-learning called Q(λ)-learning (Watkins, 1989), is an important reinforcement learning method. It combines Q-learning and TD(λ) (Sutton 1988). Q(λ) is widely used and is believed to outperform the one step Q-learning, since it uses single experiences to update evaluations of multiple state-action pairs that have occurred in the past.

As it is possible to represent the value function with a neural network in the context of value iteration, it is also possible to represent the Q-function with a neural network in the context of Q-learning.

The values converge, as each state is visited infinitely often and α_t is decreased in such a way that $\sum_t \alpha_t = \infty \quad and \quad \sum_t \alpha_t^2 < \infty$ (Mitchell, 1997). Q-learning is an off-policy method, meaning the policy used to gather experience may be different from the current policy. This allows the reduction of the number of expensive world interactions by repeatedly reusing experience gathered under an old policy to improve the current policy. We have used $Q_t(s, a)$ and reduced the cost of introducing partial observability (Singh et al., 1995, Engel and Mannor, 2001) by automatically aggregating states into clusters.

In this research, we have used Q(λ=0) learning, as a future work we plan to investigate the effects of different λ values. Moreover, the world model generator layer in ARKAQ and KAFAQ-learning that will be introduced in sections 3 and 4 respectively, online segment the continuous belief simplex into finite number of spatially connected belief segments sharing a best common action, ε-convergence was sufficient to calculate the approximate optimal policy. For KBVI algorithm, the Q-values are approximately calculated from the beginning and only the policy convergence is important.

2.2. Partially Observable Markov Decision Processes

In many real world environments, it will not be possible for the agent to have complete perception, in other words the world is partially observable. When designing agents that can act under uncertainty it is convenient to model the environment as a Partially Observable Markov Decision Process (POMDP). The model incorporates uncertainty in the agent's perceptions, actions and feedback. As in MDP, a reinforcement signal is given as feedback to agents, either immediate or delayed.

Consider, for example, a problem of patient management. The patient comes to a hospital with an initial set of complaints. Only rarely do these allow the doctor (decision-maker) to diagnose the underlying disease with certainty, so that a number of disease options generally remain open after the initial evaluation. The doctor has multiple choices in managing the patient. He/she can choose to do nothing (wait and see), order additional tests and learn more about the patient state and disease, or pro-

ceed to a more radical treatment (e.g. surgery). Making the right decision is not easy task. The disease the patient suffers can progress over time and may become worse if the window of opportunity for a particular effective treatment is missed. On the other hand, selection of the wrong treatment may make the patient's condition worse, or may prevent applying the correct treatment later. The result of the treatment is typically non-deterministic and more outcomes are possible. In addition, both the treatment and investigative choices come with different costs. Thus, in a course of patient management, the decision-maker must carefully evaluate the costs and benefits of both current and future choices, as well as their interaction and ordering. Other decision problems with similar characteristics (e.g. complex temporal cost-benefit tradeoffs, stochasticity, and partial observability of the underlying control process) include robot navigation, robot soccer, car driving, and the like.

To model such problems, POMDP can be used. POMDP contains two sources of uncertainty: the stochasticity of underlying controlled process (e.g. disease dynamics in the patient management problem), and imperfect observability of its states via a set of noisy observations (e.g. symptoms, findings, test results). Noisy observations and the ability to model and reason with information gathering actions are the main features that distinguish the POMDP from the widely known MDP.

Although useful in modeling, POMDP has the disadvantage of being hard to solve, it has been shown that computing exact optimal policy is intractable for problems more than few tens of states, observations and actions. The complexity of POMDP algorithms grows exponentially with the number of state variables, making it infeasible for large problems (Boyen and Koller, 1998, Sallans, 2000).

The approximate methods dealing with the POMDP model seem more applicable to large or continuous state spaces. Most of the methods whether approximate or exact, assume that the underlying POMDP parameters are known. It is very optimistic to expect the existence of such prior data exists for a situated agent. We will assume that these parameters are not known, making the task extremely difficult.

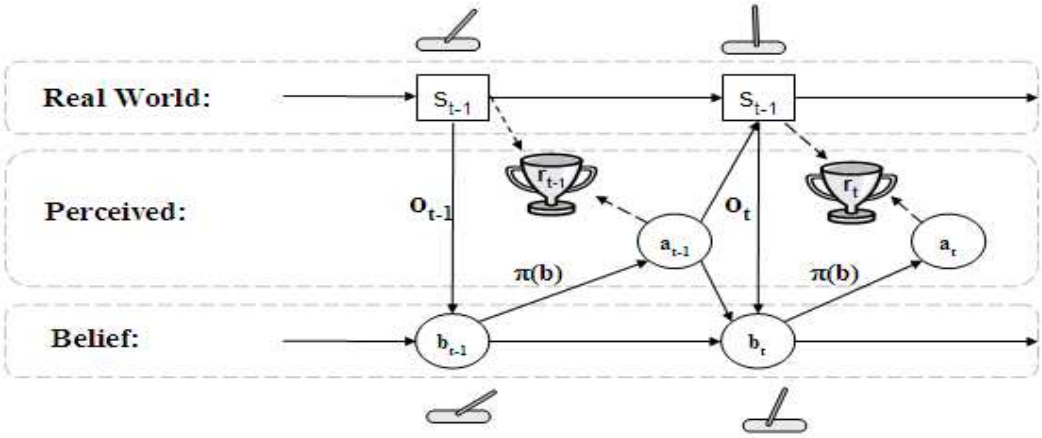

Figure 2.2. Graphical Representation of POMDP

Figure 2.2 shows a graphical outline of POMDP framework. Suppose the agent made the action a_{t-1} at time $t-1$ according to its policy π. It was rewarded by r_{t-1} and observed o_t. Using these values it updates its internal belief, b_{t-1} to b_t using its observations (what is seen). The real world states (what really goes on) are not fully observable by the agent, rather it keeps an internal interpretation of these states, the *belief* (what is inferred) which is a sufficient statistic in order to decide an action.

The formal definition of POMDP is given below (Kaelbling et al., 1996). A POMDP consists of 6-tuple (S, A, Θ, T, O, R) where :

- S is the set of states,
- A is the set of actions,
- Θ is the set of observations,
- R is the reward function $S \times A \rightarrow R$,
- O is the set of observation probabilities that describe the relationships among observation states and actions $S \times A \times \Theta \rightarrow [0, 1]$,
- T is the state transition function $S \times A \rightarrow \pi(S)$, where a member of $\pi(S)$ is a probability distribution over the set S.

The goal of the agent is to learn a policy π which maps the observation history $h_{1-t} = [(O_1, A_1, R_1), \ldots, (O_{t-1}, A_{t-1}, R_{t-1}), (O_t, A_t, R_t)]$ into an action A_t to maximize π's quality of value. If the policy is stochastic then this returns a probability

distribution over actions.

2.2.1. Belief States

For POMDP approaches, the agent's state is not fully observable; the agent tries to convey uncertain or incomplete information about the world's state. The agent remembers past observations up to some finite number of observations, called a *history*.

Aström developed an alternative to storing histories: a belief state (Aström, 1965), which is the probability distribution over S, summarizing the agent's knowledge about the current state. The belief state is sufficient in the sense that the agent can learn the same policies as if it had access to $h_{1...t}$ (Kaelbling *et al.* 1996). The notation $b_t(s)$ is the probability that the world is in state s at time t. Knowing the belief state $b_t(s)$ such that$\sum_{s\in S} b_t(s) = 1$, an action a and an observation, θ, the successor belief state can be computed by:

$$b_{t+1}(s) = \frac{\sum_s b_t(s)P(s'\,|s,a)P(\theta\,|s,a\,,s')}{\sum_{s,s'} b_t(s)P(s'\,|s,a)P(\theta\,|s,a\,,s')} \tag{2.9}$$

Equation (2.9) is equivalent to Bayes filter, and its continuous case forms the basis of Kalman filters (Kalman 1960). The belief state update can become challenging in environments with too many states. Efficient methods of belief state calculation are summarized later in this chapter.

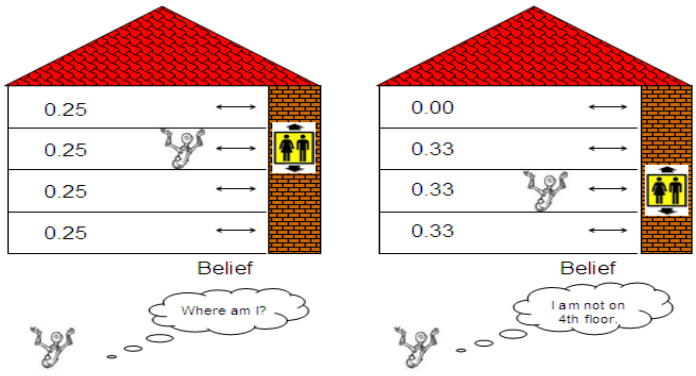

Figure 2.3. The agent updates its belief state after each action (Peshkin 1999).

The agent's momentary belief is represented by a uniform probability distribution over the space of all possible locations. This is illustrated in Figure 2.3. Initially the agent observes that it is at one of the four floors. When the agent uses the elevator it updates its belief depending on its last action. Suppose the agent has moved down so it may conclude that it is not on the highest floor, decreasing its belief of being of the fourth floor and increasing its belief of being on other floors evenly.

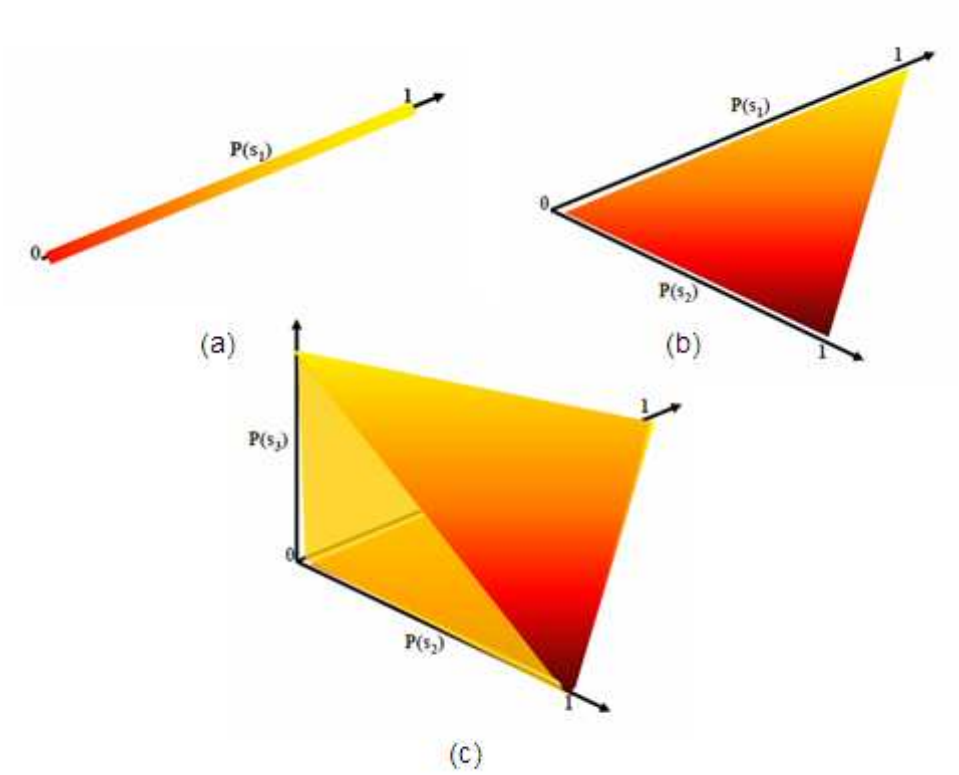

Figure 2.4. Belief state's dimension is one less the number of states, where (a) is the one dimensional space with two states, (b) is the two dimensional space with three states and (c) is the three dimensional space with four states.

The dimension of the belief state is one less the number of revealed states. Figure 2.4 shows some sketches. In this research, since we work with continuous state space, the belief space is also continuous, as in the discrete case, but now with an infinite number of dimensions.

2.2.2. Gaussian Based Belief State Representation

Assume a one dimensional maze as shown in Figure 2.5. For a discrete POMDP, initially the agent is in either one of the four cells, and the agent's belief can be represented as a vector given in Equation (2.10).

$$b = [0.25 \quad 0.25 \quad 0.25 \quad 0.25] \tag{2.10}$$

Figure 2.5. One-dimensional maze.

In the continuous state case it is impossible to represent each belief as a vector, since there are infinitely many states. So we have assumed the belief as a Gaussian probability distribution where the mean is the most probable current state and the covariance is the state estimation error or uncertainty. Belief formulation at step t is given in Equation (2.11), where s_t is the most probable current state and P_t is the covariance.

$$b_t(s) = N(s_t, P_t) \tag{2.11}$$

In the discrete case, the belief state space dimension is one less than the number of states as shown in Figure 2.6(a). It is normally expected that for the continuous case since the number of states is infinite, the dimension of the belief state space is also infinite. Using a parametric distribution, i.e. Gaussian, for the belief state representation reduces the infinite dimensions to two dimensions as shown in Figure 2.6(b). In our experiments, the initial belief point covariance or uncertainty is $P_0 = P_{max}$. At the next decision steps Kalman filter decreases the uncertainty level until $P_{converge}$. Therefore with 99% probability, the current state is within $2.5 \times \sqrt{P}$ neighborhood of the most probable current state. Although $P_{converge}$ is always greater than 0 when an observation noise exists, the case where P is equal to 0 is given in the figure since we calculate the reward function from the definition for the MDP case.

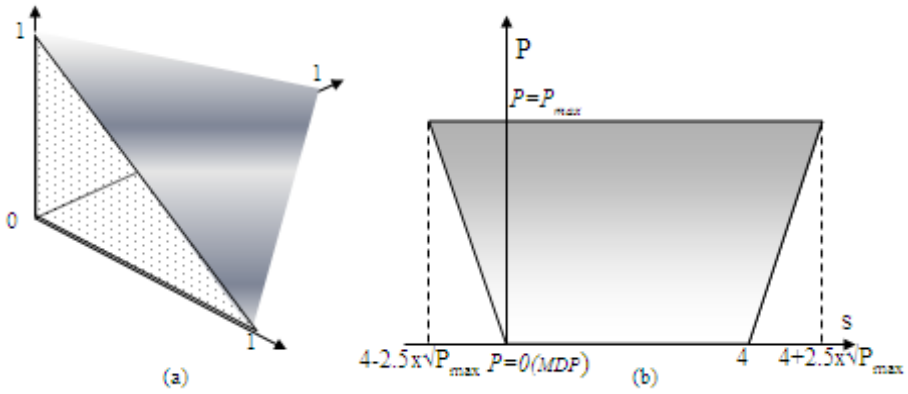

Figure 2.6. One dimensional maze belief state space representations: (a) discrete case (b) continuous case with parametric distribution.

In the POMDP approaches, the agent's state is not fully observable; the agent tries to convey uncertain or incomplete information about the state of the world. The agent remembers past observations up to some finite number of observations, called a history denoted by the set, $\{h_{t-1}, h_{t-2}, \ldots, h_{t-n}\}$.

Aström conceived an alternative to storing histories: belief state (Aström 1965), which is the probability distribution over the state space, S, summarizing the agent's knowledge about the current state. The belief state is sufficient in the sense that the agent can learn the same policies as if it had access to history (Kaelbling *et al.* 1996).

$$\sum_{s \in S} b_t(s) = 1 \tag{2.12}$$

Here $b_t(s)$ is the probability that the world is in state s at time t. Knowing $b_t(s)$ in Equation (2.12) , an action, a and an observation, o, the successor belief state can be computed by Equation (2.13).

$$b_{t+1}(s) = \frac{\sum_s b_t(s) P(s'|s,a) P(o|s,a,s')}{\sum_{s,s'} b_t(s) P(s'|s,a) P(o|s,a,s')} \tag{2.13}$$

Equation (2.13) is equivalent to Bayes filter, and its continuous case forms the basis of Kalman filters. The Kalman filter described in section 2.3 addresses the general problem of trying to estimate the state of the agent. We will use the Kalman filter for

belief state estimate and update.

2.2.3. Value Function

A policy is simply a mapping from beliefs to actions and may be represented as a decision tree over belief ranges. Let us demonstrate this with an example. Imagine a clinic where a patient applies with symptoms of sneezing and fatigue. Dr. John has to diagnose and suspects that the patient suffers from either flu or allergy. For this POMDP problem we have two cases: having flu or having allergy. In order to make the right decision, Dr. John may require tests. After each test, he may require additional tests or may write a prescription. The decision tree of Dr. John is given in Figure 2.7 where each ellipse represents an action that is valid for a belief range over having flu given in parenthesis. The top ellipse represents the case when Dr. John's belief that the patient's sickness is flu is between 0.4 and 0.6. For this belief state, Dr. John's decision is to require additional tests.

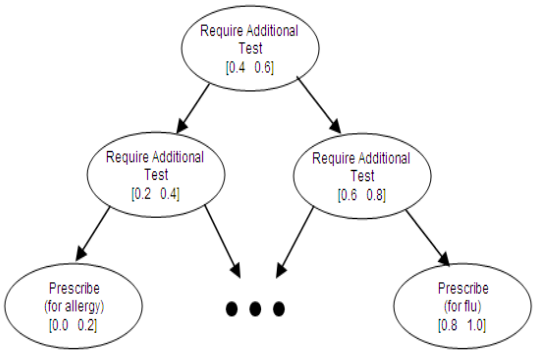

Figure 2.7. The agent updates its belief state after each action.

If an n-step policy, π, represented by a decision tree, collects maximum reward in the long term, it is called the *optimum policy*, π^*. Each node in the decision tree has a value representing the benefit of choosing the action for that belief range. The benefit calculated by the value function for discrete POMDPs given in Equation (2.14) and Equation (2.15).

$$V_t^{\pi}(b) = \max_a Q_t^{\pi}(b, a) \tag{2.14}$$

where $Q_t^{\pi}(b,a)$ is the value of performing action a at a belief state b at time step t. For a POMDP problem value function is calculated for the belief state rather than the MDP state. Moreover, the transition function now includes the observation probabilities. The calculation of this value is given in Equation (2.15).

$$Q_t^{\pi}(b,a) = E_{\pi}\left\{\sum_{k=0}^{n-1}\gamma^k r_{t+k+1}|b_t = b, a_t = a\right\} \tag{2.15}$$

where n is the planning horizon, γ is the discount factor, $0 < \gamma < 1.0$. Another recursive representation of Equation (2.15) is given in Equation(2.16), where the current value function is recursively calculated from the next step's value function.

$$Q_t(b,a) = r_t(b,a) + \gamma \sum_{o} p(o|b,a)V_{t+1} \tag{2.16}$$

In this research, we will use the discrete value function given in Equation (2.16), where in continuous version summations are modified to integrations given in Equation (2.17). If we substitute Q in Equation (2.14) with Equation (2.17), the value calculation will be as given in Equation (2.18). Obviously being able to use the discrete version will simplify value calculations, reducing computational complexity.

$$Q_t(b,a) = r_t(b,a) + \gamma \int_{o} p(o|b,a)V_{t+1} \tag{2.17}$$

$$V_t(b) = \max_{a}(r_t(b,a) + \gamma \int_{o} p(o|b,a)V_{t+1}) \tag{2.18}$$

2.2.3.1. Value Function Calculation with Temporal Difference Learning. Temporal difference (TD) learning is a prediction method for solving a reinforcement learning problem. "*TD learning is a combination of Monte Carlo ideas and dynamic programming ideas*" (Sutton 1999). TD resembles a Monte Carlo method because it learns by sampling the environment according to some policy. TD is related to dynamic programming techniques because it approximates its current state value estimate based on previously

learned estimates (a process known as *bootstrapping*).

Since consecutive predictions are most probably correlated, TD learning basically adjust current state value estimates to match other, more accurate predictions, about the future. Sutton summarizes the idea by the following example (Sutton 1988).

"Suppose you wish to predict the weather for Saturday and that you have some model that predicts Saturday's weather given the weather of each day in the week. In the standard case, you would wait until Saturday and then adjust all your models. However, when it is, for example Friday, you should have a pretty good idea of what the weather would be on Saturday - and thus be able to change, say, Monday's model before Saturday arrives."

In our algortihm we use TD($\lambda = 0$) Learning where the state value estimation is affected by one step future. TD($0 \leq \lambda < 1$) the state value estimation takes into account more steps further in the future. In TD($\lambda = 0$) approach, as in value iteration approach, expected discounted reward in Equation (2.2) is optimised, using the formulation given in Equation (2.19), where α is the learning rate and $b^{'}$ is the next belief state.

$$V_t(b) = V_t(b) + \alpha[r(b) + \gamma V_{t+1}(b') - V_t(b)] \tag{2.19}$$

2.2.4. POMDP Agent Types

The controller of a POMDP agent can be categorized along the following axes and a sketch of agent types is given in Figure 2.8:

1. **World Model Requirement:** Since Q-values associate directly the state with the action, the agent does not need to know the entire world model. The current state knowledge is sufficient in order to decide the current action. However, if the agent can only access the value function or value of the states, then it will need the world model, specially the transition model in order to decide.

2. **Internal State Maintained:** The agent's prediction accuracy about the current state, is crucial to the survival and accomplishment of the task.
   (a) **Reactive (memoryless) agent:** $S_t = \Theta_t$. The agent in this case assumes the observation represents the state, and ignores the possibility of state confusion.
   (b) **Agent with a finite fixed length window of past observations:** $S_t = h_{t-k}$. The idea is to keep the relevant information necessary to accomplish the task. For example, in the five step load unload problem, the agent has to keep the last visited station for five steps in order to decide in the right move.
   (c) **Agent that uses decision trees:** S_t is a *suffix tree*. This is a variation of a decision tree used to determine the most probable state.
   (d) **Agent that uses neural networks:** S_t is a recurrent neural network. This is a straight forward approach where the observations are fed as input to the network, and mapped to internal state space representation. It suffers from the use of back propagation techniques. In our research, the ARKAQ-learning used recurrent ART-2A networks (Carpenter and Grossberg, 1987b) which has a totally different tuning technique.
   (e) **Agent that uses belief states:** S_t is a belief state. It is summarizes the history of the agent, and used instead because of low space requirement.
   (f) **Agent that uses Finite State Machines:** S_t is a Finite State Machine (FSM). Actually, the policy graph is represented by a FSM where the nodes represent states and labeled with an optimal action and the transitions represent the possible observations. Although, this representation is valid for decision type of POMDP problems like tiger, in this research, we noticed that in navigational type of POMDP problems where the observation is a function of the most probable state the transitions from the belief state nodes may be labeled by the action.
3. **Policy Representation and Learning:**
   (a) **Reinforcement Learning:** The agent learns $Q(s, a)$ using some reinforcement technique. Note that this may not work since the environment might not be Markov in S. If S is continuous, one may need to use function

approximators to represent Q.

(b) **Exact Value Iteration:** If the POMDP is known one can convert it to a belief state MDP, and compute V for that. This is the optimal approach if the model is known, but is often computationally intractable. To avoid this, the approximation methods of V or the belief state, or both may be considered.

(c) **Approximate Methods:** If POMDP is known, one can solve the underlying fully observed MDP, and use that as the basis of various heuristics. One example computes the most likely state and act accordingly (Cassandra, 1998).

(d) **Policy Search:** If one has the ability to compute $V(\pi)$ or $\frac{\partial V(\pi)}{\partial W_\pi}$, where W_π are the parameters of the policy, then he can perform a (local) search for the best controller.

In this research we have combined 2(d), 2(e), 2(f) and 3(a) approaches into our algorithms. Detailed information can be found in chapters 3, 4 and 5.

2.2.5. POMDP Solution Techniques

The methods for learning within the POMDP framework may be divided as exact methods, which are intractable and approximate methods. There exist two versions of the POMDP learning problem: learning when a model of the POMDP is known, and the much harder problem learning when a model is not available. The methods used to solve POMDP's are referred to as reinforcement learning algorithms, since the only feedback to the agent is a scalar reinforcement signal.

2.2.6. Learning with a model

The methods for producing POMDP policies when the model of the POMDP is known include exact methods which are guaranteed to find the optimal policy. The exact methods suffer from both curses of POMDP problems, meaning that their computational complexity grows exponentially with the number of observations. In our

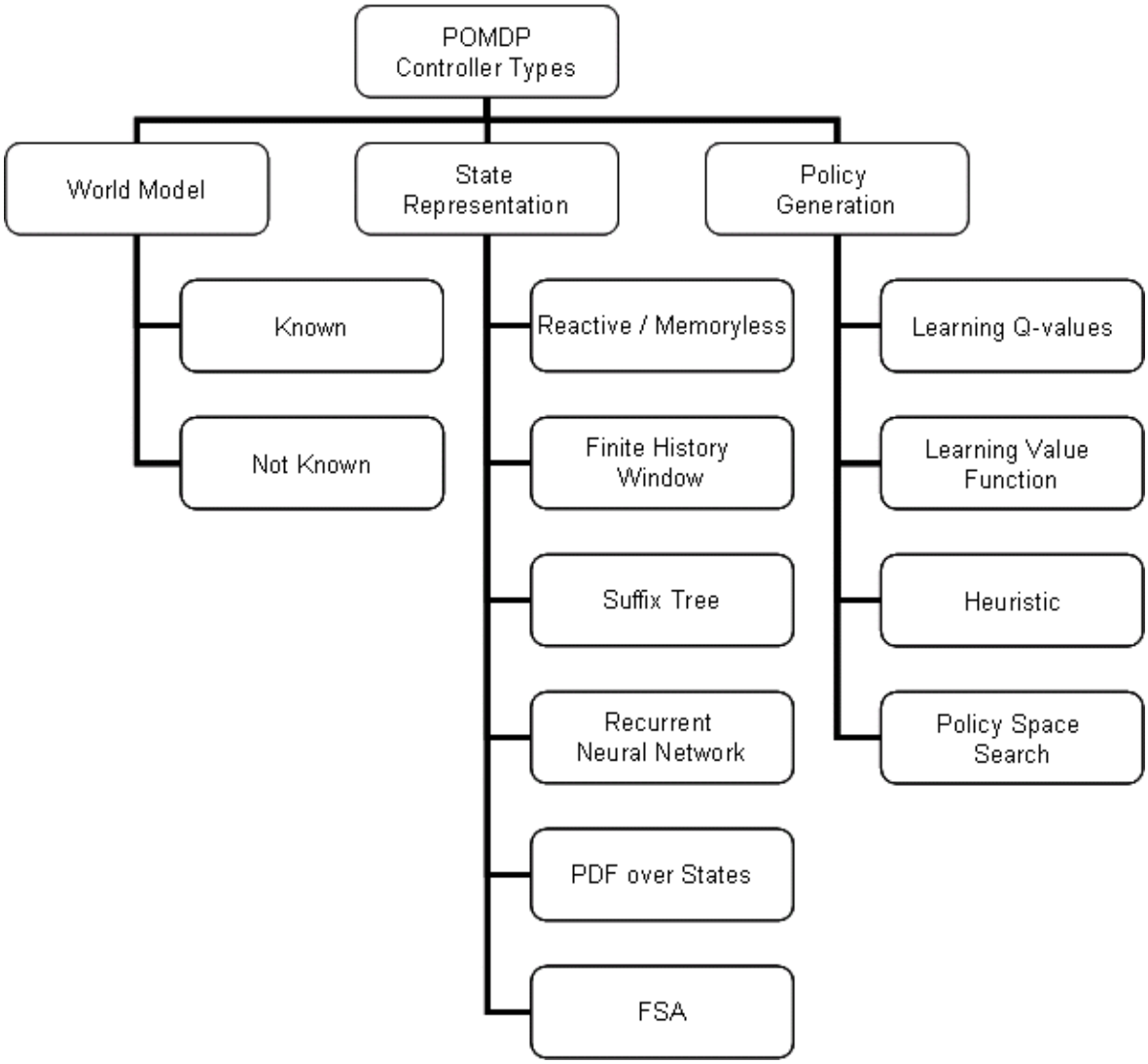

Figure 2.8. POMDP Agent Types

case, we assumed the observation space infinite making such approaches infeasible.

2.2.6.1. Exact Methods. The observations of an agent may not reveal the state to choose the optimal action. A simple example is the packing task, which involves four steps: opening a box, putting a gift into it, closing it, and sealing it. An agent driven only by its current visual percepts cannot accomplish this task, because when facing a closed box, the agent does not know if a gift is already in the box and therefore cannot decide whether to seal or open the box. The solution is to keep history of the past observations and actions h_{1-t} to allow the agent to determine the true state. This history approach is used in the form of distinction trees and prediction suffix trees (Ron et al., 1994).

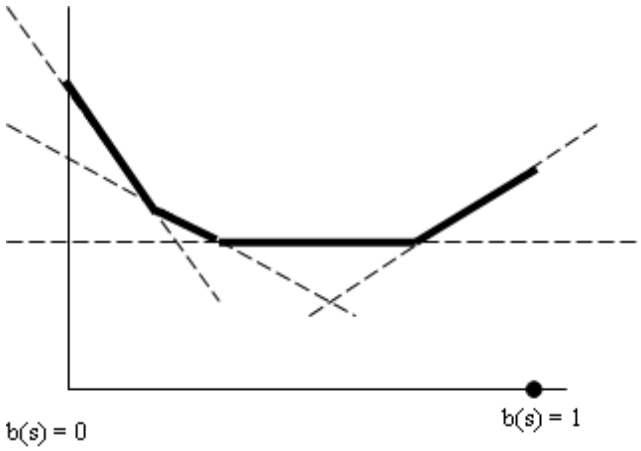

Figure 2.9. Linear and convex value function for $|S|$=2

Let B be the uncountable set of belief states the system can reach. By replacing the states in Equation (2.5) with belief states (Aström 1965, Cassandra et al., 1994), we obtain,

$$V(b) = \max_a \left(R(b,a) + \gamma \sum_{b' \in B} T(b,a,b') V(b') \right), \forall b \in B \tag{2.20}$$

Using value iteration, Equation (2.20) converges in finite time to within ε of the optimal policy value (Lovejoy, 1991). Maintaining and updating values for each belief state is intractable. Fortunately, it has been shown that the value over all belief states can be represented exactly by a piecewise convex linear function (Smallwood 1973) as shown for a two state system in Figure 2.9. This function is given in Equation (2.21), where the set L =$\{l_1, l_2, \ldots, l_n\}$ is the set of hyper planes for representing value function.

$$V(b) = \max_{l \in L} b.l \tag{2.21}$$

Constructing the value function proceeds using value iteration on belief states by choosing a number of points in the belief space that will be sufficient to find all parts of the piecewise linear convex value function. The algorithms differ depending on how they explore the belief space to determine the necessary belief points for constructing the value function. This is often done by solving a set of linear equations.

The simplest exact method is Monahan's enumeration algorithm (Monahan, 1982), which enumerates all hyper planes, and then eliminates useless hyper planes by solving a set of linear equations. In order of increasing efficiency, other algorithms include Sondik' s One-Pass algorithm (Sondik 1978), Cheng's Linear Support (Cheng, 1988), Littman et al's Witness algorithm (Cassandra 1994).

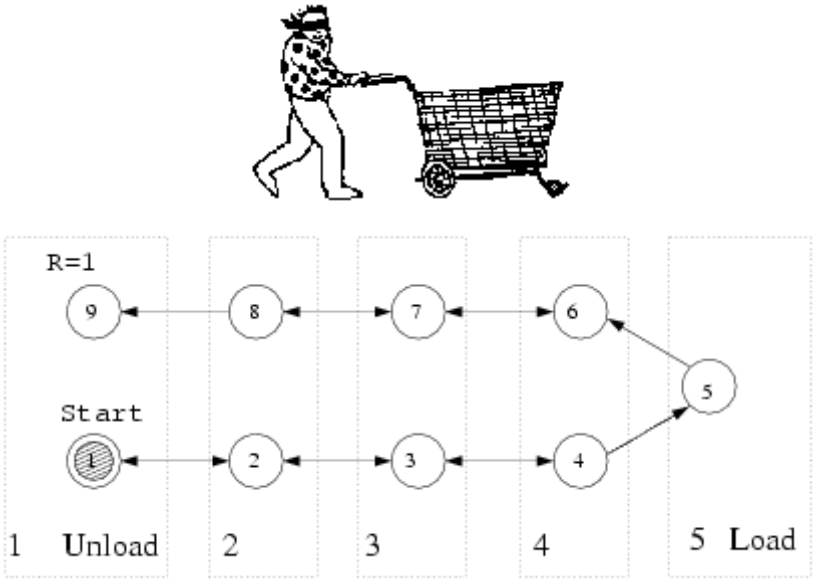

Figure 2.10. The state transition diagram of the load unload problem.

2.2.6.2. Finite State Controller Approaches. For some discrete POMDP, the optimal controller can be represented by a finite state machine, namely a policy graph which is a directed and possibly cyclic graph where each node is labeled with a single action and transitions out of each node are labeled with observations. Each node in the policy graph maps to a partition in the value function (Cassandra 1994). If the optimal policy can be represented by a cyclic graph with a finite number of nodes, then the policy is called *finitely transient* (Sondik 1978). In this case, it is possible for large infinite-horizon POMDP's to be controlled by a simple policy graph. Consider the Load/Unload problem (Peshkin et al., 1999) shown in Figure 2.10. The observations alone do not allow the agent the agent to determine if it should move to left or right while it is in the middle null observation states. However, if the agent remembers whether it last visited the load or unload location then it can act optimally. The optimal policy graph is shown Figure 2.11. This policy graph suffices no matter how many intermediate locations are between the "load" and "unload" locations. As the

number of intermediate locations increases, the value function becomes more complex but the optimal policy graph does not. This example motivates the idea of searching in the space of policy graphs instead of learning value functions.

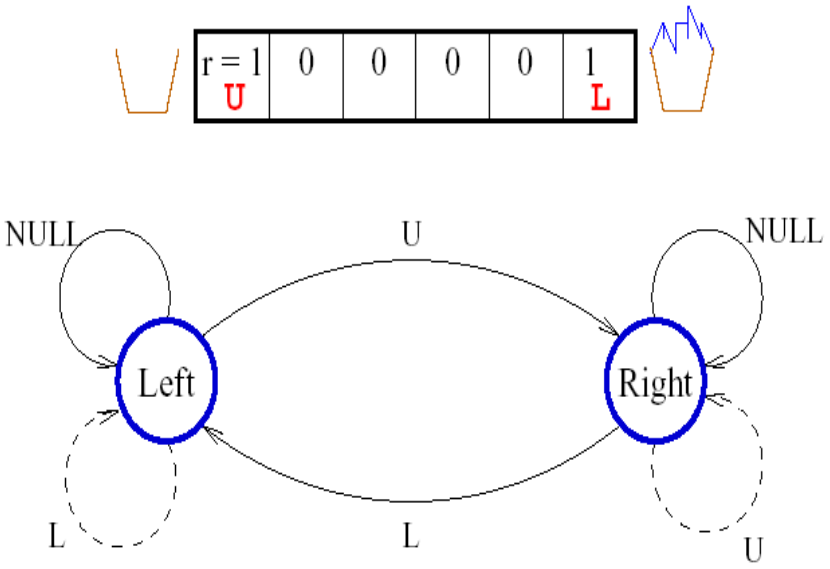

Figure 2.11. Policy graph learned for the Load/Unload problem.

2.2.6.3. Approximate Methods. The approximate methods learn approximations to the exact value function. Some of these methods use heuristics to simplify the representation of $V(b)$ by solving the much simpler underlying MDP. One choice for the policy is to assume that the agent is in the most likely state, and act accordingly, known as the MLS heuristic (Nourbakhsh et al., 1995). This approach ignores the agent's confusion over which state it is in. The voting heuristic (Simmons and Koenig, 1995) weights the vote for the best action in each state by the probability of being in that state. The Q_{MDP} heuristic (Littman et al., 1995) takes into account the belief state for one step then assumes that the underlying $Q(s, a)$ is known (Cassandra 1998). These heuristics will generally perform poorly if the belief state is close to uniform. This motivates choosing actions that decrease the uniformity of belief state in the hope that heuristics above will perform better with peaked belief. For example, consider an agent that must reach the other side of a featureless desert (Roy and Thrun, 2001). If it goes straight across it will quickly get lost due to lack of landmarks and movement errors. The better policy is to skirt the desert, taking longer but remaining certain of reaching the goal because the agent location is not accurate. (Cassandra 1998) shows how to

use the entropy of belief state to switch between information gathering policies and exploitive policies.

Grid methods are another family of approximation methods. The value function over the belief space can be approximated by values at a finite set of points along with an interpolation rule to compute values that do not correspond to grid points. Regular grids are an obvious choice though they fail to scale for large state spaces (Lovejoy 1991). Methods for choosing irregular grids include the use of simulation to determine useful grid points (Hauskrecht, 1997) and adding points where large variations in values are detected for two local points, which have observations in common (Brafman, 1997). Interpolation schemes examples include linear interpolation and kernel regression (Hauskrecht, 2000).

2.2.6.4. Linear Quadratic Gaussian POMDPs. This approach assumes that the cost function is quadratic. The coefficients of the value function expressed as a set of linear equations called Ricatti equations is also quadratic. Szita and Lõrincz in their work (Szita and Lõrincz, 2004) consider applying linear quadratic gaussian solution technique with the assumption of quadratic cost and value function. They further assumed that value function depends on the real world states rather than belief states. This assumption, however, will generate a value function independent of the agent's uncertainty conflicting the basic POMDP approach.

2.2.6.5. Efficient Calculation Of Belief States. Bayesian Networks have been applied to POMDPs by Boutilier and Poole (Boutilier and Poole, 1996) to simplify both the belief state calculation problem and the value function representation problem. However, Boyen and Koller (Boyen and Koller 1998) observed that representing belief states as Bayesian networks can lead to an accumulation of errors over many time steps that result in the divergence of the belief state approximation diverging. They show that projections of the Bayesian network that produce strictly independent groups of state variables results in converging belief states that can recover from errors. With approximate belief state calculation and factoring linear value functions, dynamic programming

techniques become tractable (Hansen and Feng, 2000).

2.2.6.6. Continuous State and/or Observation Space. Discrete worlds have the advantage that distributions over belief states can be represented exactly, using one parameter per state. The optimal value function (for finite planning horizon) has been shown to be convex and piecewise linear, which makes it possible to derive exact solutions for discrete POMDP's. In general, continuous POMDP's are not solvable exactly, and little is known about the special cases that can be solved.

Porta, Spaan and Vlassis (2005) assume discrete observation over continuous state space. Hoey and Poupart (2005) assume discrete state space with continuous observation. In both works, since the value function is still PWLC, they have built their solution techniques on this theorem. In the fully continuous case, the problem of representing the value function is made more difficult than the discrete case because inputs are continuous distributions.

Brooks et al (2005) consider robot-navigation POMDP problems. They have not mentioned how to calculate belief updates and further assumed that the reward may be represented as a mixture of gaussians, which is actually not correct. They solve robot-navigation POMDP using grid based methods. However, grid-based approximations suffer from the curse of dimensionality: the number of samples required to achieve a given sample density is exponential in the number of states.

2.2.7. Learning without a model

These types of algorithms assume the POMDP parameters are not known. To learn policies the agent interacts with the world, and compares the samples of different policies. Given enough samples, rewards can be correlated with actions and the probability of choosing actions increases to maximize reward. Below we describe the main approaches.

2.2.7.1. Reactive Agents. The simplest type of agent does not maintain any internal state, i.e., it is memoryless. The agent assumes $S_t = \theta_t$. Littman (1994) showed that the problem of finding optimal deterministic reactive policy for a discrete POMDP is in general intractable. Singh et al (1994) showed that a deterministic reactive policy can do arbitrarily worse than a stochastic reactive policy. The problem with reactive policies is that two states may appear the same but in fact can be different. If the underlying states require different actions, this can lead to unstable policies. The agents, that use more than the current observation for making decisions, may have the chance of finding stable policies.

2.2.7.2. Finite History Windows.. Most of the algorithms typically use the last k observations as input to the policy. For discrete observations, these histories can be arranged into a suffix tree, a particular type of data structure that lends itself greatly to the solution of the exact string matching.

A basic definition of the suffix tree data structure is as follows. If $text = t_1t_2...t_i...t_n$ is a string, then T_i=t_it_{i+1} ... t_n is the *suffix* of *text* that starts at position i. These suffixes are then assembled into a tree. From this tree, simple searches can be made starting at the root and following the branches that match the particular desired pattern.

The "Utile Suffix Memory" algorithm uses a suffix tree with variable depth (McCallum, 1995). The "U-Tree" algorithm is a slight extension where the observation is treated as a vector, and different branches can be created depending on the value of "utility test" (McCallum, 1996). The algorithm, although very successful in large discrete spaces, keeps the instances of previous histories at the leaves and has a high space complexity.

2.2.7.3. Neural Networks. Window-Q uses a neural network to learn Q-values where the inputs are the last k observations and actions (Long–Ji and Mitchell, 1992). Recurrent-Q augments the output of a neural network with continuous state outputs, which are

fed back into a previous layer of network (Long 1992). The network has a continuous internal-state space allowing a POMDP to be controlled by presenting observations as inputs and interpreting the outputs as action distributions. Training algorithms include Back-propagation through time (Rumelhart et al., 1986), which has difficulty in learning long-term memory because the back propagated error signals tend to blow up or shrink to nothing depending on the feedback weights.

2.2.7.4. Internal State Controllers. The idea behind *internal state controllers* (ISC) is that past events, which are relevant to choosing optimal actions, can be remembered indefinitely by a directed cyclic graph of internal states. The internal states serves as the agent memory and the agent learns to control the memory in such a way that the hidden state relevant to maximizing performance is revealed. Learning ISCs is sometimes considered synonymous with learning policy graphs, however, policy graphs are usually assumed to have nodes with deterministic actions whereas ISCs have distributions over actions (Peshkin et al 1999).

2.2.8. Multi-Agent Problems

All the algorithms described so far can be extended to multiple agents trivially. The idea is to alter the set of actions A such that it contains the cross product of all the actions available to each agent, $A = A_1 \times A_2 \times \ldots A_n$. If the agent parameters are independent, then each agent independently chooses actions, which are combined to form meta-action (Bartlett and Baxter, 2000).

2.3. Kalman Filtering

The Kalman filter (Kalman 1960, Maybeck, 1979, Grewal and Andrews, 1993) addresses the general problem of trying to estimate the state of the agent. Nearly all algorithms that exist for spatial reasoning make use of this approach. It estimates the agent's current state recursively based on the previous states.

The Kalman filter addresses the general problem of trying to estimate the state of a discrete-time controlled process that is governed by the two stochastic equations, the state equation given in Equation (2.22)

$$s_t = f(s_{t-1}, a_{t-1}, v_{t-1}) \tag{2.22}$$

and the measurement equation given in Equation (2.23).

$$o_t = h(s_t, n_t) \tag{2.23}$$

Remember the graphical representation of POMDPs, in Figure 2.2. The Kalman filter relates the current state, s_t, to previous state, s_{t-1}, and action, disturbed by actuator noise using a state transition equation. Likewise, the observation is a function of the current state disturbed by sensor noise. Both the actuator and the sensor suite of real world agents are subject to some noise. Kalman assumes that these are Gaussian as given in Equations (2.24) and (2.25). In practice, the process noise (v) covariance, Q, and measurement noise (n) covariance, R, matrices might change with each time step or measurement, however in this research we assume they are constant.

$$p(v) \sim N(0,\ Q) \tag{2.24}$$

$$p(n) \sim N(0,\ R) \tag{2.25}$$

In this research, we use a linearized the motion model. The general linearized motion model equation is given in Equation (2.26).

$$s(t) = As_{t-1} + Ba_{t-1} + v_{t-1} \tag{2.26}$$

Likewise, we have also linearized the sensor model. The general linearized sensor model

equation is given in Equation (2.27).

$$o_t = Hs_t + n_t \tag{2.27}$$

The matrix A in Equation (2.26) relates the state at the previous time step to the state at the current step, in the absence of either a driving function or process noise. Note that in practice A might change with each time step, but here we assume it is constant. The matrix B relates the optional control input to the state s. The matrix H in the measurement Equation (2.27) relates the state to the measurement o. In practice H might change with each time step or measurement, but in this research we assumed it to be constant.

To compute the minimum mean square error estimate of the state and covariance the following equations are used. Equation (2.28) is the estimate of the state variables,

$$s_{t|t-1} = As_{t-1} + Ba_{t-1} \tag{2.28}$$

Equation (2.29) is the estimate of the sensor reading,

$$o_{t|t-1} = Hs_{t|t-1} \tag{2.29}$$

Equation (2.30) is the covariance matrix for the state P,

$$P_{t|t-1} = AP_{t-1}A^T + Q \tag{2.30}$$

and Equation (2.31) is the covariance matrix for the sensors.

$$B_{t|t-1} = HP_tH^T + R \tag{2.31}$$

The state and covariance together form our belief state. At each time step the state and covariance is calculated recursively based on previous calculations. The Kalman filter divides these calculations into two groups: time update equations and measurement

update equations. The time update equations are responsible for projecting forward (in time) the current state and error covariance estimates to obtain the a priori estimates for the next time step. The measurement update equations are responsible for the feedback, i.e. for incorporating a new measurement into the a priori estimate to obtain an improved a posteriori estimate. The time update equations can also be thought of as predictor equations, while the measurement update equations can be thought of as corrector equations. Indeed the final estimation algorithm resembles that of a predictor-corrector algorithm for solving numerical problems as shown below in Figure 2.12.

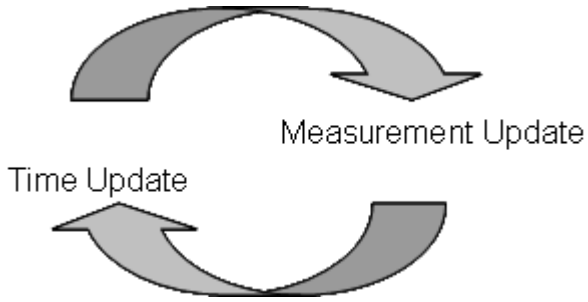

Figure 2.12. The recursive structure of Kalman Filter.

The specific equations of time update functions are given through equations (2.28) to (2.30), and the specific equations of measurement update functions are given below:

$$K_t = P_t^- H^T (H B_t^- H^T)^{-1} \tag{2.32}$$

$$s_t = s_t^- + K_t(o_t - H s_t^-) \tag{2.33}$$

$$P_t = (I - K_t H) P_t^- \tag{2.34}$$

K_t, the Kalman gain, determines the degree of estimate correction. It serves to correctly propagate or "magnify" only the relevant component of the measurement information. The detail of the derivation of Kalman gain can be found in (Maybeck 1979).

After each time and measurement update pair, the process is repeated with the previous *a posteriori* estimates used to project or predict the new a priori estimates. This recursive nature is one of the very appealing features of the Kalman filter; making it very suitable for POMDP problems which has to keep history. Moreover, it keeps the history implicitly by summarizing all the previous states to the current state estimate and its covariance. This is obviously superior to some approaches that keep time window over past history.

2.4. State Segmentation

In the continuous state space domain, the value function is computed using integrals and is not piecewise linear convex as in the discerete case. The integral cannot be computed in closed form, even when the computation is possible, since there are infinite number of belief points. In ARKAQ-learning, instead of learning a value function for all belief points, we decided to group spatially connected states that share common features, and represent each group by a feature vector. We call this process "State Segmentation" and the feature vector which represents the group "State Center". Likewise in KBVI we keep belief point backups which are representative points of the approximately optimal policy. KAFAQ-learning uses a different technique that keeps and updates spatially connected belief segment boundaries that share a common best action.

For "State Segmentation" process ARKAQ-learning algorithm uses an online unsupervised clustering technique namely adaptive resonance theory (ART) (Carpenter and Grossberg 1987a), which has superior features like "remain plastic and at the same time be elastic" compared to other clustering methods for the problems in our experiments.

Since ART models are inspired from a simpler type of adaptive pattern recognition network, we will also discuss competitive learning.

2.4.1. Competitive Learning

A learning system should adapt to significant changes and remain stable in response to irrelevant events. This is known as *stability-plasticity dilemma*. The stability-plasticity dilemma asks: How does the agent know to switch between its stable and its plastic modes to achieve stability without rigidity and plasticity without chaos? How can it preserve its previously learned knowledge while continuing to learn new things? What prevents the new learning from washing away the memories of prior learning? As an example, imagine that you grew up in Kastamonu before moving to Istanbul, but periodically return to Kastamonu to visit your parents. Although you may need to learn many new things in order to live in Istanbul, these new learning experiences do not prevent you from remembering how to find your parent's house in Kastamonu.

In the absence of a self-stabilization mechanism, an external teacher must act as the system's front end to independently recognize the inputs and make the decision. In this case, the external teacher must be able to carry out the recognition tasks that the learning system was supposed to carry out. Hence, non-self-stabilizing learning systems are not capable of functioning autonomously in stochastic environments. In learning systems that need an external teacher to supply the correct representation to be learned, the learning process is driven by the mismatches between desired and actual outputs. Such schemes must learn slowly, or risk unstable oscillations in response to the mismatch. These learning models also tend to be trapped in a local minima, or globally incorrect solutions.

In a *competitive learning model* as shown in Figure 2.13, a stream of input patterns to a network F_1 can train the adaptive weights, or *long-term memory* (LTM) traces, that multiply the signals on the path from F_1 to a coding level F_2. Input patterns to F_1 are normalized before passing through the adaptive filter defined by the paths from F_1 to F_2. Level F_2 is designed as a competitive network capable of choosing the node, which receives the largest total input ("winner-take-all"). The winning population then triggers associative pattern learning within the vector of LTM traces, which sent its inputs through the adaptive filter.

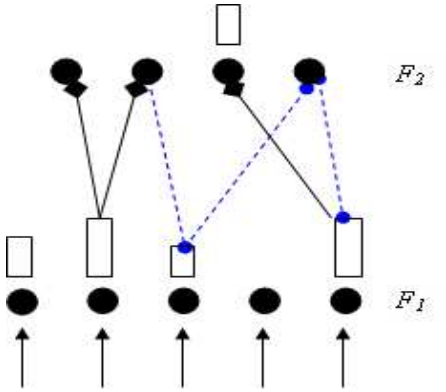

Figure 2.13. Stages of bottom-up activation

The input pattern I generates a pattern of *short-term memory* (STM) activation X=$\{x_1, x_2, \ldots, x_M\}$ across F_1. Sufficiently active F_1 nodes emit bottom-up signals to F_2. The signal pattern S is multiplied, or gated, by LTM traces z_{ij} within from F_1 to F_2 paths. The LTM-gated signals are summed before activating their target nodes in F_2. This LTM-gated and summed signal pattern T, where $T_j = \sum_i S_i z_{ji}$, generates a pattern of STM activation $Y = \{x_{M+1}, \ldots, x_N\}$ across F_2.

For example, as in Figure 2.13, let I_i be the input to the i^{th} node v_i of F_1, I= *1,2,. . . ,M*; let x_i be the activity, or STM trace, of v_i; let x_j be the activity, or STM trace, of the jth node v_j of F_2, $j = M+1, \ldots, N$; and let the z_{ij} be the adaptive weight, or LTM trace, of the path from v_i to v_j. Then let the following be the normalized activity of v_i in response to input pattern.

$$x_i = \frac{I_i}{\sum_{k=1}^{M} I_k} \tag{2.35}$$

For simplicity, let the output signal S_i of v_i equal x_i. Let the following be the total signal received at $v_j$from F_1.

$$T_j = \sum_{i=1}^{M} x_i z_{ij} \tag{2.36}$$

and let

$$x_j = \begin{cases} 1 & if \quad T_j > \max(T_k : k \neq j) \\ 0 & if \quad T_j < \max(T_k : k \neq j) \end{cases} \tag{2.37}$$

Equation (2.37) shows the fact that the node x_j in $F2$ which receives the largest signal is chosen for short-term memory storage, and let an update rule in Equation (2.38) specify that only the vector Z_j=$\{z_{1j},\ldots,z_{Mj}\}$ of adaptive weights about the winning node which are changed due learning.

$$\frac{d}{dt} z_{ij} = \varepsilon x_j(-z_{ij} + x_i) \tag{2.38}$$

The vector Z_j learns by reducing the error between itself and the normalized vector X = $\{x_1,\ldots,x_M\}$ in the direction of steepest descent.

2.4.2. Adaptive Resonance Theory

Adaptive resonance theory (ART) (Carpenter and Grossberg 1987a) was developed to model how humans might be able to recognize unexpected patterns and to remember them for future use. The unexpected patterns are interpreted as new cluster centers by ART. One of the goals of ART is to make sure that even if slightly different new instances of the pattern called for some adjustments to this new cluster center, it would retain its basic characteristic in a stable fashion as shown in Figure 2.14. Instability and recording can occur during competitive learning, as illustrated in this simple case of two patterns and two cluster centers. Two patterns, x_1 and x_2, are presented to a 2-2 network represented by two weight vectors. At $t = 0$, w_1 happens to be most aligned with x_1 and hence this pattern belongs to cluster 1; likewise, x_2 is most aligned with w_2 and hence it belongs to cluster 2, as shown at the left. Next, suppose pattern x_1 is presented several times; through the competitive learning weight update rule, w_1 moves to become closer to x_1. Now x_2 is most aligned with w_1, and thus it has changed from class 2 to class 1. Surprisingly, this recording of x_2 occurs even though x_2 was not used for weight update. It is theoretically possible that such recording will occur

numerous times in response to particular sequences of pattern presentations (Duda et al., 2001).

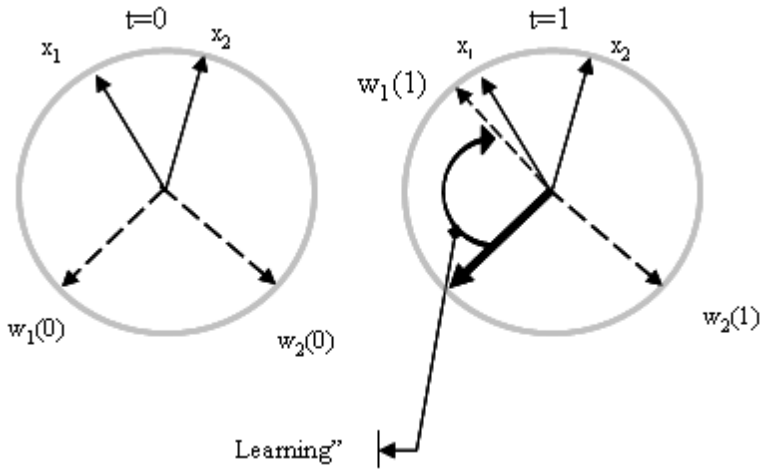

Figure 2.14. Two patterns and two cluster centers.

Since an ART (Carpenter and Grossberg 1987a, Carpenter and Grossberg 1987b) network with short-term memory models recognition of unexpected patterns and can remember these patterns where needed and is designed to learn quickly and stably in response to a possibly non-stationary world, it is used to incrementally segment and distinguish the belief simplex of the agent. Approximate matches, rather than mismatches, drive the learning process in ART. Learning in the approximate match mode enables rapid and stable learning to occur while buffering the system's memory against external noise. The hypothesis testing cycle replaces internal system noise as a scheme for discovering a globally correct solution. It does not use an external teacher. A vigilance parameter, γ, determines the maximum tolerable difference between two patterns in the same category. If this parameter is set too high, it leads to a poor generalization, i.e. slight variations of the same pattern become separate categories. If it is set too low, it leads to classifying dissimilar patterns into the same category, i.e. totally different patterns might be grouped together. The ART architecture is a neural network that self-organizes stable recognition in real time in response to input patterns. Despite the demonstration of input environments, the learning of clusters stabilizes; it is shown, through explicit counterexamples (Carpenter and Grossberg 1987a), that a competitive learning model does not always learn a temporally stable code in response to an arbitrary input environment.

Learning systems that can become unstable in response to many input environments cannot safely be used in autonomous machines that might be unexpectedly confronted by one of these environments. Adaptive resonance theory embeds a competitive learning model into a *self-regulating control structure* whose autonomous learning and recognition proceed stably and efficiently in response to an arbitrary sequence of input patterns.

2.4.3. ART 1

ART1 is an architecture that is capable of stably learning in response to binary input patterns (Carpenter and Grossberg 1987b). Figure 2.15 shows a typical example of the ART 1 architecture. Moreover, the adaptive weights, or LTM traces, of an ART 1 system oscillate at most once during learning in response to an arbitrary input sequence, yet they do not get trapped in local minima.

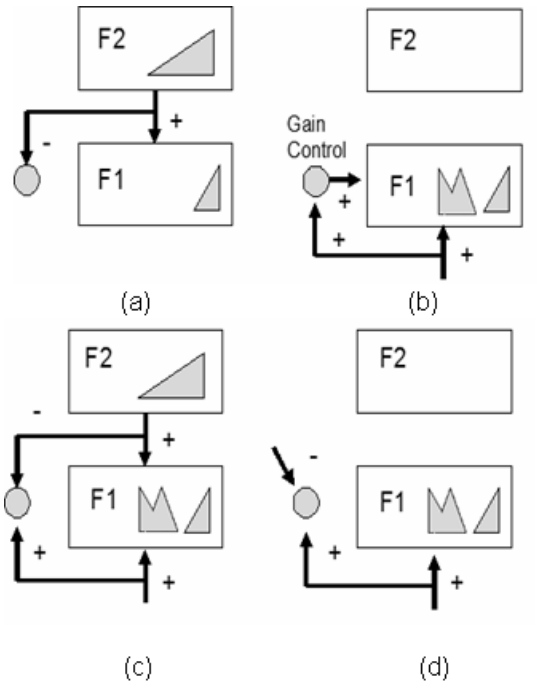

Figure 2.15. Matching by the 2/3 Rule.

As in the competitive learning model, ART architecture encodes a new input pattern by changing the adaptive weights, or LTM traces, of a bottom-up adaptive filter. In an ART network, however, a second, top-down adaptive filter, contained on the path from F_2 to F_1, gives rise to the important property of self-stabilization.

To illustrate how self-stabilization occurs in intuitive terms, suppose that an input pattern I activates F_1. Let F_1 activate in turn the hypothesis, the node at F_2, which receives the largest total signal from F_1. Then F_2, reads out its learned top-down expectation to F_1, where the bottom-up input pattern and the top-down learned expectation are matched across F_1. If these patterns are badly matched, F_1 triggers a reset burst to F_2. This reset burst shuts off the activated node for the remainder of the clustering cycle, and deactivates the top-down expectation controlled by this node. Then F_1 reactivates the same bottom-up signal pattern to F_2 as before. F_2 reinterprets the signal pattern, knowing that the previously activated hypothesis or node was incorrect, and another node is automatically chosen.

The parallel search, or hypothesis testing cycle of bottom-up adaptive filtering from F_1 to F_2, hypothesis selection at F_2, sending a top-down learned expectation from F_2 to F_1, matching at F_1, and reset at F_2 now repeats itself automatically until one of three possibilities occurs:

1. a node is chosen whose top-down expectation approximately matches input I;
2. a previously uncommitted F_2 node is selected; or
3. cannot accommodate input I. Learning in ART occurs when the hypothesis testing cycle comes to an end.

If the hypothesis testing cycle ends in an approximate match, then the bottom-up input pattern and the top-down expectation quickly deform the activity pattern $X = (x_1, x_2, \ldots, x_M)$ across F_1 into a pattern that computes a fusion between the bottom-up and top-down information. Then fusion occurs, the bottom-up and top-down signal patterns mutually reinforce each other via feedback and the system is locked into the resonant state of STM activation. Only then, the LTM traces learn. Thus, the system alters one of its prior learned clusters only if an input pattern is sufficiently similar to what it already knows to risk a further refinement of its knowledge. If the hypothesis testing cycle ends by selecting an uncommitted node at F_2, then the bottom-up and top-down adaptive filters linked to this node learn the F_1 activation pattern generated directly by the input. No top-down alteration of the F_1 activation occurs in that case.

If no match exists, learning is automatically terminated.

One of the key constraints on the design of the ART 1 architecture is its rule for matching a bottom-up input pattern with a top-down expectation at F_1. This rule is called *2/3 Rule*, and regulates both the hypothesis testing cycle and the self-stabilization of an ART 1 system.

In response to an arbitrary bottom-up input pattern, F_1 nodes can be supraliminally activated; that is activated enough to generate output signals to other parts of the network and to initiate hypothesis testing cycle. In response to an arbitrary top-down expectation, F_1nodes are subliminally activated. They attentionally prime F_1 for future input patterns that may or may not generate an approximate match with this expectation. Such a subliminal reaction enables an ART system to anticipate future events and thus to function as an intentional machine. The 2/3 Rule says that at least 2 out of 3 input sources are needed to supraliminally activate an F_1 node. The three are a bottom-up input, a top-down input, and a gain control input. In the top-down processing mode, shown on Figure 2.15(a), each F_1 node receives a signal from at most one input source and is only subliminally activated. In the bottom-up processing mode, shown on Figure 2.15(b), each active bottom-up path can turn on the gain control node, whose output, once on, is independent of the total number of active bottom-up paths. Then all F_1 nodes receive at least a gain control input, but only those nodes that also receive a bottom-up input are supraliminally activated.

When both bottom-up and top-down inputs reach $F1$, as shown in Figure 2.15(c), the gain control source is shut off so that only those $F1$ nodes, which receive top-down configuration of the bottom-up input, are supraliminally activated. The 2/3 Rule maintains supraliminal activity only within the spatial intersection of the bottom-up input pattern an expectation. Consequently if a bottom up input pattern, as shown in Figure 2.15(b), causes the readout of a badly matched top-down expectation, as shown in Figure 2.15(c), then the total number of supraliminally active *F1* nodes can suddenly decrease, thereby causing a decrease in the total output signal emitted by *F1*. This property is used heavily in controlling the hypothesis testing and self-stabilization

processes.

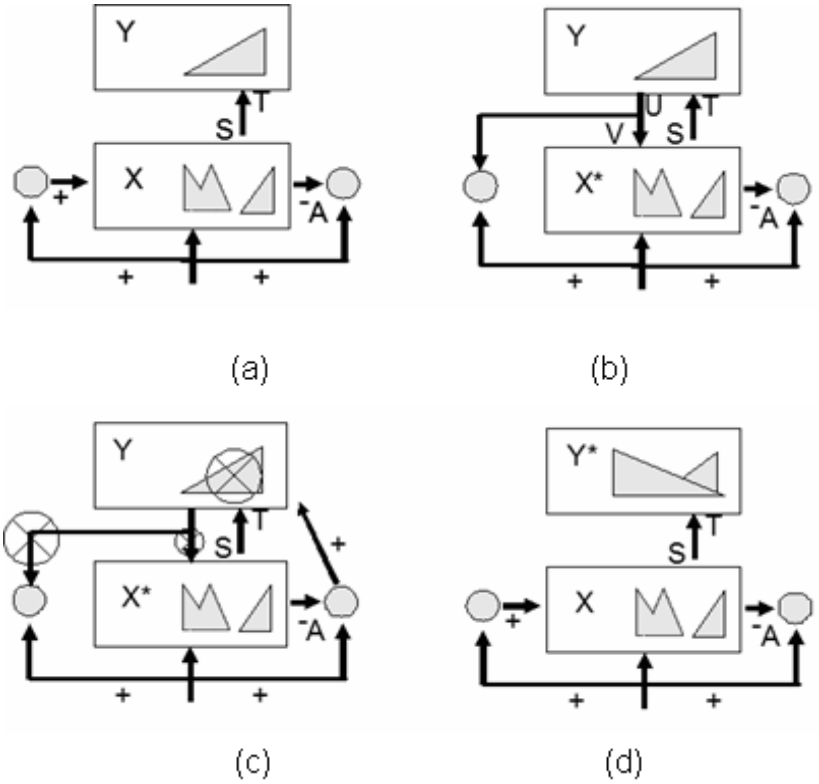

Figure 2.16. ART1 Hypothesis testing cycle.

Figure 2.16 illustrates the interaction levels F_1, F_2 and A. In Figure 2.16(a), the input pattern I generates the STM activity pattern X at F_1 as it activates. Pattern X both inhibits A and generates the bottom-up signal pattern S. Signal pattern S is transformed via the adaptive filter into the input pattern T=ZS, which activates the compressed STM pattern Y across F_2. In (b), pattern Y generates the top-down signal pattern U which is transformed by the top-down adaptive filter V=ZU into the expectation pattern V, if V mismatches I at F_1, then a new STM activity pattern X^* is generated at F_1. The reduction in total STM activity that occurs when X is transformed into X^* causes a decrease in the total inhibition from F_1 to A. In (c), then, the input driven activation of A can release a nonspecific arousal wave to F_2, which resets the STM pattern Y at F_2. In (d), after Y is inhibited, its top-down expectation is eliminated, and X can be reinstated at F_1. Now X once again generates input pattern T to F_2, but since Y remains inhibited T can activate a different STM pattern Y^*at F_2, if the top-down expectation due to Y^* also mismatches I at F_1, then

the rapid search for an appropriate F_2 code continues (Grossberg et al., 1991c).

When a mismatch attenuates STM activity on F_1, the total size of inhibitory signal from F_1 to A is also attenuated. If the attenuation is sufficiently great, inhibition from F_1 to A can no longer prevent the arousal source A from firing Figure 2.16(c) depicts how this inhibition of A releases an arousal burst to F_2, which equally excides all the F_2 cells. The cell population of F_2 reacts to such an arousal signal in a state dependent fashion. In the special case that F_2 chooses a single population for STM storage, the arousal burst selectively inhibits the active population in F_2 and this inhibition is long lasting.

Vector V is the top-down expectation into F_1. Expectation V mismatches input I, significantly inhibiting STM activity at F_1. The amount by which activity in X is attenuated to generate activity pattern X^* depends upon how much of the input pattern I is encoded within the expectation V, via the 2/3 Rule.

In Figure 2.16(c), inhibition of Y leads to removal of the top-down expectation V, and thereby terminates the mismatch between I and V. Input pattern I can thus reinstate the original activity pattern X on F_1 which again generates the output pattern S from F_1 and the input pattern T to F_2. Due to enduring inhibition at F_2, the input pattern T can no longer activate the previous pattern Y at F_2. A new pattern Y^* is generated at F_2 by I.

The activity pattern Y^* generates a new top-down expectation V^*. If a mismatch again occurs at F_1, the orienting subsystem is again engaged, leading to another arousal reset of STM at F_2. In this way, a rapid series of STM matching and reset events may occur. Such an STM matching and reset controls top-down expectation that approximately matches I or that has not yet undergone any prior learning. In the former case the access recognition code is refined based on any novel information contained in the input I. In the ladder case, a new recognition category is established as a new bottom-up cluster and top-down template are learned.

2.4.4. ART 2: Learning In Analog World

Although self-organized recognition of binary patterns is useful in many applications, many other applications require the ability to categorize arbitrary sequence of analog input patterns. A class of architectures, called ART2, has been developed for this purpose (Carpenter and Grossberg 1987b). ART2 Architectures can autonomously classify arbitrary sequences of analog input patterns into categories of arbitrary coarseness while suppressing arbitrary levels of noise. They accomplish this by modifying the ART1 architecture to incorporate solutions of several additional design problems into their circuitry. Three versions of ART2 architecture are now being applied to problems such as visual pattern recognition, speech perception and radar classification.

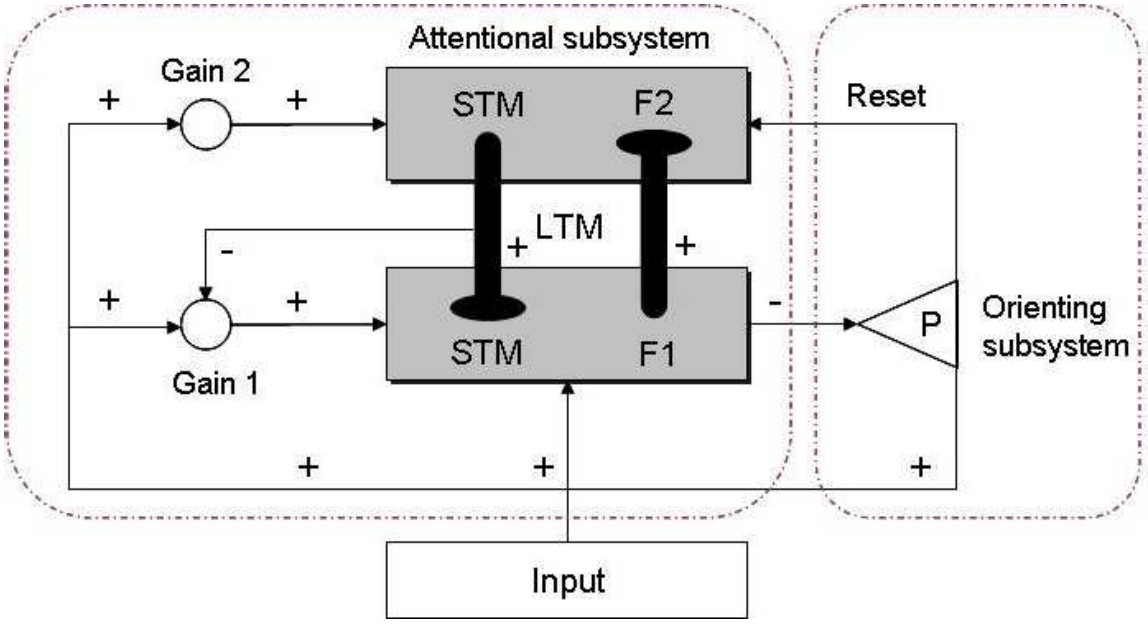

Figure 2.17. Sketch of ART2A architecture.

ART2 is a variant of ART network where analog input patterns can be categorized compared to self-organized recognition of binary patterns. ART 2-A (Carpenter et al., 1991a) is a faster version of ART2 and can be used in large-scale neural computations and was chosen as the network to be used.

The world model generator component of ARKAQ learning uses ART-2A with a heuristic update rule, which is different from ART1. ART1 only works with binary inputs. This is an efficient algorithm that emulates the self-organizing pattern recog-

nition and hypothesis testing properties of the ART2 neural network architecture, but at a speed of two to three orders of magnitude faster.

The update rule adjusts LTM weights in a single step for each presentation interval during the input vector is held constant.

Given a non-uniform m-dimensional vector to F (first layer) the input I^0 to F_1 satisfies:

$$I = RJ_0RI^0 \tag{2.39}$$

where

$$Rx \equiv \frac{x}{\|x\|} \tag{2.40}$$

and

$$(J_0x)_i \equiv \begin{cases} x_i & if \quad x_i > \theta \\ 0 & otherwise \end{cases} \tag{2.41}$$

Threshold θ in Equation (2.41) satisfies the inequalities

$$0 < \theta \leq 1/\sqrt{M} \tag{2.42}$$

Equations (2.39) – (2.41) imply that I is nonzero. The input to the j^{th} F_2 is given by the following equation.

$$T_i = \begin{cases} \alpha \sum_t I_i & if\ j\ is\ an\ uncommited\ node \\ I \cdot z_j^* & if\ j\ is\ an\ commited\ node \end{cases} \tag{2.43}$$

The constant α in Equation (2.43) satisfies the following inequality.

$$\alpha \leq \frac{1}{\sqrt{M}} \tag{2.44}$$

Initially, all F_2 nodes are uncommitted. The set of committed F_2 nodes and the scaled LTM vectors z_j^* are defined iteratively below. The initial choice at F_2 is one node with index j satisfying the following equation.

$$T_j = \max_i(T_j) \tag{2.45}$$

If more than one node is maximal, then one of them is chosen at random. After an input presentation on which node j is chosen, j becomes committed.

The node j initially chosen by Equation (2.45) remains constant if j is uncommitted or if j is committed and where ρ^* is constrained so that

$$T_j \geq \rho^* \tag{2.46}$$

$$0 \leq \rho^* \leq 1 \tag{2.47}$$

If j is committed and the following condition is met then j is reset to the index of an arbitrary uncommitted node. T_j in Equation 2.45 equals the cosine of the angle between I and z_j^*.

$$T_j < \rho^* \tag{2.48}$$

At the end of an input presentation, z_j^* is equal to $z_j^{*(new)}$ defined by the equation below.

$$T_i = \begin{cases} I & if\ j\ is\ an\ uncommited\ node \\ R\left(\beta R\Psi + (1-\beta)\, z_j^{*(old)}\right) & if\ j\ is\ an\ commited\ node \end{cases} \tag{2.49}$$

If j is a committed node, $z_j^{*(old)}$ denotes the value of z_j^* at the start of the input presentation,

$$\Psi_i \equiv \begin{cases} I_i & if\ z_j^{*(old)} > 0 \\ 0 & otherwise \end{cases} \tag{2.50}$$

and

$$0 \leq \beta \leq 1 \tag{2.51}$$

3. ARKAQ-LEARNING

ARKAQ is a novel online reinforcement learning algorithm developed in this study. The goal is to formulate plans for partially observable environments. To achieve this goal, it autonomously segments the belief simplex and emerges complex behaviours that deal with these segments. Since the environment is partially observable, we used POMDP to model the real environment. We have assumed that POMDP parameters are not known, indeed the agent will discover them and that these parameters are continuous. These two assumptions make the problem even harder and such POMDP's cannot be solved exactly. ARKAQ utilizes combination of various techniques, Kalman Filtering, ART2 Networks and Q-learning.

The proposed architecture shown in Figure 3.1 consists of two layers. The first layer is a world model generator that incrementally segments belief simplex for a given task (Tesauro, 2002). The second layer is a policy generator that proposes actions that deal with each belief segment in order to maximize expected discounted reward. Since the internal world representation is non-Markovian we have used a variation of Q-learning which takes belief state as input at this layer.

3.1. World Model Generator

Although the real world has the Markovian property, an agent can observe it partially due to noisy and incomplete perception. In order to achieve its goal, the agent must learn the underlying real world states related to its task(s), i.e. construct an internal belief state representation that interprets continuous and non-Markovian perception into clustered belief points sharing a common action.

The first layer takes the current action a_t, observation o_{t-1} and the belief b_{t-1} as inputs. Then it incrementally segments and distinguishes current internal world representation b_t, which in turn is recursively conditioned current estimate on all of past measurements. As a result, history is implicitly maintained, which is important

for successfully building an internal world model in a POMDP environment. The world model generator is a hybrid architecture composed of a state representation network augmented with state estimation.

It makes use of Kalman filtering and ART2-A networks. Prediction phase of Kalman Filter is modified and calculated by ART2-A network. We have modified the update rule, given in Equation (3.1), as follows, that made the algorithm converge faster and yield better policies:

$$T_i = \begin{cases} I & if\ j\ is\ an\ uncommited\ node \\ \left(z_j^{*(old)} + \beta(I - z_j^{*(old)})\right) & if\ j\ is\ an\ commited\ node \end{cases} \tag{3.1}$$

The concept of our hybrid network can be summarized as follows:

- ART updates the state space and predicts the current state.
- Kalman recursively conditions the current estimate on all of the past measurements and fine-tune the current state estimate. This helps to keep history (which is important for POMDP solutions) in an implicit manner.

3.2. Policy Generator

We used a method of determining the long-term value of acting in each state. Using this information, an agent that knows the state can act optimally by the definition of a Markov process.

Since a Markovian internal world model is constructed, Q-learning is used to generate the optimum policy. The world model generator reduces the complexity of $Q(b, a)$ by aggregating states into cluster centers. The optimal action or actions from any cluster center is the one with the highest Q-value. Closer Q-values may result in more than one optimal action from a state. This layer is activated after the world model layer has converged, or in other words the number of states has converged. The final architecture is given in Figure 3.1.

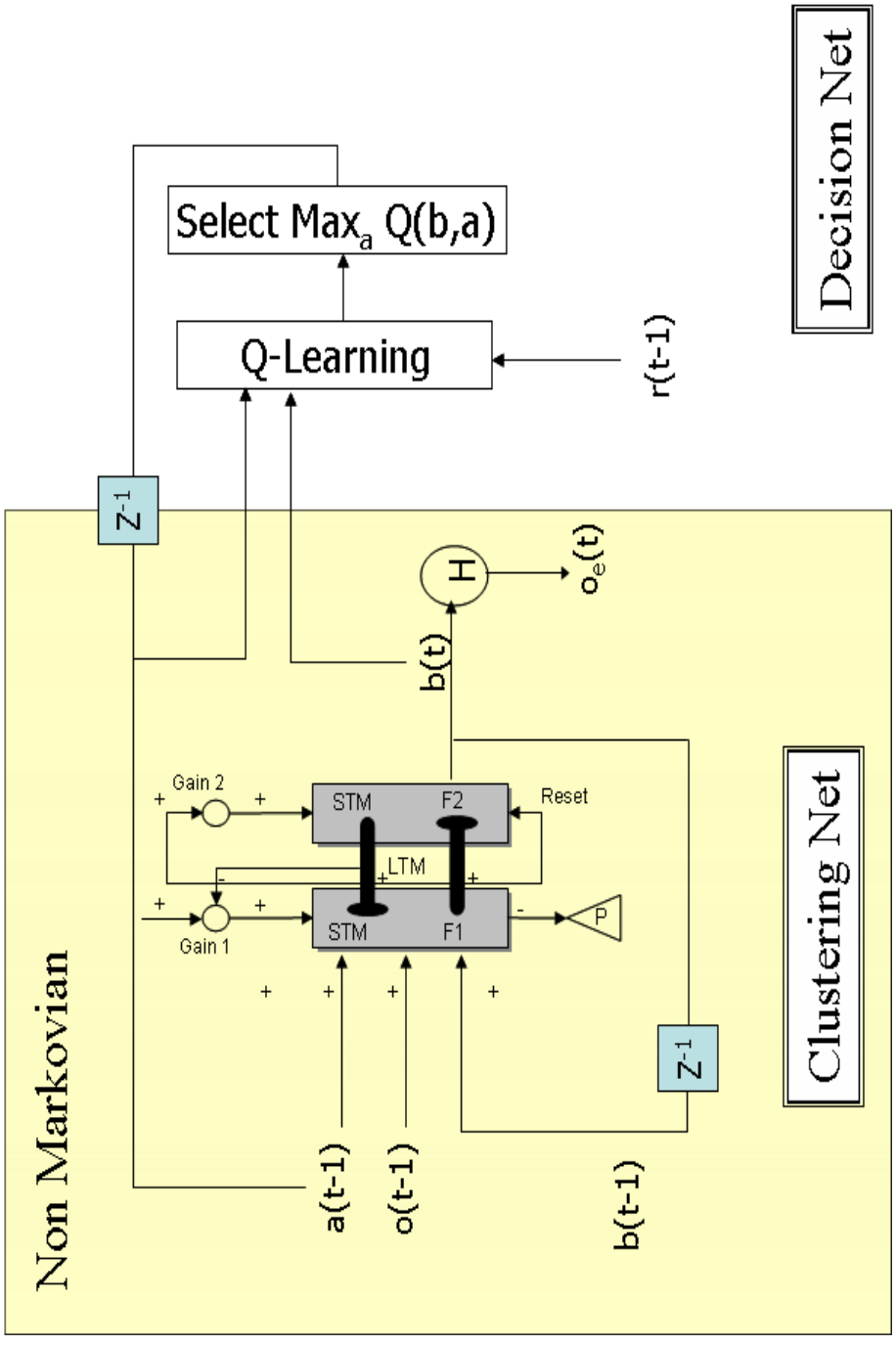

Figure 3.1. The ARKAQ architecture.

3.3. ARKAQ Algorithm

The pseudo code of the ARKAQ algorithm is given in Figure 3.2.

procedure $WorldModelGenerator(a, o, r, s)$

1: $stepCount \leftarrow 0$
2: **repeat**
3: $stepCount \leftarrow stepCount + 1$
4: $a \leftarrow BoltzmanExplorationExploitationCriteria$
5: $B \leftarrow KalmanBeliefCalculation$
6: $b \leftarrow ART2BeliefSimplexUpdate(b)$
7: **until** $simplexBConverged$ or $stepCount \geq maxStepCount$ or $size(B) \geq maxStateCount$

procedure $PolicyGenerator(a, b)$

1: $stepCount \leftarrow 0$
2: **repeat**
3: $stepCount \leftarrow stepCount + 1$
4: $s \leftarrow ARTKalmanFindCurrentMarkovianState(b)$
5: $a \leftarrow BoltzmanExplorationExploitationCriteria$
6: $Q - learningTuneParameters$
7: **until** $explorationExploitationOver$ or $stepCount \geq maxStepCount$

Figure 3.2. ARKAQ-learning Pseudo-code

3.4. Experimental Results

The experiments for ARKAQ learning are done using a customized simulator, developed by Microsoft® Visual C++. We have chosen some well known POMDP that have previously appeared in the literature (Cassandra 1999). Although these problem definitions assume the state and observation space is discrete, we modified their definitions so that the state and observation space is continuous. To convert discrete world parameters to continuous case, we have made a heuristic mapping. For example, in the shuttle docking problem, if the shuttle cannot sense 30 percent of time,

then we likewise disturbed the sensor values by a Gaussian noise $N \sim (0, 0.3)$. The same heuristic is applied to actuator noise definitions.

3.4.1. The Basic POMDP Problem

The basic problem is a one-dimensional hallway where the goal is placed closer to one end. A simulator screenshot is given in Figure 3.3. The agent can move *EAST* and *WEST*. The goal state is the right end ehich is rewarded by +10, a correct action is rewarded with +1 and an invalid action is punished by -1. The agent has two range finders disturbed with a Gaussian noise $R \sim (0, 0.25)$. Actions are disturbed with Gaussian noise $Q \sim (0, 0.45)$. These parameters are arbitrarily chosen. Note that the region is closed so that the agent cannot move outside the region.

Figure 3.3. Basic POMDP Problem with 4 cells one of which is a goal state

ARKAQ identified four states including the goal state, when $\gamma = 0.9$, and two states when $\gamma = 0.1$.

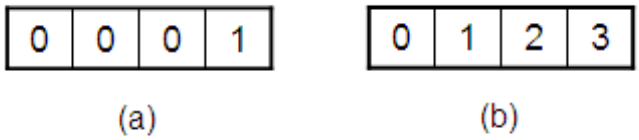

Figure 3.4. (a) ARKAQ identified states $\gamma = 0.1$ (b) ARKAQ identified states $\gamma = 0.9$

Figure 3.5 shows Q-value convergence for two states with $\gamma = 0$. Figure 3.6 shows Q-value convergence for four states with $\gamma = 0.9$. Line above is the goal state. The Q-values have converged for both goal and non goal states. Non oscillating feature of these results show the resulting policies are stable. For this problem low vigilance value performed better in terms of number of discovered states. For all of the non-goal states, the action policies are equal. For both vigilance values, the goal state could be identified separately.

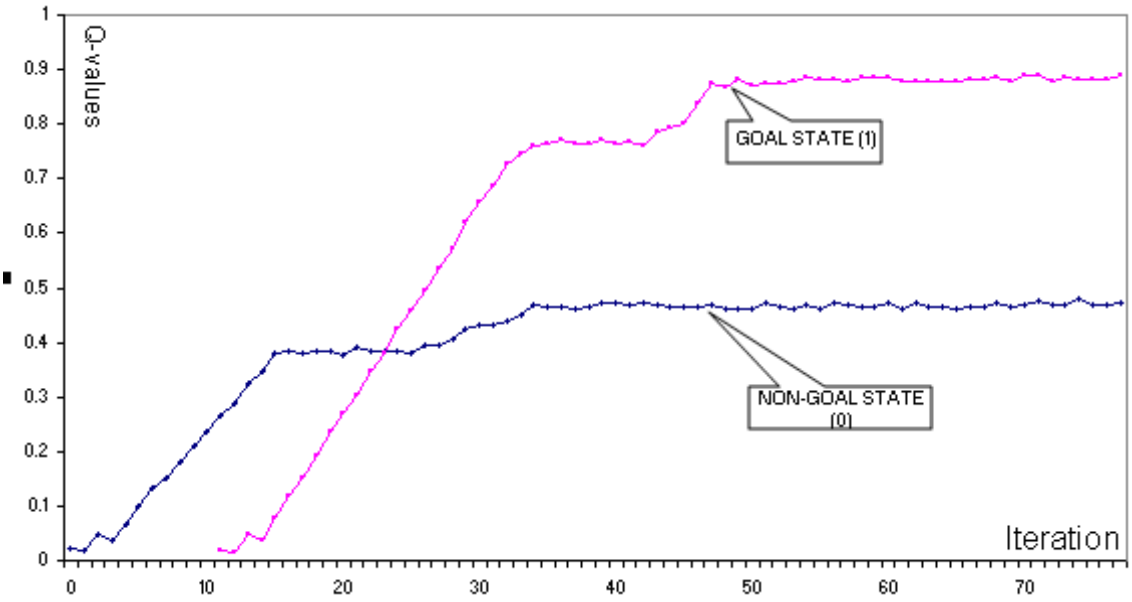

Figure 3.5. Q-Value Convergence for Basic POMDP Problem ($\gamma = 0.1$)

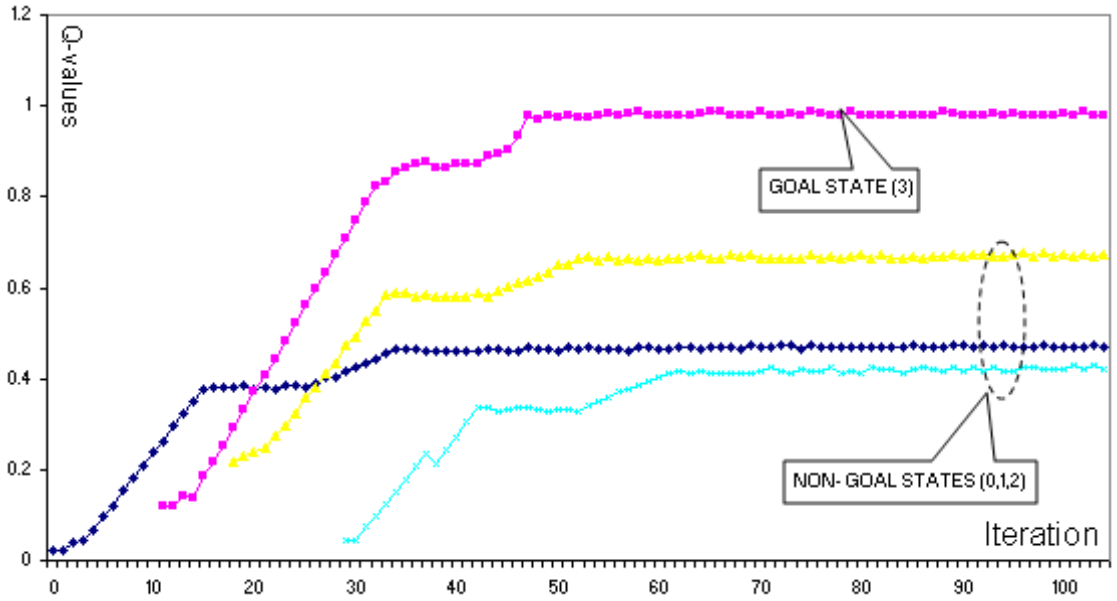

Figure 3.6. Q-Value Convergence for Basic POMDP Problem ($\gamma = 0.9$)

3.4.2. 4x4 POMDP Maze Problem

This problem is a two dimensional maze. A simulator snapshot is given in Figure 3.7. The goal state is the lower right corner. The agent can move *EAST, WEST, LEFT* and *SOUTH*. A square is split into 16 blocks where each edge of the block measures 0.25 units. The goal of the agent is to position itself to the bottom right corner. Four distance sensor gives observation and they are disturbed with Gaussian noise $R \sim (0, 0.25)$. Actions are disturbed with Gaussian noise $Q \sim (0, 0.45)$. Trying to move in an impossible direction leaves the agent where it is.

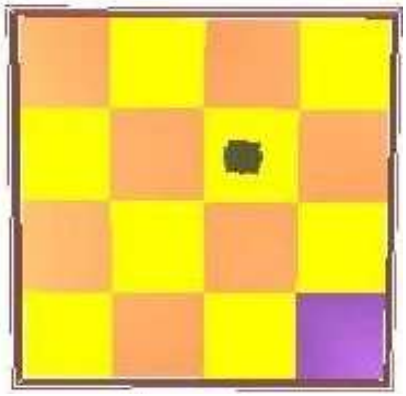

Figure 3.7. 4x4 maze.

Kalman parameters are listed in the equations given below. Note the parameters are given as 2x2 matrices, since the degree of freedom of actions is two.

$$Q = \begin{bmatrix} 0.45 & 0 \\ 0 & 0.45 \end{bmatrix} \tag{3.2}$$

$$R = \begin{bmatrix} 0.25 & 0 \\ 0 & 0.25 \end{bmatrix} \tag{3.3}$$

The agent can only sense the reward and the region is bounded. Since the agent is almost blind and the reward is delayed, this is a challenging problem.

Figure 3.9 shows Q-value convergence for $\gamma = 0.1$. The series with higher coverged

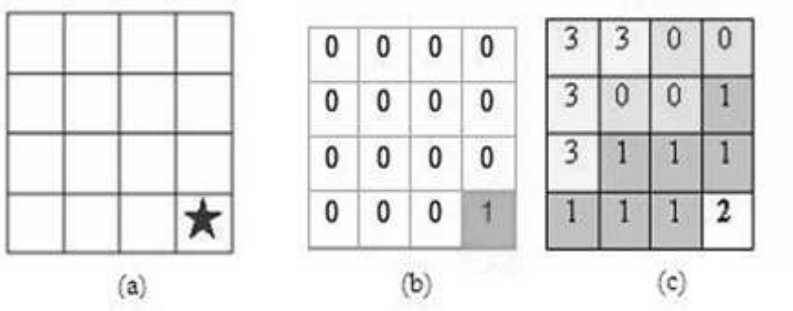

Figure 3.8. (a) 4x4 maze problem (b) ARKAQ identified states ($\gamma = 0.1$) (c) ARKAQ identified states ($\gamma = 0.9$).

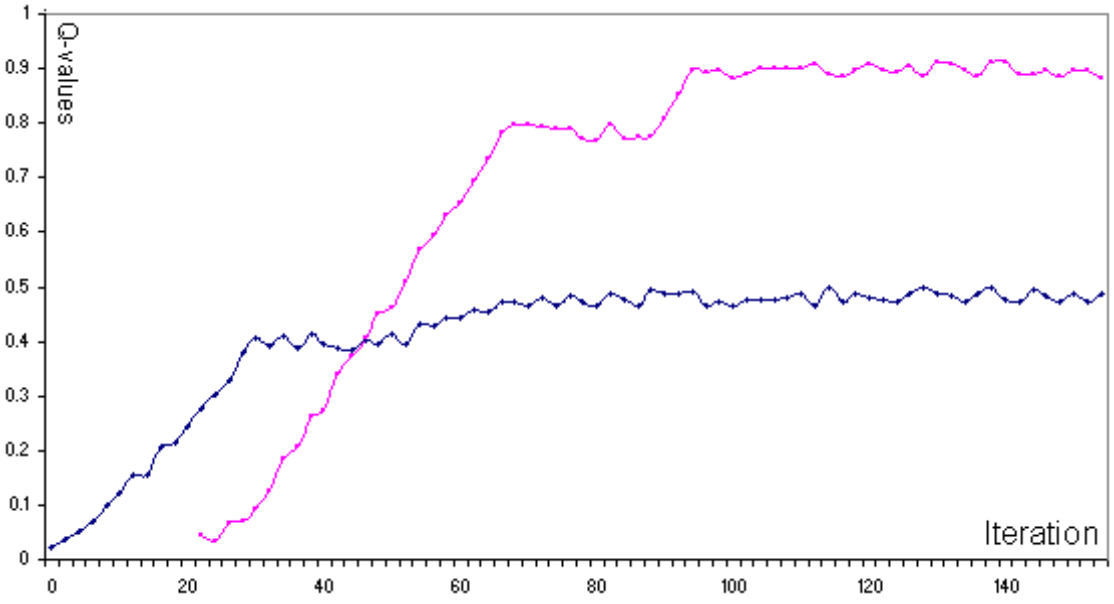

Figure 3.9. Q-Value Convergence for 4x4 maze problem ($\gamma = 0.1$)

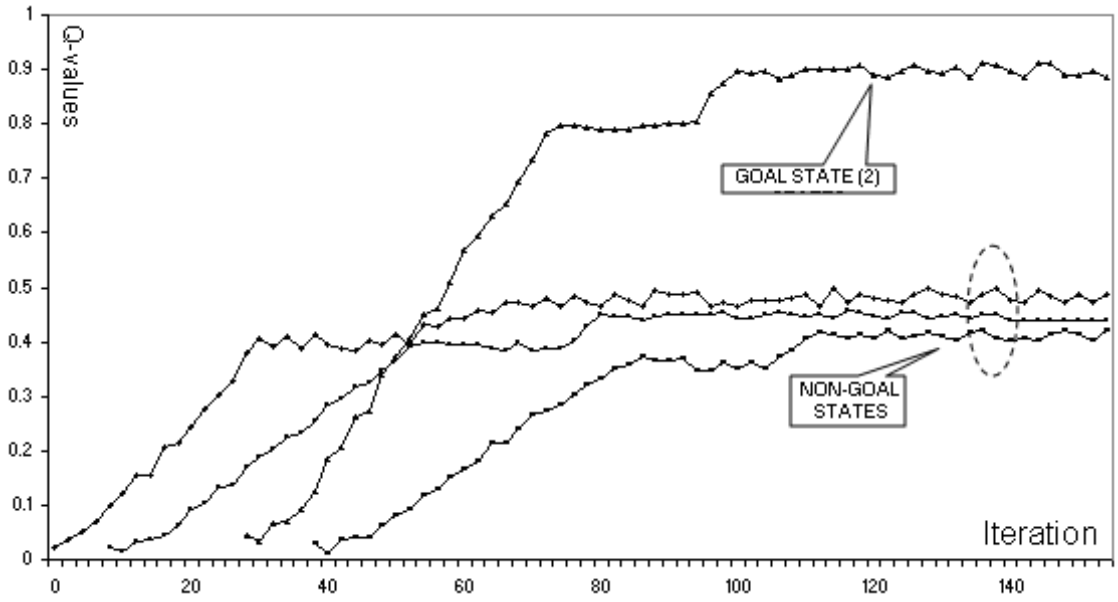

Figure 3.10. Q-Value Convergence for 4x4 maze problem ($\gamma = 0.9$)

Q-value is the goal state. Figure 3.10 shows Q-value convergence for $\gamma = 0.9$. Again the series with higher coverged Q-value is the goal state. Non oscillating feature of these results show that the resulting policies are stable. For this problem also high vigilance value performed better in terms of number of discovered states.

With $\gamma = 0.1$, grouping all non-goal states into one state, the agent could only reveal a stochastic policy. Although the agent favored action *SOUTH* or *EAST*, with similar highest Q-values, it is clear that the cells at the *EAST* boundary, should favor *SOUTH* and the cells at the *SOUTH* boundary should favor *EAST*. For $\gamma = 0.9$, this problem suffers from the same problem for some revealed states. For both vigilance values, the goal state could be identified separately. As in ARKAQ, the non-oscillating Q-values show that the resulting optimal policy is stable.

3.4.3. Load Unload Problem

In this very challenging problem, the agent has a cart that must be driven from an Unload location to a Load location, and then back to Unload. This problem is a simple POMDP with a hidden variable that makes it partially observable (the agent cannot see whether it is loaded or not): If the agent had memory it would remember

its latest Load or Unload action, and it would go left or right correspondingly.

The agent is a cart designed to shuttle loads back and forth between two endpoints on a line. A road is split into 4 segments where each measures 0.25 units. At the left end we pick up an item from an infinite pile. At the right, we drop our item. A reward of +20 is received for picking up or putting down, but only one item can be carried at a time. The cart does not have sensors to indicate whether it is loaded or unloaded, but it can determine its position on the line. Two distance sensor gives observation and they are disturbed with Gaussian noise $R \sim (0, 0.25)$. To act optimally the belief state must encode whether we have an item or not. Actions are simply move *WEST* or move *EAST* which are disturbed with Gaussian noise $Q \sim (0, 0.45)$. Moving in an impossible direction leaves the agent where it is. This is also a nice problem because we can easily increase the mixing time by increasing the length of the road, and the optimal policy is one where the cart moves back and forth between the leftmost and rightmost states moving.

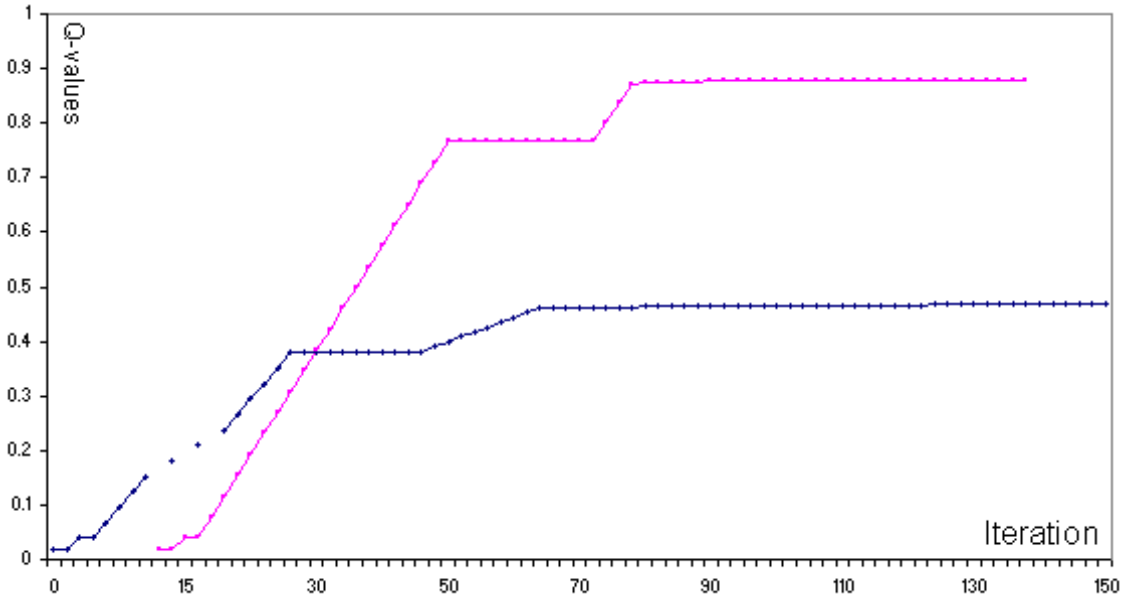

Figure 3.11. Q-Value convergence of two of the states for Load Unload Problem

Figure 3.11 shows the Q-value convergence for a sample non-goal state and a goal state with $\gamma = 0.8$. The goal state converged with a higher Q-value. State segmentation layer classified the same regions as two seperates states revealing the hidden state, loaded or unloaded.

The world model generator has revealed whether the cart is loaded or unloaded. State 1 denotes the cells marked 0 when the cart is unloaded and State 4 denotes the cells marked 0 when the cart is loaded. Also note that the optimal action indicated with arrows differs for these two states. And once again the agent could classify goal states from the non-goal states.

3.4.4. 5x5 Grid Multi Station Load-Unload Problem

In this variation of the standard Load-Unload problem, there are two "load" stations namely L1, L2, and two "unload" stations, U1, U2, placed in a 5x5 grid world. When the agent loads at station L1 it has to unload at U1, or visa versa. The agent can move *EAST, WEST, LEFT* and *SOUTH*. The goal state is rewarded by +10, a correct action toward closest goal is rewarded with +2, a correct action toward closest goal is rewarded with +1 and an invalid action is punished by -1. Note the goal state changes, depending on the loaded station. Figure. 3.12 shows the sketch of the 5x5 world. The agent is to stabilize itself between one of the Load and Unload station pairs.

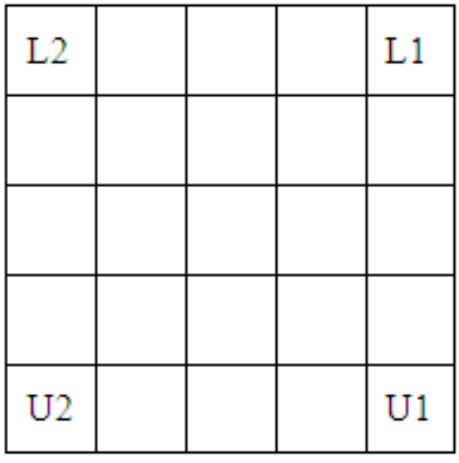

Figure 3.12. Multi station Load Unload Problem

As shown in Figure 3.13, ART successfully preserved the previously learned knowledge despite of the new learned things, as defined by the stability-plasticity dilemma.

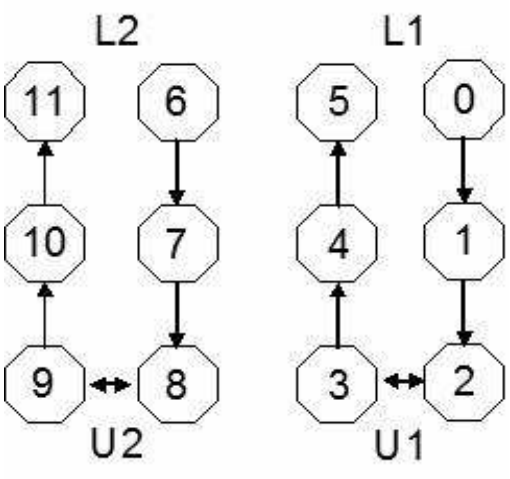

Figure 3.13. ARKAQ identified state space model for Multistation Load Unload Problem

Figure 3.14 shows the Q-value convergence for a sample non-goal state and a sample goal state. The goal state converged with a higher Q-Value. As in Load Unload problem, the agent could reveal the hidden state. However, the agent stabilizes itself among the first identified Load/Unload stations.

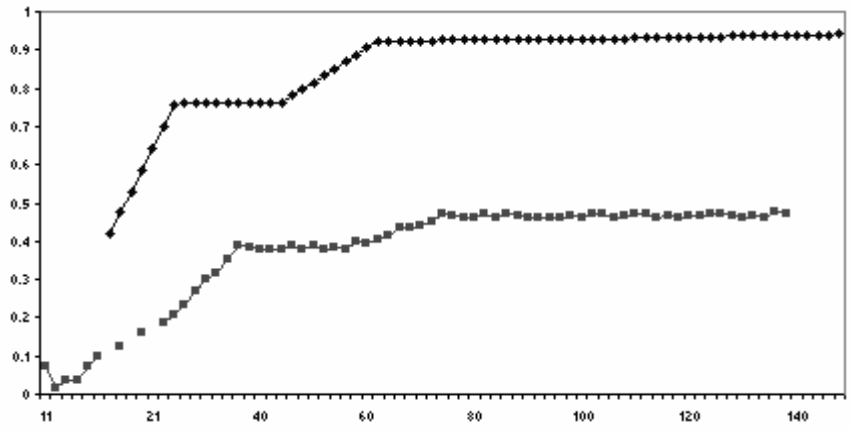

Figure 3.14. Q-Value convergence for the two of the states 5x5 Grid Multi Station Load Unload Problem

3.4.5. Shuttle Docking Problem

The agent in this scenario also has incomplete perception, non-deterministic actions and noisy sensors. There are two space stations separated by free space with locking docks on each station. The task is to transport supplies between two docks. Each time the agent successfully attaches the least recently visited station; it receives

a reward of +10. In order to dock, the agent must position itself in front of the station with its back to the dock. Whenever the agent collides with the station propelling forward into the dock, it receives a penalty of -10. At all other times it receives zero reinforcement. The actions available to the agent are: *MOVE* and *TURN*, which are disturbed with Gaussian noise $Q \sim (0, 0.30)$. The agent always faces one of the stations, and *TURN* causes it to face the other. Depending on the state, *MOVE* action detaches from the loading dock, launches into free space, approaches next station from free space or collides into the space station directly ahead. The agent is equipped with the two-distance sensor, disturbed with a Gaussian noise $R \sim (0, 0.25)$.

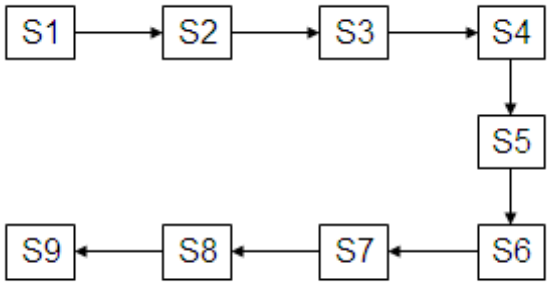

Figure 3.15. ARKAQ Shuttle Docking Problem

As shown in Figure 3.15, ART successfully preserved the previously learned knowledge despite of the new learned things, as defined by the stability-plasticity dilemma.

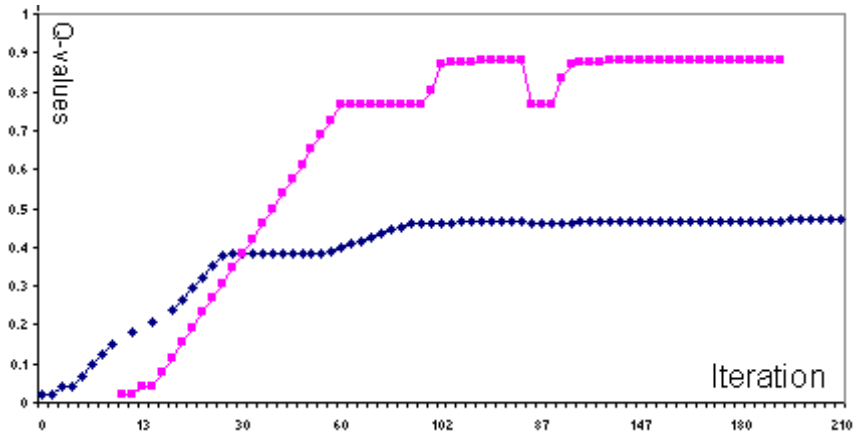

Figure 3.16. Q-Value convergence for two stace for the Shuttle Docking Problem

Figure 3.16 shows the Q-value convergence for a sample non-goal state and a sample goal state. The goal state converged with a higher Q-Value. As in Load

Unload problem, the agent could reveal the hidden state.

3.4.6. Cart and Inverted Pendulum Problem

We have also modified the definition of a well known continuous state space MDP problem, namely *Cart Pole Problem* or *Inverted Pendulum Problem*. The inverted pendulum is a widely used benchmark for testing control algorithms. In the literature, inverted pendulum problem is modelled as a fully observable markov decision process. To convert this problem to a continuous POMDP problem, we have added Gaussian noise to the observable system variables, as well as to the applied force.

Here, the task is to balance an inverted pendulum hinged to a cart by accelerating and slowing down the cart. If the pole falls past a given angle the episode ends with a failure and the cart is reset. If the cart moves beyond given limits, the cart is reset with a failure too. If a failure occurs the agent gets a negative reward of -1.

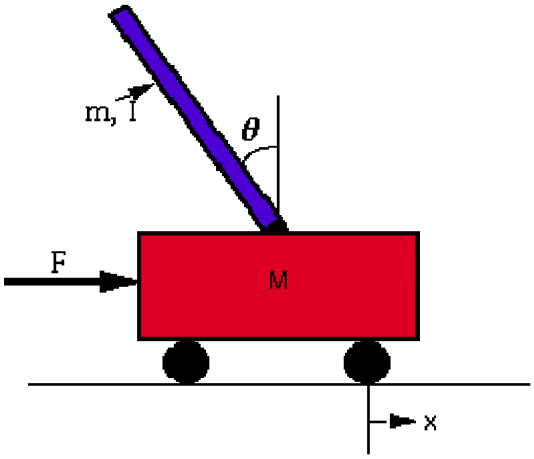

Figure 3.17. Sketch of the cart pole problem.

The cart with an inverted pendulum, shown below, is bumped with an impulse force, F. We determine the dynamic equations of motion for the system, and linearize about the pendulum's angle, $\theta = \pi$ in order to use Kalman filter in the position estimate. We assume that the pendulum does not move more than a few degrees away from the vertical, chosen to be at an angle of $\pi/2$.

For this derivation we will assume the parameters given in Table 3.1.

Table 3.1. Cart and Inverted Pendulum Problem

Symbol	**Expression**	**Value**
M	mass of the cart	$0.5kg$
m	mass of the pendulum	$0.2kg$
b	friction of the cart	$0.1N/m/sec$
l	length to pendulum center of mass	$0.3m$
I	inertia of the pendulum	$0.006kg*m^2$
F	force applied to the cart	
x	cart position coordinate	
θ	pendulum angle from vertical	

We will further assume that the system starts at equilibrium, and experiences an impulse force of 1N. The pendulum should return to its upright position within 5 seconds, and never move more than 0.05 radians away from the vertical.

3.4.7. Force analysis and system equations

Below are the two Free Body Diagrams (FBD) of the system.

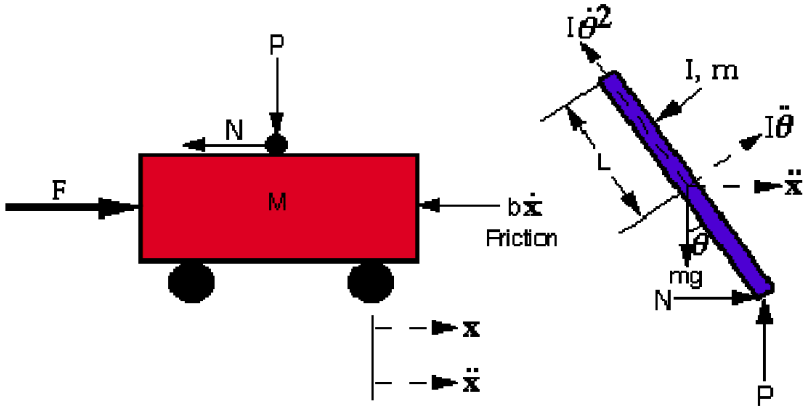

Figure 3.18. Free Body Diagram for Cart and Inverted Pole Problem

Summing the forces in the FBD of the cart in the horizontal direction, we get the

following equation of motion:

$$M\ddot{x} + b\dot{x} + N = F \tag{3.4}$$

Summing the forces in the FBD of the pendulum in the horizontal direction, we can get an equation for N:

$$N = m\ddot{x} + ml\ddot{\theta}\cos\theta - ml\dot{\theta}^2\sin\theta \tag{3.5}$$

If we substitute this equation into Equation (3.4), we get the first equation of motion for this system:

$$(M+m)\ddot{x} + b\dot{x} + ml\ddot{\theta}\cos\theta - ml\dot{\theta}^2\sin\theta = F \tag{3.6}$$

To get the second equation of motion, we summed the forces perpendicular to the pendulum. Solving the system along this axis we get the following equation:

$$P\sin\theta + N\cos\theta - mg\sin\theta = ml\ddot{\theta} + m\ddot{x}\cos\theta \tag{3.7}$$

To get rid of the P and N terms in the equation above, we sum the moments around the centroid of the pendulum to get the following equation:

$$-Pl\sin\theta - Nl\cos\theta = I\ddot{\theta} \tag{3.8}$$

Combining these last two equations, we get the second dynamic equation:

$$(I + ml^2)\ddot{\theta} + mgl\sin\theta = -ml\ddot{x}\cos\theta \tag{3.9}$$

This set of equations should be linearized about $\theta = \pi$. Assume that $\theta = \pi + \phi$ represents a small angle from the vertical upward direction). Therefore, *cos(θ) = -1, sin(θ) = -θ,* and $(d\theta/dt)^2 = 0$. After linearization the two equations of motion become

(where u represents the input):

$$\begin{aligned} (I + ml^2)\,\ddot{\phi} - mgl\phi &= ml\ddot{x} \\ (M + m)\,\ddot{x} + b\dot{x} - ml\ddot{\phi} &= u \end{aligned} \tag{3.10}$$

3.4.8. Kalman Filter State-Space Equation

The linearized system equations can also be represented in state-space form:

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\phi} \\ \ddot{\phi} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{-(I+ml^2)b}{I(M+m)+Ml^2} & \frac{m^2gl^2}{I(M+m)+Ml^2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{-mlb}{I(M+m)+Ml^2} & \frac{mgl(M+m)}{I(M+m)+Ml^2} & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \phi \\ \dot{\phi} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{I+ml^2}{I(M+m)+Ml^2} \\ 0 \\ \frac{ml}{I(M+m)+Ml^2} \end{bmatrix} u \tag{3.11}$$

$$\begin{bmatrix} x_t \\ \dot{x_t} \\ \theta_t \\ \dot{\theta_t} \end{bmatrix} = \tau. \begin{bmatrix} \dot{x_{t-1}} \\ \ddot{x_{t-1}} \\ \dot{\theta_{t-1}} \\ \ddot{\theta_{t-1}} \end{bmatrix} + \begin{bmatrix} x_{t-1} \\ \dot{x_{t-1}} \\ \theta_{t-1} \\ \dot{\theta_{t-1}} \end{bmatrix} \tag{3.12}$$

We can rewrite Equation (3.11) and Equation (3.12) as

$$\dot{s}(t) = Fs(t-1) + Gu(t-1) \tag{3.13}$$

$$s(t) = \tau\,\dot{s}(t) + s(t-1) \tag{3.14}$$

then substituting Equation (3.13) in Equation (3.14) we get Equation (3.15).

$$s(t) = \tau\,[F + I]\,s(t-1) + Gu(t-1) \tag{3.15}$$

Finally we get Kalman filter motion model parameters A and B as given in Equation (3.16) and Equation (3.17) respectively.

$$A = \tau \left[F + I\right] \tag{3.16}$$

$$B = G \tag{3.17}$$

The agent's state estimation and observation are both composed of $\left[x, \dot{x}, \Phi, \dot{\Phi}\right]$, and each element is disturbed with a Gaussian noise $R \sim (0, 0.15)$ and the action is disturbed with a Gaussian noise $Q \sim (0, 0.45)$. Kalman Filter parameters are calculated according to the above equation (MathWorks, 1994).

ARKAQ could manage to achieve a stable set of states when continuous inputs were discrete. In the continuous case we could not come up with a stable solution.

4. KAFAQ – LEARNING

Throughout our work, we suggested a variation of ARKAQ-learning. This chapter will present this suggested approach.

4.1. Why KAFAQ-learning

The need for a new approach has arisen from the reasons listed below.

4.1.1. Curse of Noisy Data

As we already know, the ART framework introduces the concept of "resonance" so that each state is processed by:

- Finding the "nearest" cluster center that "resonates" with the state;
- Updating that cluster center to be "closer" to the state.

"Resonance" is just a matter of being within a certain threshold of a second similarity measure. A crucial feature of ART is that if no cluster center resonates with the current state, a new cluster is created as in Hartigan's leader algorithm (Hartigan, 1975). This feature is said to solve the "stability-plasticity dilemma".

But the main limitation of ART in case of POMDP problems where large amount of noisy data is inherently involved, is that it may create irrelevant clusters continuously depending on the vigilance parameter choice.

4.1.2. Curse of static vigilance parameter

With a continuous state space, the belief space is also continuous, as in the discrete case, but now with an infinite number of dimensions. In ART networks the vigilance parameter, which defines the state sizes, is decided before running the exper-

iment and is static during the experiment. As one may guess it generates states with the same range, which is very rare in real world problems. Moreover, the number of clusters may explode depending on the vigilance parameter.

ART algorithm keeps the cluster center called the "long term memory" and, the input data are categorized, depending on its resonance with the cluster center according to the vigilance parameter. To deal with the real world problems with varying range of belief state segments sharing a common best action, we need to move from cluster center representation to cluster range representation.

4.1.3. Overfitting depending on the vigilance parameter

When the vigilance is small the ART network produces large states. As it gets larger, the vigilance of the network increases, and finer states result. When equal to one, the prototype vectors have to match the input vectors perfectly. In this situation every input produces a new state equal to itself. The result is overfitting the input data.

In our experiments we noticed that no single rule of thumb exists for the choice of the vigilance parameter. The vigilance value, which is valid for maze-kind exploration problems, causes really unacceptable results for control problems like cart-pole. Moreover the vigilance value for the maze-kind problems is very dependent on the dynamics of the agent and the characteristic of the agent's world.

The only choice for the experimenter to find the optimal vigilance parameter is to repeat the experiment each time with different vigilance value until the optimal solution is obtained as the result of the experiment.

4.1.4. Effect of Exploration Phase on results

One of the challenges that arise in reinforcement learning and not in other kinds of learning is the trade-off between exploration and exploitation. To obtain a lot of

reward, a reinforcement-learning agent must prefer actions that it has tried in the past and found to be effective in producing reward. But to discover such actions, it has to try actions that it has not selected before. The agent has to exploit what it already knows in order to obtain reward, but it also has to explore in order to make better action selections in the future. As a result, in the exploration phase, state and action pairs, which are irrelevant to the solution of the problem, may have been visited and categorized as a new state by the ART network. In continuous environments, parameters which causes a long exploration phase combined with the unnecessary categorization caused overflow in the number of discovered states. Therefore the choice of vigilance parameter now involving the dependency on exploration parameters becomes even more difficult to decide.

4.1.5. Need for Merging states

One way to get rid of spurious states mentioned in the previous section is pruning states by merging. In ART there is no pruning phase. If a pruning phase existed an open question is how the vigilance value would have been affected; would it remain the same or change, because if not changed one will risk with the re-creation of merged states. But changing the vigilance parameter, while running the experiment, may cause the previously discovered states become meaningless.

Moreover, it is very difficult to define a merging or pruning criteria for an algorithm that keeps the cluster centers but not their range. It seems that we are again faced with the problem of state representation.

4.1.6. Representation of past states

In the ARKAQ-learning algorithm the single previous state is fed as one component of the input vector. As a result the ART categorizes separately a single state, which can be reached from many previous states, which causes an overflow in the number of states in continuous domain problems. How to change the ART algorithm to accept variable number of previous states or input data and how the organization

of the network (e.g. the long term memory representation) will be affected are very difficult questions. Instead of taking the risk of changing the functioning of the ART algorithm and getting probably a very unstable system, we preferred to keep the necessary concepts which help to solve the POMDP problems and combine them with the representation and functioning of our new algorithm.

4.1.7. Vulnerable to input sequence representation

The often-repeated claim that ART algorithms are "capable of rapid stable learning of recognition codes in response to arbitrary sequences of input patterns" merely means that ART algorithms are clustering algorithms that converge; it does not mean, as one might naively assume, that the clusters are insensitive to the sequence in which the training patterns are presented, quite the opposite is true.

In the exploration phase, state visitation pattern has greatly changed depending on the choice for action in each experiment run. Those changes at the exploration phase corresponding to changes in the input sequence representation, gave different representations of the same world. We may say a different perspective of the world, but in each case a solution to the optimal policy is generated. Although the internal representation of the same world is changing, actions to cope with this new world are successfully generated by the agent.

4.1.8. Concepts borrowed from ART

Although ART network has previously stated disadvantages, many concepts used in the development of the KAFAQ-learning are borrowed form ART networks:

- The concept of committed and uncommitted states.
- The parallel search of currently discovered states.
- Creation of a new state if no previously discovered state resonates with the current state.

To overcome the difficulties we encountered during the solution of POMDP problems we have changed:

- The cluster center representation
- The static value of vigilance parameter
- The static representation of "long term memory"

And we added new phases such as splitting and merging of belief states, which will efficiently group and ungroup belief states that a single best action rule until the system converges to an optimal policy in finitely-transient problems.

Keeping the belief states grouped according to their best action will avoid the curse of dividing the same state with the sub-optimal policies and cause an overflow in the number of states. Keeping the number of states in the acceptable ranges for online POMDP algorithms, which runs on continuous world without the world model, is one of the most difficult problems to deal with.

4.2. KAFAQ Algorithm

We aim to use POMDP as a model of the agent's environment and assume the state and observation spaces are continuous. We have further assumed that POMDP parameters are not known. These two assumptions make the problem even harder and such POMDP's cannot be solved exactly.

Our Gaussian based POMDP belief states are calculated by a Kalman filter. This filter returns the best state estimate and its covariance, a measure of uncertainty around the best estimate. The uncertainty level gradually converges to a value depending on the measurement and process noise. Our algorithm will first discretize and group the uncertainty levels into segments. For each uncertainty segment, our algorithm will group the continuous state space into closed connected regions, where one best action dominates. At the end, we will able to get the approximate optimal policy for all belief state space.

As the agent explores, these regions may be extended, divided or merged according to their dominant actions. If it can find a belief point where a different action would be better, then this serves as a fact that our current state is composed of more than one region. If the dominating action stays the same although the agent travels outside the region, the state boundaries are extended. This procedure continues until the policy converges, which means that either a finite transient policy is obtained or the simulation time is over. The internal state and policy representation is a simple finite state automaton.

4.2.1. Belief State Representation

As defined in section 2.2.2, in the continuous state case it is impossible to represent each point probability in tabular form, since there are infinetely many such points. So we have assumed the belief to be a Gaussian probability distribution where the mean is the most probable current state and variance is the uncertainty level. Belief formulation at step n is given in Equation (2.9), where s_n is the most probable current state and P_n is the uncertainty level. We make use of Kalman filters to estimate current belief state.

$$b_n = N(s_n, P_n) \tag{4.1}$$

4.2.2. Control Under Uncertainty

In a POMDP problem, the agent may exhibit different behaviours depending on the degree of uncertainty. For the diagnostic example given in section 2.2, the doctor instead of prescribing, he repeats tests until his certainity is above some threshold value. If he had prescribed right from the start, where too much uncertainty exists, he would most likely apply the wrong medication. This type of sequential decision making problems are so general that if a good way of solving is found, it it will be applicable to a wide range of problems in stochastic control.

We suggest a method to autonomously segment levels of uncertainty for each system variable. Once these levels are segmented, then we search for subsegments sharing a common best action among these levels. In other words each uncertainty level has its own optimal policy. We will demonstrate our results with a classical POMDP problems and the inverted pendulum problem.

4.2.3. Uncertainty Level Segmentation

In our experiments we will linearize the systems and use the time-invariant parameters for A, H, Q and R. Maybeck has also shown that the estimation error covariance P of the Kalman filter converges to a unique value from any initial condition where parameters A, H, Q and R are time-invariant.

Although steady state error can be computed by solving Ricatti equation given in Equation (4.2), our algorithm decides on convergence if the current value is the same for last k steps.

$$P = A(P - PH^T(HPH^T + R)^{-1}HP)A^T + Q \tag{4.2}$$

Take a simple example where only one input parameter and its variance exist where R and Q are both equal to 0.5, and A and H are equalt to identity. If $N_{converge}$ is the number of steps that the state estimation error or variance converges, and P_0 is the initial variance, Table 4.1 shows some calculated convergence steps for various initial variance values. For example, if the initial state estimation error or uncertainty is 1000, the system converges to the minimum uncertainty level after 9 steps. All uncertainities with 9 steps to converge are grouped and labeled as uncertainty level 9, all uncertainities with 8 steps to converge are grouped and labeled as uncertainty level 8, and so on.

By the POMDP problem definition, the agent always knows the initial estimation error, which is the highest in our case, in other words the agent knows the boundaries of the estimation error, or the uncertainty value. Using this information, the algo-

Table 4.1. Uncertainty Levels

P_0	$N_{converge}$	Uncertainty Group Label
1000	9	9
5	9	9
1	9	9
0.8	8	8
0.3	7	7

rithm begins with calculating the maximum $N_{converge}$, $N_{maxConverge}$ and the steady state estimation error, $P_{steadyState}$, using Kalman state error estimation methods, given in Equation (2.30), (2.32) and (2.34). Pseudocode for this initial run is given in Figure 4.1.

procedure $FindMaxLevelCount$

1: $p \leftarrow initialStateEstimationError$
2: $convergeCount \leftarrow 0$
3: **repeat**
4: $\quad P_{previous} \leftarrow P$
5: $\quad P \leftarrow KalmanPComputations(P, A, Q, R)$
6: $\quad convergeCount \leftarrow convergeCount + 1$
7: $\quad$ **if** $P_{previous} = P$ **then**
8: $\quad\quad k \leftarrow k + 1$
9: $\quad$ **end if**
10: **until** $k \geq convergeThreshold$ or $convergeCount \geq maxConvergeCount$
11: $N_{maxConverge} \leftarrow convergeCount - k$
12: $P_{steadyState} \leftarrow P$

Figure 4.1. Pseudocode of finding $N_{maxConverge}$ and $P_{steadyState}$

Once $N_{maxConverge}$ and $P_{steadyState}$ are known, for any given state estimation error, P_l in any time step the uncertainty level, l, can be computed. Pseudocode for level computation is given in Figure 4.2. Note that for any P_l, Kalman decreases the uncertainty level by one at each iteration until steady state uncertainty level which is

one is reached. In other words, if the estimation error level is l at time step t, It will be *l-1* at time step *t+1* according to Kalman state estimation error calculation given in Equation (2.30), (2.32) and (2.34). In our experiments, we always initialized the state estimation error P_0 at a level greater than $P_{steadyState}$.

procedure $FindCurrentLevel$

1: $level \leftarrow 1$
2: $P \leftarrow currentEstimationError$
3: **while** $P \neq P_{convergence}$ **do**
4: $\quad level \leftarrow level + 1$
5: **end while**
6: $currentLevel \leftarrow level$

Figure 4.2. Pseudocode for finding an uncertainty level

4.2.4. Connected Belief Regions for an Uncertainty Level

Once uncertainty levels are known, our agent can compute the current uncertainty level, using the algorithm given in Figure 4.2. Now, our algorithm continues with grouping the continuous belief simplex into closed connected regions, where one best action dominates and the connected belief points lie in the same uncertainty level.

For each belief point at uncertainty level $l + 1$, and at step t, the value function is computed using Equation (2.18) as sketched in Figure 4.3.

Since we have infinitely many belief points, the exact calculation of the value function will take infinite time. Instead we suggest an approximation method using TD learning for value calculation. In our algorithm, spatially connected belief points lying at the same uncertainty level are grouped according to their best actions as the agent explores the belief space. This procedure is called *belief region segmentation*. A sketch is given in Figure 4.4, where value estimation for a belief point at an uncertainty level is done by TD(0).

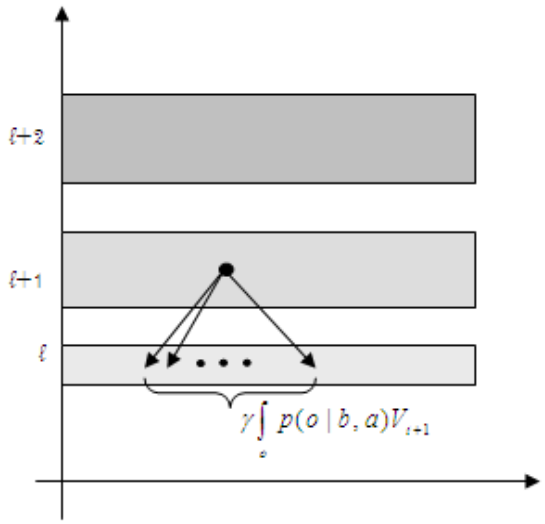

Figure 4.3. Value Iteration at step t and level $l + 1$

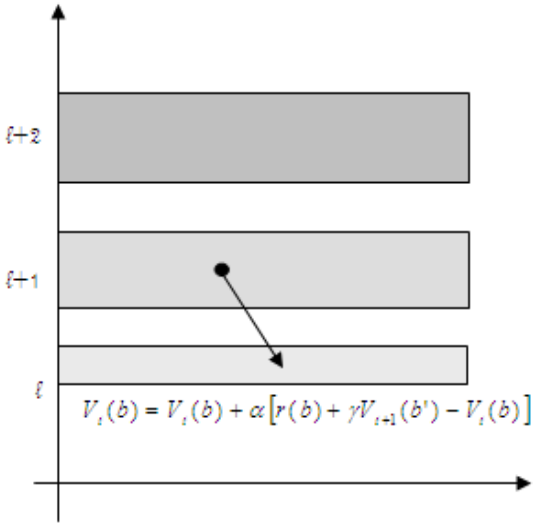

Figure 4.4. TD(0) update for value function

The grouping of belief points begins with points labeled at uncertainty level of $P_{steadyState}$ resembling a bottom up approach, and higher uncertainty levels are grouped once the lower level segmentation has finished. For each level, when the number of belief regions converge that is the number of belief regions do not change further we conclude that the level segmentation is finished. Once a layer is segmented, the belief regions lying at that uncertainty level are marked as commited, and are not updated anymore. The reason for selecting the lowest uncertainty level to start segmentation of belief space is to build up the next upper uncertainty level belief segmentation upon a steady policy. It is normally expected that the policy will be more stable for lower state estimation errors. The pseudocode for the segmentation of the steady state uncertainty level is given in Figure 4.5.

The algortihm constructs a different policy for each uncertainty level, where higher uncertainty level policies are built upon the lower and steadier policies. The policy at the uncertainty level $P_{steadyState}$ may be thought as horizon one policy where the agent is always driven to that level by Kalman filter and explore the level until convergence.

4.2.4.1. Policy Representation and Belief Regions. Our algorithm constructs belief regions, B, for each uncertainty level from spatially connected belief states that share a common best action. The policy is represented as a finite state automaton where each node is a belief region and transitions representing the best action of the source node. The internal representation of a belief region is as follows:

- Uncertainty level label, l,
- Set of estimated state boundary vectors, X_b, where the dimension of an estimated state vector is equal to the number of observable system variables $\{x_1, x_2, \cdots, x_n\}$. This set of vectors define a closed subspace of the observation space,
- A matrix $Q(a)$ that keeps utility, of choosing an action, a, from B during exploration;
- A set of previous belief regions B, from which this belief region is reached.

procedure *SegmentSteadyLevel*

1: B(0).l $\leftarrow N_{converge}$
2: B(0).X $\leftarrow$ initial_belief_range
3: B(0).Q[] $\leftarrow$ zeroes(ActionCount)
4: B(0).UB[] $\leftarrow$ NULL
5: stepCount $\leftarrow 0$
6:
7: **repeat**
8: $s_{current} \leftarrow$ Kalman-State-Estimation
9: $B_{current}$, currentSegmentNo $\leftarrow$ FindCurrentSegment($s_{current}$)
10: TD-Update($B_{previous}$, reward)
11: $a_{current} \leftarrow$ Boltzman-Action-Selection($B_{current}$)
12: $a_{best} \leftarrow$ Best-Action($B_{current}$)
13: **if** $a_{best} = a_{current}$ **then**
14: ExtendRegion ($s_{current}, B_{current}$)
15: **else**
16: **if** $currentSegmentNo > totalSegmentCount$ **then**
17: $s_{current} \leftarrow$ GenerateNewRegion($B_{current}, s_{current}$)
18: **else**
19: $s_{current} \leftarrow$ SplitRegion($B_{current}, s_{current}$)
20: totalSegmentCount $\leftarrow$ totalSegmentCount + 1
21: **end if**
22: **end if**
23: $stepCount \leftarrow stepCount + 1$
24: **until** $simplexBConverged$ or $stepCount >= maxStepCount$

Figure 4.5. Pseudocode for recursive segmentation

4.2.4.2. Generation, Merging and Splitting of Belief Regions. To explain the generation and merging of belief states let us begin with a very simple example where the agent can either move right or left horizontally in a corridor. The goal state is at the left end. Initially the agent is in a belief region bounded by the initial location estimate, s_0 and co-variance $P_{converge}$. Since our algorithm begins with the steady state covariance the state estimation error will be constant until the policy converges. As the robot takes a step Kalman filter estimates the current state, s_1, by recursively conditioning the current estimate on all of the past measurements. If the best action for the current estimate is equal to the previous state estimate then the current belief region boundaries are extended with s_1. Figure 4.6 shows a sketch of the extension of belief region. Finally a belief region is created for the uncertainty level one.

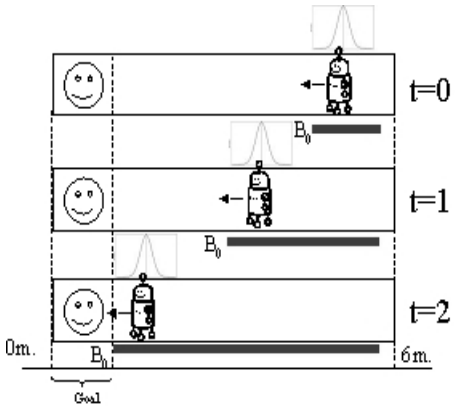

Figure 4.6. Extending a belief region for the uncertainty level one.

While extending belief regions, unvisited belief states with different best actions may have also been included in the region. As the agent navigates, when a belief state already bounded by a belief region with a different best action is reached, the bounding region is split into sub-belief regions. Again assume an agent acting at uncertainty level one, a sketch is given in Figure 4.7, where belief region B_1 is split into two. The newly constructed belief region consists of the belief states that lie within the $\sqrt{P}/16$ neighbours of the current state estimate.

Belief region segmentation of higher uncertainty layers are the same for the steady state belief segmentation except that the value updates are built upon the lower uncertainty level.

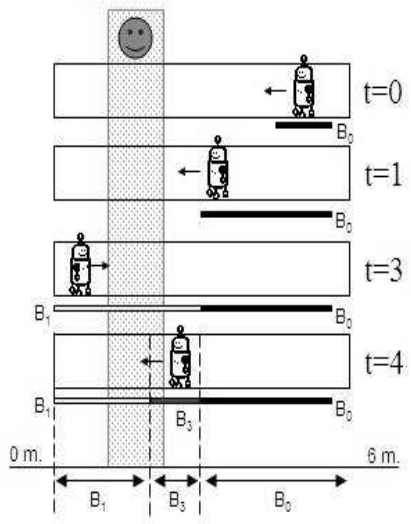

Figure 4.7. Revealed belief regions may be splitted if a different best action dominates for some subregion.

4.3. Experimental Results

The experiments for KAFAQ learning are done using the Webots Simulator (Michel, 2004). The problem definitions were already given in the previous chapter. The robot used in the experiments is a two wheeled, cubic, differential drive robot, with four infrared sensors positioned at the middle of each side.

4.3.1. Basic POMDP Problem

The resulting automaton for the basic problem is sketched in Figure 4.8. The continuous ranges are divided into three belief regions: s_1 to the west of the goal region with dominating action *EAST*, s_2, one to the east of the goal region with dominating action *WEST*. Although the goal area is placed close to one end, the robot could pass through that region and explored till the end of the maze. The very tiny region between the goal state and the end of the maze is also classified as a region, s_3.

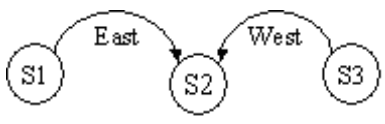

Figure 4.8. Resulting Steady State FSA with KAFAQ.

Wherever the agent starts it will tend to locate itself in state $S2$. KAFAQ could describe the system with two states.

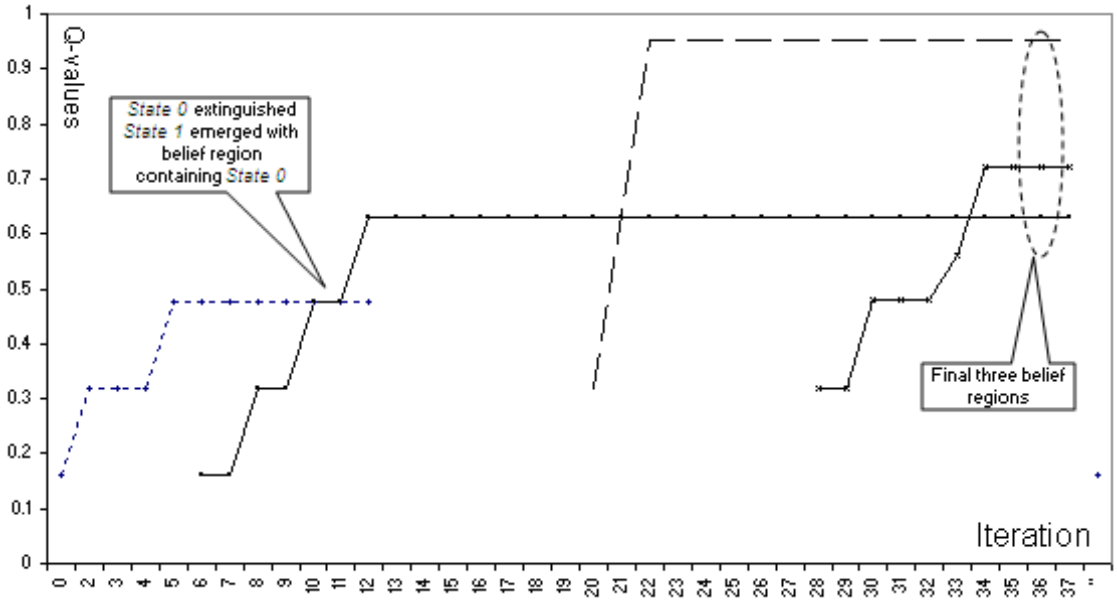

Figure 4.9. Q-Value convergence for one dimensional maze.

The Q-values for each state converged in about 7 steps. Q-value for each state is given in Figure 4.9. This is the approximate optimal solution.

4.3.2. 4x4 POMDP Maze Problem

The resulting automaton found by KAFAQ for the 4x4 maze problem is sketched in Figure 4.10. The continuous range is divided into three belief regions: one that covers all points to the north of the goal region with dominating action $SOUTH$, S_1, one to the west of the goal region with dominating action $EAST$, S_3 and the goal region S_2. Note that both S_2 and S_3 can be reached by taking action $SOUTH$ from state S_1.

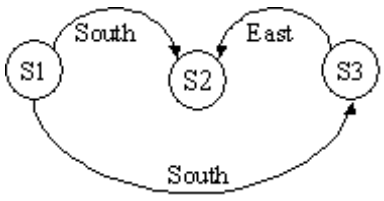

Figure 4.10. Steady State FSA for 4x4 maze with KAFAQ

KAFAQ could identify the problem with three states, whereas ARKAQ could identify with four states. Remember the problem of ambiguous action selection in

ARKAQ.

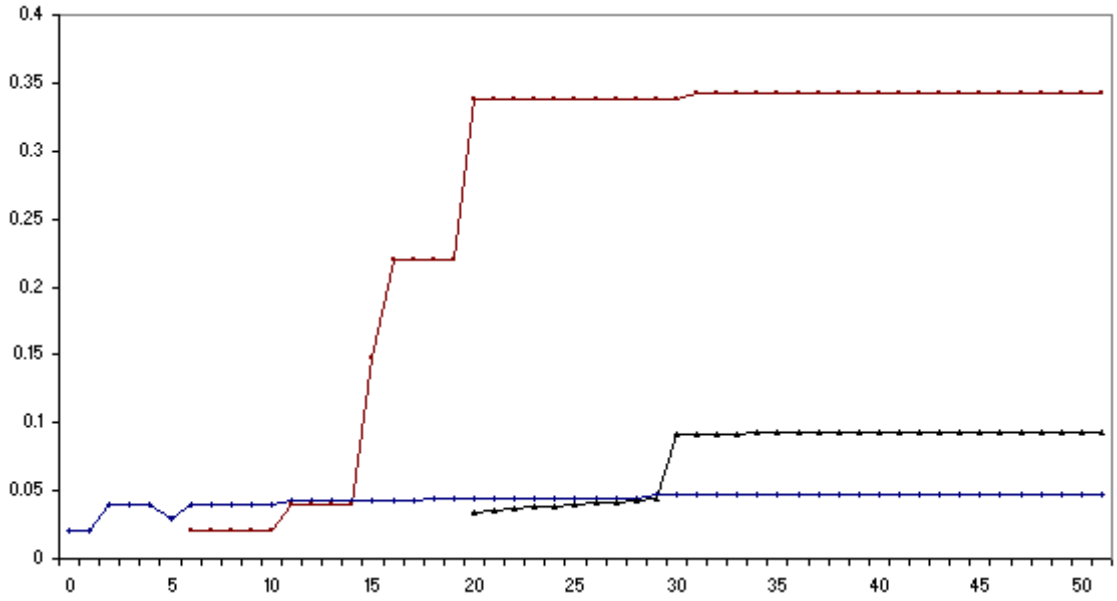

Figure 4.11. Q-Value convergence for 4x4 maze

Not only this algorithm has solved this problem, but it also allowed some ambiguity on the overlapped regions. Some areas of the cells lying on the diagonal from the top left corner are both grouped in S_1 and S_3. This is not a problem, since it is obvious that these cells have an inherent ambiguity.

4.3.3. Load Unload Problem

Figure 4.12 shows the resulting finite state automata. The world model generator has revealed whether the cart is loaded or unloaded. Also note that the optimal action indicated with arrows differs for these two states.

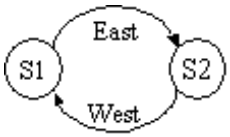

Figure 4.12. Steady State FSA for the Load Unload Problem

The Q-values for each state converged in about 7 steps. Q value for each state are given in Figure 4.13. KAFAQ could describe the world model in two states, where ARKAQ revealed nine states.

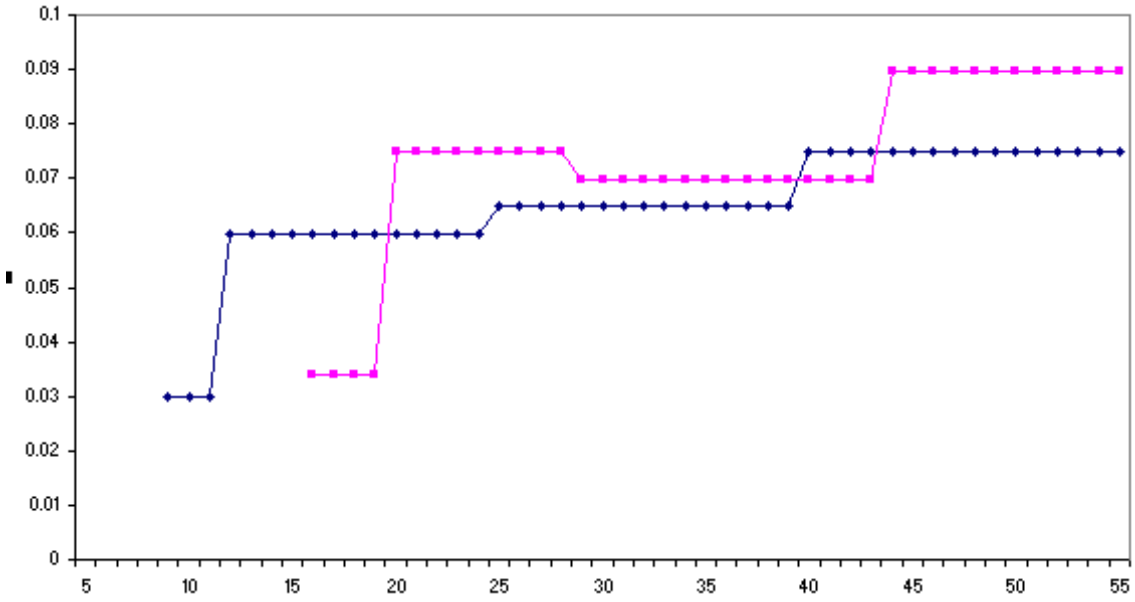

Figure 4.13. Q-Value convergence for Load Unload Problem

4.3.4. 5x5 Grid Multi Station Load-Unload Problem

As shown in Figure 4.14, the finite state automata are composed of two disconnected networks. The agent is directed to the nearest load or unload station. The FSA consists of two unconnected FSAs. Note *EAST* and *WEST* actions are extinct. Again KAFAQ could describe the world model in less number of states than ARKAQ.

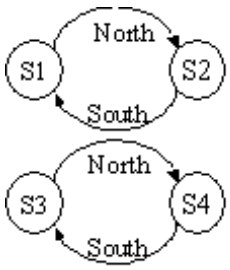

Figure 4.14. Steady State FSA for Multistation Load Unload Problem

4.3.5. Shuttle Docking Problem

KAFAQ-learning described the system with a finite state automaton of four states. After 47 steps the agent could stabilize itself between the load unload stations successfully. Figure 4.16 shows a sketch of the revealed states and a list of best actions emerging from the state and previous actions.

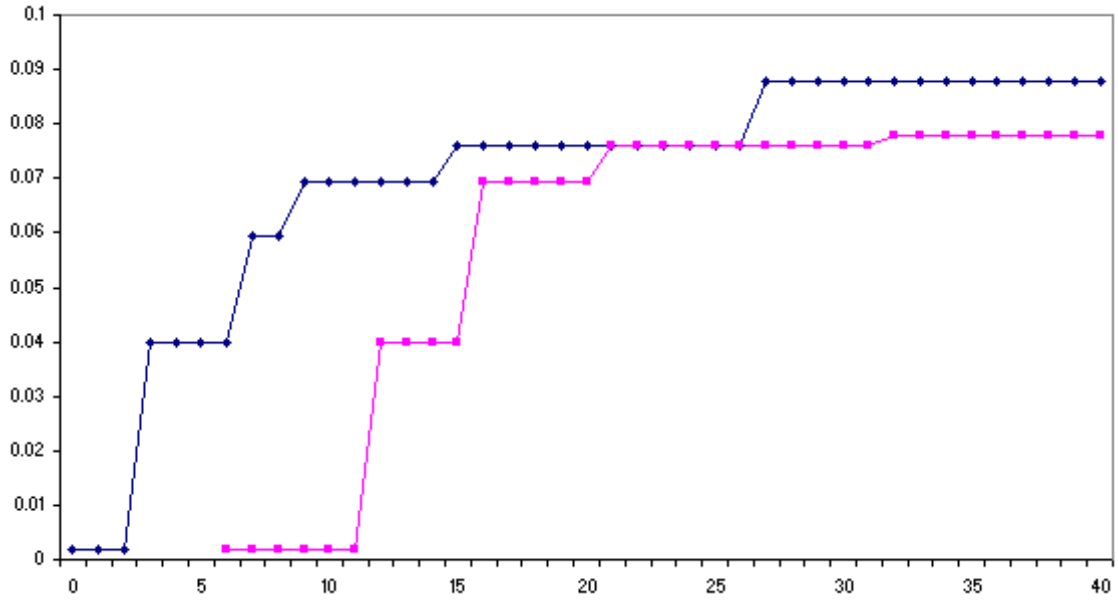

Figure 4.15. Load Unload Problem in 5x5 Grid Multistation POMDP.

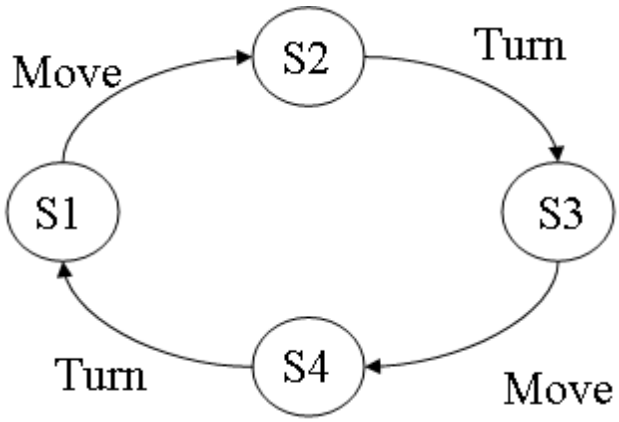

Figure 4.16. Steady State FSA for Shuttle Docking Problem

Note Move and Turn phases follow each other, forming a recursive Move Turn action sequence. Again KAFAQ could describe the world model in less number of states, than ARKAQ.

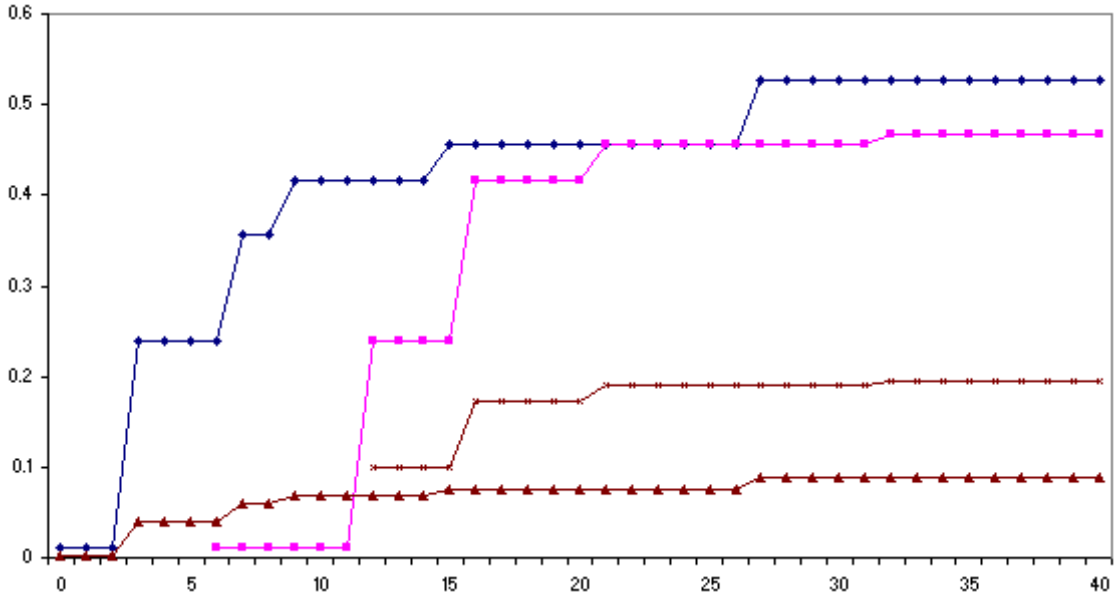

Figure 4.17. Q-Value Convergence for Shuttle Docking Problem

4.3.6. Cart and Inverted Pendulum Problem

KAFAQ could segment the input space and revealed 162 states. The pole could be balanced for more than 4 hours, where each simulation step is about 0.02 seconds. Q-Value table is given in APPENDIX A.

5. KALMAN BASED Q-VALUE CALCULATION AND POLICY GENERATION

This chapter introduces a new technique, called KBVI (Kalman Based Q-Value Iteration) to calculate Q-value for continuous POMDP problems. The Kalman filter is again used in the calculation of the current state estimate and also for assigning weights to the belief state that will arise in the Monte Carlo forward view technique. This technique enables us to shorten the long lasting exploration phases that we came up to in ARKAQ-learning and KAFAQ-learning.

We use a Kalman filter to calculate Gaussian-based POMDP belief states, under the assumption of A, H, Q and R being time-invariant. This filter returns the best state estimate and its covariance, a measure of uncertainty around the best estimate. The uncertainty gradually converges depending on the measurement, process noise and initial uncertainty value. This feature eliminates the unvisited uncertainty values from value calculation. Moreover, the predictor-corrector algorithm of Kalman filter determines from the current state estimate, the boundaries of the next estimated state at the next uncertainty level for each logically grouped action. Finally, the optimal policy is constructed, which is represented by a set of belief point backups.

5.1. The Reward Function Calculation

The reward for POMDP problems is always defined for the MDP case. For an MDP, whether discrete or continuous, the reward of taking action a for state s is constant. In the one dimensional maze POMDP problem, there exists four discrete states and the goal state is marked with a circle. The reward definition for this problem is +1 for being at the goal state and 0 elsewhere for each possible action. The reward function can be represented by a four dimensional vector [0 0 0 1]. To calculate the reward at belief state [0.25 0.25 0.25 0.25], the vector product of the belief state vector and the reward vector for MDP case is taken which is $0.25 \times 0 + 0.25 \times 0 + 0.25 \times 0 + 0.25 \times 1 = 0.25$.

The formal definition of reward $r(b)$ for a belief state b is given in Equation (5.1), where n is the number of states.

$$r(b) = \sum_{k=0}^{n-1} b_k(s) r_k(s, a) \tag{5.1}$$

Using this formal discrete definition, to calculate the reward for the belief state $N(s, P)$, we compute an infinite sum of the product of the MDP reward by its probability as shown in Figure 5.1. Since the reward in the MDP case is constant and the state probabilities are represented as Gaussian, this infinite sum is equivalent to the integral in Equation (5.2), which is equivalent to cumulative distribution function (cdf) of Gaussian across the range where the reward is non-zero. Since with 99% confidence the states lie between $2.5\times\sqrt{P}$ apart from the most probable state, we have considered only this range as an approximation. In other words we assume the integral bound is s $\pm$ $2.5\times\sqrt{P}$, where s is the most probable state.

$$\lim_{\Delta s \to 0} \sum_{s=-\infty}^{s=\infty} b(s) r_a(s) \Delta s = \int_s b(s) r_a(s) ds \tag{5.2}$$

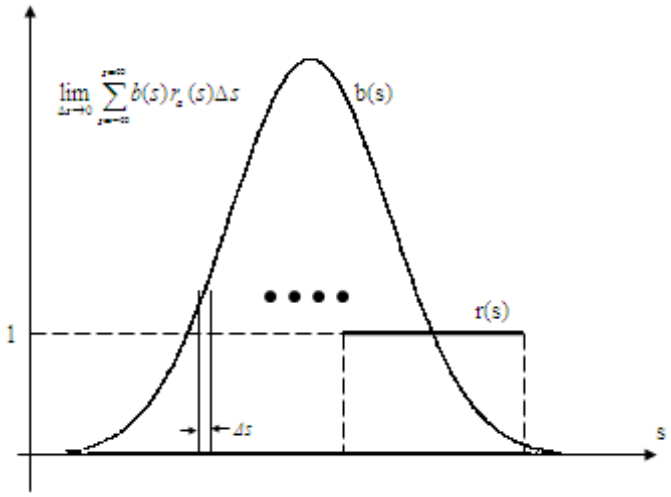

Figure 5.1. The continuous case reward value calculation at a belief state.

To exactly calculate the reward function at some uncertainty level, we need to convolve the MDP reward with all belief points. We suggest a method to approximately

calculate the reward function for the continuous case for different uncertainty levels. Since the reward for each action in the MDP case is constant over states, we adjust the edges of the reward function by an approximation. For each edge of the MDP reward, the rewards for the two points $2.5 \times \sqrt{P}$ apart from that edge are computed. Using these two reward values, a new reward function, which is a line passing through these reward values, is generated. The overall reward function for the uncertainty level P is constructed by taking the minimum of the MDP reward and the newly generated lines calculated for each edge where the MDP reward is positive and maximum when the MDP reward is negative .

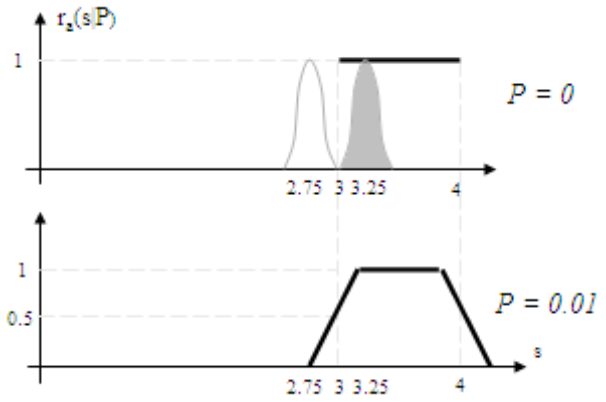

Figure 5.2. (a) MDP reward function at uncertainty 0. (b) Approximate reward function at uncertainty 0.01.

For the continuous case one dimensional maze, the reward function for MDP case and uncertainty level 0.01 is sketched in 5.2. 5.2(a) is the MDP reward function where the uncertainty is 0. 5.2(b) is the newly calculated reward function at the uncertainty level 0.01. Note that for uncertainty 0.01, $2.5 \times \sqrt{P}$ is 0.25, and the edges are approximated. For example, for the edge at location 3, the reward for 2.75 is 0 and the reward for 3.25 is +1. The minimum of generated lines for the edges at 3 and 4, and the MDP reward is shown on 5.2 (b).

5.2. Reachable Belief Region Calculation

In this section, we show how to calculate the next step reachable belief points and their weights in continuous case Gaussian-based POMDPs using internal dynamics

of the agent and Kalman filter. The value iteration algorithm in the classical discrete case uses the world parameters to calculate next step belief points and their weights. We will show that the knowledge of the agent internal dynamics and the use of Kalman filter for both next belief points and their weights is sufficient for the value iteration in the continuous case. The final result is a Gaussian probability distribution which represents the weights of reachable belief points as shown in Figure 5.3. The Kalman filter addresses the general problem of trying to estimate the state of the agent. It is governed by stochastic equations given in Equation (2.22) and Equation (2.23) Since Kalman filter computes the minimum mean square error estimate of the current state, we use it in the prediction of the next step's reachable belief range calculation. In our experiments, we linearize the systems and use time-invariant parameters for A, H, Q and R. Therefore, at each run the agent starting with the same initial state estimation error, visits the same state estimation error values or uncertainty levels. Reachable belief points for a next step all share a common covariance or uncertainty or state estimation error, P_t.

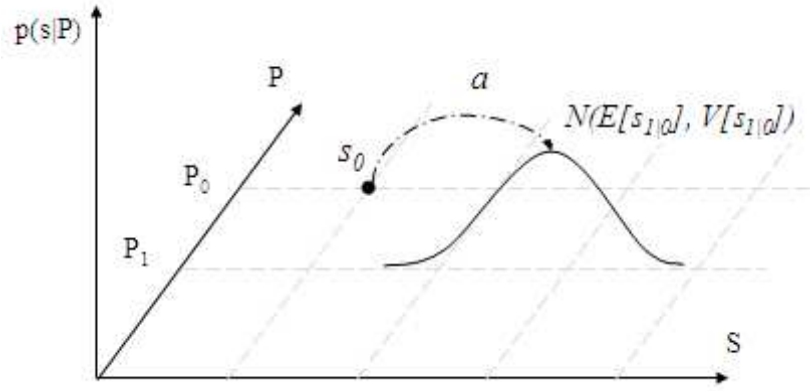

Figure 5.3. Initial belief and the next belief state distribution.

Initial belief state is a Gaussian distribution over states $N(s_0, P_0)$. It can be represented in the belief state space as a single point as shown in Figure 3.4. According to the motion model, taking action a_0 disturbed with a Gaussian noise $N(0, Q)$, will result in a Gaussian distribution over a new range of belief states with mean defined by Equation (5.3) and the covariance defined by Equation (5.4) at the uncertainty level P_1 as shown in 3.4 Since the initial belief point is given, at the uncertainty level P_0

then $V(s_0)$ is 0.

$$E(s^-_{1|0}) = AE(s_0) + BE(a_0), s.t. E(s_0) = s_0 \tag{5.3}$$

$$V(s^-_{1|0}) = AV(s_0)A^T + BV(a_0)B^T, s.t. V(s_0) = 0 \tag{5.4}$$

The state estimation in the prediction phase will be updated in the correction phase depending on the observation. The new mean of the distribution over the next uncertainty level belief states is given in Equation (5.5) and the covariance in Equation (5.6).

$$E(s_{1|0}) = E(s^-_{1|0}) + K_0(E(o_0) - HE(s^-_{1|0})) \tag{5.5}$$

$$V(s_{1|0}) = V(s^-_{1|0}) + K_0(V(o_0) - HV(s^-_{1|0})H^T)K_0^T \tag{5.6}$$

In POMDP problems like tiger, the observation of the agent is independent of the agent's location. It is dependent on the location of the tiger which is behind one of the doors. Therefore, the possible observations are represented by a Gaussian with mean as the location of the tiger and covariance as the observation noise's covariance. But in navigational type of POMDP problems like one dimensional maze (Cassandra 1999) the observation of the agent is dependent on its location. In the navigational type of POMDP problems, for a given uncertainty level, the observation probability is the convolution of two Gaussian distributions given in Equation (5.7)

$$p(o) = \int p(o|s)p(s)ds \tag{5.7}$$

Since the convolution of two Gaussian distributions is also a Gaussian distribution

(Bracewell, 1999), mean and covariance can be calculated as given in Equation (5.8).

$$\begin{aligned} E(o) &= E\left[o|s\right] + E(s) \\ V(o) &= V\left[o|s\right] + V(s) \end{aligned} \tag{5.8}$$

In navigational type of problems, the observation noise is $N(0, R)$. Consequently, *p(o|s)* is a Gaussian distribution with mean 0 and covariance R, around s. The reorganized version of Equation (5.8) is given in Equation (5.9).

$$\begin{aligned} E(o) &= E(s) \\ V(o) &= R + V(s) \end{aligned} \tag{5.9}$$

By substituting the mean and the covariance given in Equation (5.9) into Equation (5.8), we get a Gaussian distribution over the belief states at uncertainty level P_1 for the action a_0 taken at belief state $N(s_0, P_0)$ which represents the probability of next step reachable belief points. Due to the nature of Kalman filter, reachable belief points at the next step all share a common covariance or uncertainty or state estimation error, P_t. The reachable belief points covariance or uncertainty levels can be calculated using Equation (2.30), Equation (2.32) and Equation (2.34). $p(s_{t+1}|P_{t+1})$ can be calculated recursively from previous step $p(s_t|P_t)$ using the Equation (5.5), Equation (5.6) and Equation (5.9) as shown in Figure 5.4.

To illustrate the idea in the one dimensional maze problem, a single Kalman iteration is shown in Figure 5.4. A, B and H are modeled as the identity matrix. Let the initial belief be $N(s_0, P_0)$, the observation noise $N(0, R)$, and the actuator noise is $N(0, Q)$. $E[a_{RIGHT}]$ is $+1$ and as expected $E[a_{LEFT}]$ is -1. Then the next reachable belief points' probability distribution from P_0 to P_1 for a_{RIGHT} is$N(s_0 + 1, Q + K^2R)$ computed using the Equation (5.5), Equation (5.6) and Equation (5.9). This result for the next reachable belief points probability distribution complies with the result calculated by convolution of probabilities suggested by Porta, Spaan and Vlassis (Porta

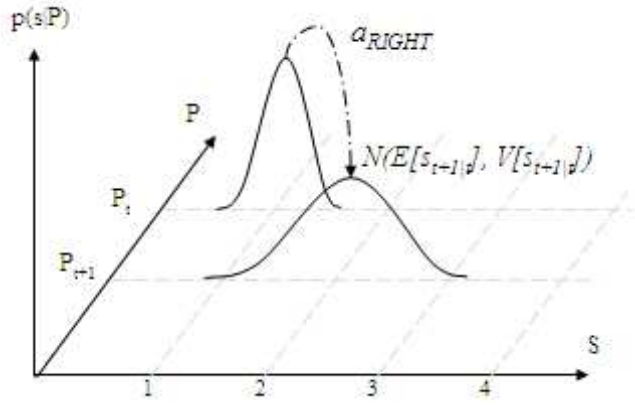

Figure 5.4. Final Gaussian showing probability of next step reachable points for one dimensional maze.

et al., 2005) which is $N(s_0 + E[a]\ ,\ \Sigma + \Sigma^a\ +\ \Sigma^o)$ Note that they calculated the belief state, whereas we calculate the probability distribution over belief states for a given uncertainty level depending on the possible set of observations and a given action. The slight difference K^2 which is the coefficient of the observation noise in our calculation is due to Kalman's measurement update equations and since K is a positive number smaller than one, our calculated covariance is always smaller.

5.3. The Horizon N Q-Value Calculation

Let the uncertainty level calculated by the Kalman Filter at time step t be P_t. Using the probability distribution of the next reachable belief points obtained by the methods described in section 5.2 and the approximate reward function for each P_t described in section 5.1 the immediate expected reward can be calculated by Equation (5.10).

$$E(r_t|P_t, a) = \int p(s_t|P_t) r_a(s_t|P_t) ds \tag{5.10}$$

To compute Equation (5.10) for an n step horizon, a tree consisting of all possible actions with depth n should be generated. Each branch in this tree, from root to leaf, represents a policy π. The horizon n Q-value for a policy π is the sum of the rewards for each node in a branch. The optimal horizon n policy is the branch that collects

the maximum reward for n steps. For each node the reward is calculated by using Equation (5.10) and multiplied with the corresponding discount factor, γ^k where k is the depth of the current node.

The complete generation of the tree requires $O(|A|^N)$ space and time complexity, where $|A|$ is the number of logically grouped actions and N is the depth or the horizon. To reduce these, we have simply pruned the branches that are never reachable, i.e. beyond world boundaries, and performed a depth first search and kept only the maximum branch ever calculated. The world boundaries are given to the agent as a hint. The boundaries at the uncertainty level are extended by $2.5\sqrt{P_{t+k}}$. A sketch of

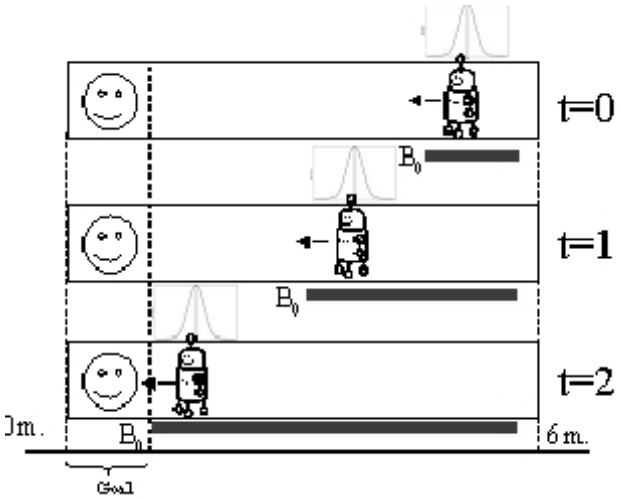

Figure 5.5. Extending a belief segment at uncertainty level $P_{converge}$.

extended boundaries along uncertainty levels for the one dimensional maze is given in Figure 5.5. In our experiments, even for the cart pole problem where horizons larger than 100 are evaluated, the evaluated branches are reduced to approximately 2^8. The space complexity of our algorithm is $O(|N|)$, and the time complexity is $O(|A|^d)$, where d is the average depth of the pruned tree.

For the computation of Equation (5.10), since the $p(s_t|P_t)$ is a Gaussian distribution denoted as $g(s)$ at uncertainty level P_t , and $r_a(s_t|P_t)$ is a linear function denoted as $l(s)$ at uncertainty level P_t, Equation (5.11) contains the product of a Gaussian

function $g(s)$with a linear function $l(s)$;

$$\int g(s)l(s)ds \tag{5.11}$$

where $g(s)$ and $l(s)$ are given in Equation (5.12) and in Equation (5.13) respectively.

$$g(s) = \frac{e^{\frac{-(s-E(s))^2}{2V(s)}}}{\sqrt{2\pi V(s)}} \tag{5.12}$$

$$l(s) = as + b \tag{5.13}$$

The solution of Equation (5.11) is given in Equation (5.14) where $\Phi(s)$ is the cdf of the Gaussian distribution. Since the approximate reward function for an uncertainty level is piecewise linear, Equation (5.14) must be computed separately for each different reward line segment. For example, in Figure 5.2 for the reward function computed for uncertainty level 0.01, Equation (5.14) is computed for three non-zero reward line segments.

$$\int \frac{e^{\frac{-(s-E(s))^2}{2V(s)}}}{\sqrt{2\pi V(s)}} \left[as + b\right] ds = -a\sqrt{\frac{V(s)}{2\pi}} e^{\frac{-(s-E(s))^2}{2V(s)}} + \left[b + \frac{aE(s)}{\sqrt{2\pi V(s)}}\right] \phi(s) \tag{5.14}$$

5.4. Policy Generation

For the policy generation strategy we have adopted several techniques. We will assume the initial belief is unimodal, meaning that it is a single gaussian distribution. Later in this chapter we will give a variation where the initial belief is composed of mixtures of gaussians. All of these techniques use Kalman-based value calculation explained throughout this chapter. The policy generation algorithms also use the convergence feature of the Kalman filter under given assumptions to calculate the approximate optimal policy.

The Kalman filter equations for the calculation of the state estimation error P only depends on the motion model matrix A, the measurement model H, the process noise Q and measurement noise R and previous state estimation error. In our experiments, we linearize the systems and use time-invariant parameters for A, H, Q and R. Therefore, at each run the agent starting with the same initial state estimation error, visits the same state estimation error values or uncertainty levels, but different most probable states, s, as shown in Figure 5.6. Moreover, Maybeck (1979) has also shown that the estimation error covariance P of the Kalman filter converges to a unique value from any initial condition where parameters A, H, Q and R are time-invariant. We call this unique value $P_{converge}$. Although $P_{converge}$ can be calculated by solving Ricatti equation, our algorithm decides on convergence if the current value is in the same ε-interval for last l steps as shown in 5.6. At each step, the agent decides on a best

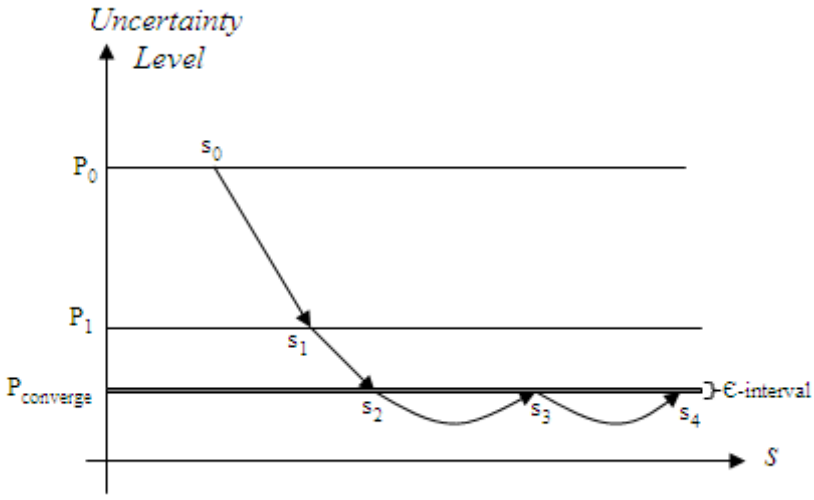

Figure 5.6. Uncertainty levels computed by Kalman Filter.

action by calculating the Q-values as explained in section 5.3. After taking the best action the agent updates its current belief state using Kalman filter, and the new belief state's Q-values and the best action are calculated. The following subsections explain different types of policy generation algorithms.

5.4.1. Belief Simplex Segmentation

In this algorithm, spatially connected belief points lying at the same uncertainty level are grouped according to their best actions as the agent explores the belief space, called belief state segments. The algorithm constructs a different policy for each uncertainty level. In our experiments the initial horizon, n is taken a value greater than the step size needed to reach $P_{converge}$. Next, all horizons up to $m > n$ are evaluated. If for horizons $m-k$ up to m the number of belief segments do not change, we conclude that the segmentation is finished and all segments are marked as committed, and are not updated anymore where a stable policy is obtained. Finally, we conclude that $m - k$ is the policy convergence horizon for the problem.

To explain the generation and merging of belief state segments let us begin with a very simple example where the agent can either move right or left horizontally in a corridor. The goal state is at the left end. Initially, the agent assumes a single belief segment bounded by the initial location estimate, s_0 and uncertainty level $P_{converge}$. Since our algorithm begins with the steady state covariance the state estimation error will be constant until the policy converges. As the agent navigates Kalman filter estimates the current state, s_1. KBVI is used to calculate the best action for each step. If the best action calculated by KBVI for the current estimate is equal to the previous state estimate's best action, then the current belief segment boundaries are extended with s_1. Figure 5.5 shows a sketch of the extension of belief segment. Finally a belief segment is created for the uncertainty level $P_{converge}$.

Each belief segment is represented by only three most probable states that have been visited as illustrated in Figure 5.7. The belief points s_1 and s_3 represent the belief segment boundaries. The belief point s_2 is the belief point with minimum value. For higher dimensions, the belief boundaries are planar.

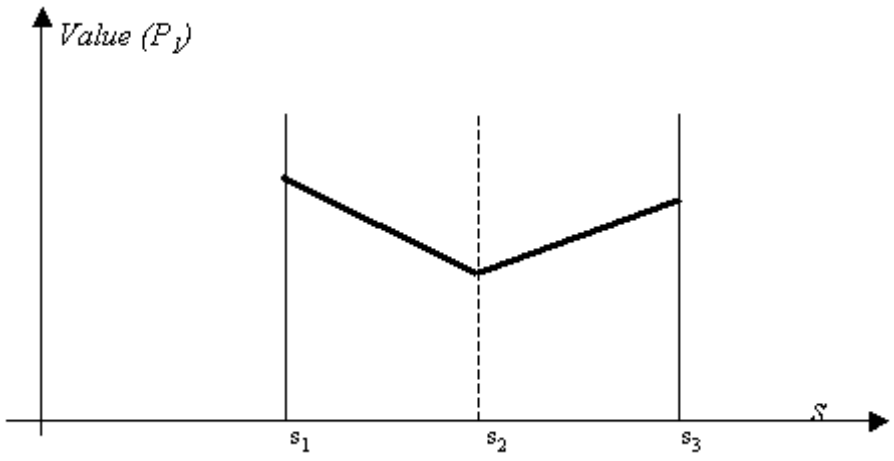

Figure 5.7. Belief segment representation at uncertainty level P_1.

5.4.2. Nearest Neighbor Policy Generation

The constructed policy is represented by a FSA. The internal representation of automaton nodes is as follows:

- Best action, a,
- Belief state, b,
- Q-value for a,
- A set of previous automaton nodes, N, from which this node is reached.
- Observation mean and variance, o_{mean}

The transition model of this automaton is deduced by taking the best action from a node. To generate nodes, at each step, the agent decides on a best action by calculating the Q-values as explained in section 5.3. After taking the best action the agent updates its current belief state using Kalman filter, and the new belief state's Q-values and the best action are calculated. In this new belief states' uncertainty level the current belief is compared with the one nearest neighbor (1-NN) (Duda et al 2001) node sharing the same uncertainty level if any. If such a point exists and has the same best action then the current belief is not stored. If such a point exists and has some other best action, then a new node is generated. Finally, if no such point exists then a new node is generated.

In our experiments the initial horizon, n is taken a value greater than the step size needed to reach $P_{converge}$. Next, all horizons up to $m > n$ are evaluated. If for horizons $m - k$ up to m the number of FSA nodes do not change, we conclude that the policy generation phase is finished where a stable policy is obtained. Finally, we conclude that $m - k$ is the policy convergence horizon for the problem.

The pseudocode of KBVI algorithm is given in Figure 5.8.

procedure $KBVI1NNPolicyGeneration(initialBelief)$

```
1: policy ← InitializePolicy
2: for horizon = 1 to horizonLength do
3:    previousPolicy ← SavePolicy(policy)
4:    k ← CheckConvergence(policy, previousPolicy, k)
5:    belief ← InitializeCurrentBelief
6:    for i = horizon downto 0 do
7:       action ← SelectActionwithKBVI(i, belief)
8:       1nnnode ← 1NNPolicyMatch(policy, belief, action)
9:       if 1nnnode = NULL then
10:         1nnnode ← AddNewPolicyNode(belief, action)
11:      end if
12:      observation ← AgentPercieve
13:      belief ← UpdateBeliefwithKalman(belief, action, observation)
14:   end for
15: end for
```

Figure 5.8. Pseudocode of KBVI Policy Generation.

5.5. Experimental Results

The algorithm is run on some well known POMDP problems, the tiger problem and the one dimensional maze problem. We have run each experiment for horizons 3 to 20. We have also applied our algorithm to the inverted pendulum problem modified as a continuous case POMDP problem for horizons 100 to 180. The discount factor for all experiments is 0.75.

5.5.1. The Tiger Problem

The tiger problem has been converted to continuous observation over discrete states case by Hoey and Poupart (2005). Although our algorithm works for continuous observation over continuous states, we modeled the tiger problem as defined in (Hoey and Poupart, 2005). The original definition is as follows. Imagine an agent sitting in front of two closed doors. Behind one of the doors is a tiger and behind the other is a large reward. If the agent opens the door with the tiger, then a large penalty is received (presumably in the form of some amount of bodily injury). Instead of opening one of the two doors, the agent can listen, in order to gain some information about the location of the tiger. Unfortunately, listening is not free; in addition, it is also not entirely accurate. There is a chance that the agent will hear a tiger behind the left-hand door when the tiger is realy behind the right-hand door with some probability, and vice versa. The actions are left, right, and listen.

Since the location of the tiger is independent of the agent's location, $E(o)$ in Equation (5.8) is equal to either -1 or $+1$, and $V(o)$ is the observation noise. The penalty of opening the door with the tiger is -100, the reward of opening the door without the tiger is $+10$ and the cost of listening is -1.For both policy generation technique, B is equal to the zero matrix, A and H are equal to the identity matrix, and there is no actuator noise as given in the problem definition.

First, we have applied 1-NN policy generation technique to the tiger problem. The observation noise for the problem is *N(0, 0.625)* as defined in (Hoey and Poupart 2005). Initially $E(s_0)$ is 0, and P_0 is 1. Since the agent may hear the tiger from -1 or $+1$ disturbed with a Gaussian noise, the agent evaluates listen action for each possible case for each horizon. On the average the agent did not open any doors until the degree of uncertainty, P_n, falls below 0.03 and the estimated location of the tiger is greater than $+1.14$ or less than -1.22 on the average. Experiments showed that the agent tends to listen for at least 12 steps, in other words, the agent did not open any door for horizons smaller than 13. For horizon 13 or higher the agent opens a door, depending on the uncertainty and the distance of the estimated state from the door locations. Horizon

13 is the policy convergence horizon for this problem. A simple sketch of the resulting FSA is shown on Figure 5.9.

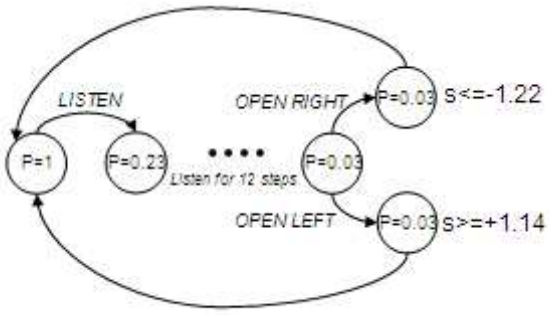

Figure 5.9. FSA for the tiger problem.

Next, we have applied belief state segmentation technique to the tiger problem with the same parameters as 1-NN. On the average the agent did not open any doors until the degree of uncertainty, P_n, falls below 0.03 and the estimated location of the tiger is greater than $+1$ or less than -1. Experiments showed that the agent tends to listen for at least five steps, in other words, the agent did not open any door for horizons smaller than 12. For horizon 12 or higher the agent opens a door, depending on the uncertainty and the distance of the estimated state from the door locations. Horizon 13 is the policy convergence horizon for this problem. A simple sketch of policy over belief state space is shown on Figure 5.10. Note that although approximately sketched, in our case we never reach zero covariance, where the problem is an MDP. For uncertainty levels greater than 0 and less than or equal to 0.03 in order to open a door, the agent has to hear from a distance greater than one.

5.5.2. The One Dimensional Maze Problem

In this problem, the goal is placed at the right end. Simulations are done using Webots simulator. A Webots screenshot is given in Figure 3.3. The agent, can move *EAST* and *WEST* or it can *STOP*.

For both policy generation technique, A, B and H are equal to the identity matrix and the sensor and the actuator noises are *N(0, 0.45)* and *N(0, 0.25)* respectively.

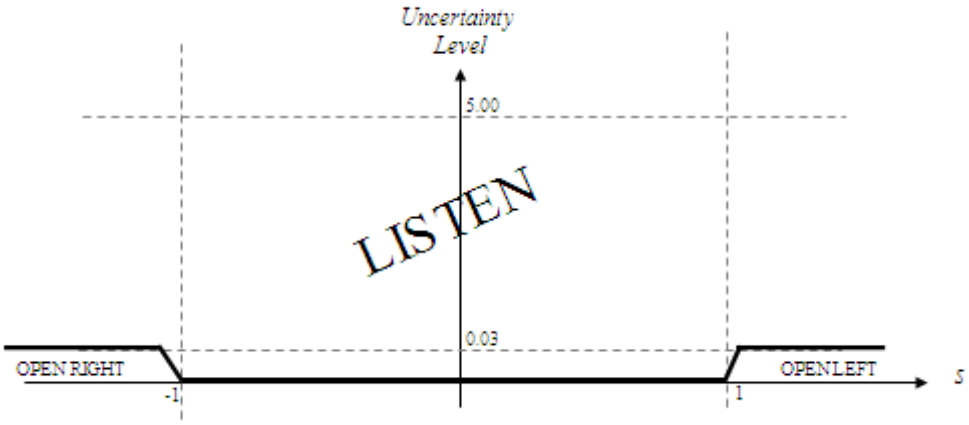

Figure 5.10. Belief segments for the tiger problem.

Initially $E(s_0)$ is 0, and P_0 is 0.30.

First, we have applied 1-NN policy generation technique to the one dimensional maze problem. For each horizon the agent preferred moving *RIGHT* till goal location. $P_{converge}$ is 0.233. Since the agent begins at zero, it could reach the goal state for horizons five or greater. Horizon five is the policy convergence horizon for this problem. A sketch of the representative belief points are shown on Figure 5.11.

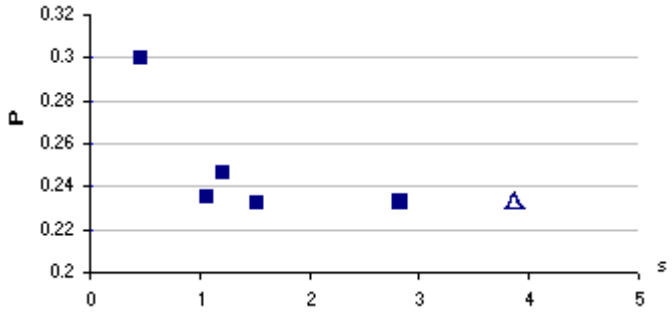

Figure 5.11. Representative belief points for the one dimensional maze.

A sketch of the policy FSA is shown on Figure 5.12.

For uncertainty levels greater than 0 and less than or equal to 0.233, as the uncertainty level increases, the agent will stop closer towards right, in other words the agent stops closer to the right end.

Next, we have applied belief state segmentation technique to the tiger problem

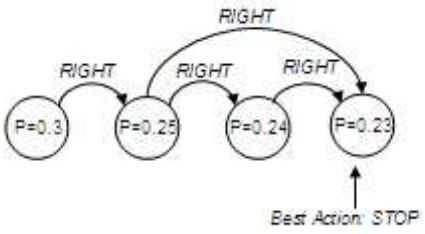

Figure 5.12. 1-NN FSA for the one dimensional maze.

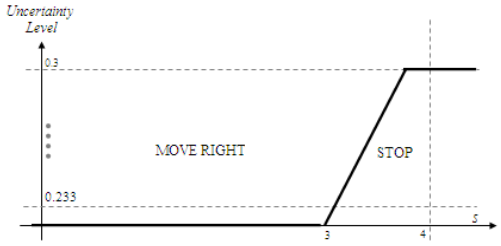

Figure 5.13. Belief segments of the simplex for one dimensional maze.

with the same parameters as 1-NN. In this technique, again, for each horizon the agent preferred moving *RIGHT* till goal location. $P_{converge}$ is 0.233. Since the agent begins at zero, it could reach the goal state for horizon five or higher. Horizon five is the policy convergence horizon for this problem. Belief segments for this solution is given in Figure 5.13.

5.5.3. The Two Dimensional Maze Problem

For the two dimensional maze defined in sectin 3.4.2, the reward is +1 for the goal cell, 0 otherwise. The agent has two range finders disturbed with a Gaussian noise, one at the front and the other at the right. A, B and H are equal to the identity matrix, as given in the problem definition and the observation and actuator noise is disturbed with noises $N(0, 0.20) \times I$, where I is the identity matrix, and $N(0, 0.15) \times I$ respectively. The discount factor is 0.75. Initially $E(s_0)$ is $0 \times I$, and P_0 is $5.0 \times I$ which is a quite large uncertainty level. Only the actions *SOUTH* and *EAST* are favored in the final policy. *EAST* and *WEST* are never favored.

Experiments with belief simplex segmentation policy generation technique showed that the segments close to the top right diagonal favored *SOUTH* or *EAST* as the

belief points diverged from this diagonal towards the bottom edge, *EAST* is favored and likewise as the belief points diverged from this diagonal towards the right edge, *SOUTH* is favored. The state estimation errors are segmented into three uncertainty levels. For the highest uncertainty level, there is only one belief segment favoring *SOUTH*, for the second level there are six segments and for the lowest uncertainty level there are 11 belief segments. For horizons one to five, the agent acted randomly. For horizon six or higher the agent decided on an action toward the goal location. Since the agent begins at top left it could reach the goal state for horizon six or higher. Horizon six is the policy convergence horizon for this problem.

5.5.4. The Load Unload Problem

In this very challenging problem as defined in section 3.4.3, the agent is a cart designed to shuttle loads back and forth between two end-points on a line of 5 units distance. At the left end we pick up an item from an infinite pile. At the right end, after 4 units, we drop our item. At the left end, before 1 unit, we pick our item. The cart does not have sensors to indicate whether it is loaded or unloaded, but it can determine its position on the line. To act optimally the belief state must encode whether we have an item or not. Actions are simply move *WEST*, -1 meters, move *EAST*, +1 unit, *LOAD* or *UNLOAD*. Moving in an impossible direction leaves the agent where it is. A reward of +1 is received for picking up when unloaded or putting down when loaded, but only one item can be carried at a time. H and A are equal to the identity matrix as stated in the problem definition. For actions *WEST* or *EAST*, B is the identity matrix, for actions *LOAD* or *UNLOAD*, B is zero. We used 1-NN technique for policy

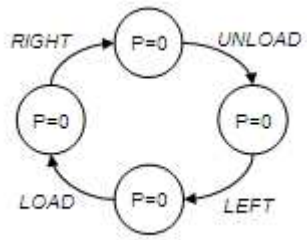

Figure 5.14. FSA for the load unload problem.

generation. The sensor and the actuator are not noisy, so this experiment demonstrates

the efficiency of the FSA generation layer only. Initially $E(s_0)$ is 0, and P_0 is 0. $P_{converge}$ is 0. The resulting FSA is shown on Figure 5.14. The policy could reveal the hidden state, whether loaded or unloaded. The previous states kept in FSA provide an implicit memory.

5.5.5. Cart and Inverted Pendulum Problem

We have applied our algorithms to the well known Cart Pole or Inverted Pendulum Problem. The task is to balance an inverted pendulum hinged to a cart by accelerating and slowing down the cart, as sketched in Figure 3.17. If the pole falls the episode ends with a failure and the cart is reset. If the cart moves beyond given limits, the cart is reset with a failure too.

The cart with an inverted pendulum is bumped with an impulse force, F. We assume that the pendulum does not move more than a few degrees away from the vertical. For this linearized model, the equilibrium ranges for the pendulum's angle and the cart's position are rewarded by 1, 0 otherwise.

For this experiment, cart pole parameters are given in section 4.3.6. H in the sensor model is the identity matrix. Actuator noise is $N(0, 0.45) \times I$, and the sensor noise is $N(0, 0.05) \times I$. We assume initial belief uncertainty $5.0 \times I$, which is quite large meaning that the agent is highly uncertain of its location.

First we applied belief simplex policy generation technique, For the higher uncertainty levels, i.e. higher than 0.045, the Q-values for the two actions are similar, where the agent acts randomly. The agent could keep the pole balanced for more than 1,500,000 steps. The state estimation errors are segmented into three uncertainty levels. For the highest uncertainty level, one belief segment exists, the Q-values for the two actions are similar, where the agent acts randomly. The second level is segmented into 41 segments. For the lowest uncertainty level, the four dimensional state space is segmented into 122 segments on the average. The final segments exhibit a fragmented belief space. Closely spaced segments sharing a common best action may be further

grouped by a smoothing algorithm, lowering the number of belief segments. The policy convergence horizon for this algorithm is 175.

Next, we applied 1-NN policy generation technique. The agent could keep the pole balanced for more than 800,000 steps. The state estimation errors are segmented into three uncertainty levels. For the highest uncertainty level, one node exists, the Q-values for the two actions are similar. The second level has 38 nodes. For the lowest uncertainty level, there are 116 nodes. The policy convergence horizon is 171.

5.6. Wake-Up Agent Problem and Multi-Hypothesis Tracking

Kidnapped agent problem defines a situation for a maze type of problem in which an agent is carried to an arbitrary location during its operation. The *wake-up* problem is the special case of the kidnapped agent problem in which the agent is told that it has been carried away. The wake-up problem can easily be converted to POMDP problem where the agent's initial belief is a probability distribution over ambigous or similar belief states. In this case, since belief states are represented as gaussian distributions, the initial belief which is a probability distribution over belief states, is represented as mixture of gaussian distributions given in Equation (5.15), where weights satisfy $w_j > 0$ and $\sum_j w_j = 1$.

$$b(s) = \sum_j w_j N(s_j, P_0) \tag{5.15}$$

KBVI algorithm for this case is slightly changed. Each gaussian component in the mixture given in Equation (5.15) is considered as a hypothesis, H, that is the algorithm tracks multi-hypothesis. The algorithm is given in Figure 5.15. KBVI is run for each possible initial belief state, whereas the policy includes all of the unimodal policies for each possible initial belief states. Once a policy is obtained the agent is kidnapped and signaled that it has been kidnapped (wake-up). Now the agent has to track multiple hypothesis. Assuming the hypothesis that has the farthest initial belief point is valid, the agent iteratively takes an action according to the valid hypothesis and updates all hypothesis weights according to the current observation, $o_{current}$. In KBVI we had

procedure $MultiTrackingKBVI(initialBeliefs)$

1: **for all** $initialBelief \in initialBeliefs$ **do**
2: $\quad policy \leftarrow policy \cup GenerateKBVIUnimodalPolicy(initialBelief)$
3: **end for**
4:
5: $multiModalHypothesis \leftarrow KidnapAndWakeUp$
6: $validHypothesis \leftarrow GetHypothesisFarthestToGoal(multiModalHypothesis)$
7: $goalIsReached \leftarrow FALSE$
8: $stepCount \leftarrow 0$
9: **repeat**
10: $\quad stepCount \leftarrow stepCount + 1$
11: $\quad a \leftarrow GetActionKBVIUnimodalPolicy(validHypothesis)$
12: $\quad multiModalHypothesis \leftarrow UpdateHypothesisCurrentBeliefs(a)$
13: $\quad o \leftarrow ObserveWorld()$
14: $\quad multiModalHypothesis \leftarrow UpdateHypothesisWeights(o)$
15: $\quad pruneHypothesisUnderThreshold(multiModalHypothesis, \theta)$
16: $\quad mergeHypothesisHavingSameState(multiModalHypothesis)$
17: $\quad$ **if** $IsPruned(validHypothesis)$ **then**
18: $\quad\quad validHypothesis \leftarrow GetHypothesisFarthestToGoal(multiModalHypothesis)$
19: $\quad$ **end if**
20: **until** $goalIsReached$ or $stepCount < maxStepCount$
21: **if** $goalIsReached$ and $Length(multiModalHypothesis) > 1$ **then**
22: $\quad multiModalHypothesis \leftarrow validHypothesis$
23: **end if**

Figure 5.15. Pseudocode of wake-up handling of KBVI.

stored a normally distributed observation means and variances, for each belief point backup in the policy. Each hypothesis weight is updated using the formula given in Equation (5.16), where $p_i(o_{current})$ is the probability of getting $o_{current}$ if i^{th} hypothesis, H_i, was valid. Hypothesis that have weights lower than a threshold are immediately pruned. If the current valid hypothesis is pruned then a new hypothesis with the highest weight is considered as valid.

$$w_i = \frac{w_i p_i(o_{current})}{\sum_j w_j p_j(o_{current})} \tag{5.16}$$

5.6.1. Experimental Results

We have applied the multi-hypothesis tracking KBVI to two different mazes. We named the mazes according to their shapes, N-shaped and U-shaped maze problem. Both maze have dimensions of 5x5 units. The reward is selected as -1 for walls and +1 for the goal location. The discount factor is 0.75 for all experiments. For both experiments the rejection threshold weight is 0.05.

5.6.1.1. N-Shaped Maze Problem. The maze is given in Figure 5.16. The goal is marked with a star. The agent, like in two dimensional maze problem, can move in four directions *EAST, WEST, NORTH* and *SOUTH*. A, B and H are equal to the identity matrix and the sensor and the actuator noises are *N(0, 0.45)* and *N(0, 0.45)* respectively. The initial unimodal beliefs of KBVI are $E(s_0)$ is $(0.5, 0.5)$ and $(2.5, 0.5)$ with $P_0 = 0.30$. Then, the agent is knapped to one of the initial belief points. The agent reduced the multi-hypothesis count to one after the first step. This is due to the fact that the KBVI policy had already grouped the second uncertainty level under a single action *NORTH*.

5.6.1.2. U-Shaped Maze Problem. The maze is given in Figure 5.17. The goal is marked with a star. The agent, like in two dimensional maze problem, can move in four directions *EAST, WEST, NORTH* and *SOUTH*. A, B and H are equal to the

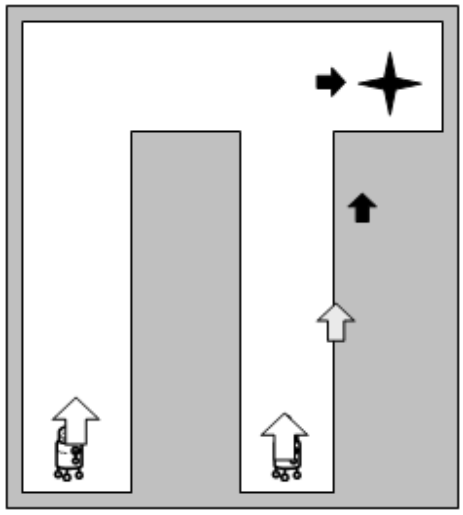

Figure 5.16. N-Shaped Maze Problem

identity matrix and the sensor and the actuator noises are *N(0, 0.45)* and *N(0, 0.45)* respectively. The initial unimodal beliefs of KBVI are $E(s_0)$ is $(1.5, 1.5)$, $(2.5, 0.5)$ and $(2.5, 1.5)$ with $P_0 = 0.30$. The generated policy is given in Table 5.1. Then, the agent is knapped to one of the initial belief points.

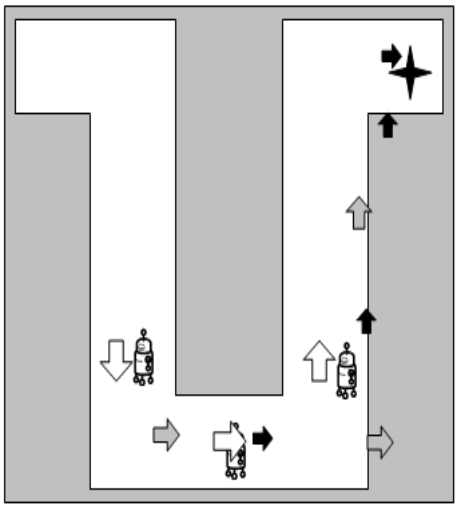

Figure 5.17. U-Shaped Maze Problem

If the agent kidnapped to the farthest location to the goal, the agent wakes up and begins running with its already generated policy with the assumption that it is at the farthest initial belief point. The agent always chooses *SOUTH* whereever its current real location is.

In case of the agent is really at p_{31} or p_{32} and moves to the south, we end up

Table 5.1. U-Shape Maze Point Backups

Uncertainty Value	Point Name	Point Backup	Action
0.3	p_{31}	(1.5, 1.5)	SOUTH
	p_{32}	(2.5, 0.5)	EAST
	p_{33}	(2.5,1.5)	NORTH
0.28	p_{21}	(2.8, 0.5)	EAST
	p_{22}	(4.1, 0.4)	EAST
	p_{23}	(3.9, 2.9)	NORTH
0.27	p_{11}	(3.8, 0.5)	EAST
	p_{12}	(3.9, 1.8)	NORTH
	p_{13}	(4.3, 3.9)	NORTH
	p_{14}	(4.3, 4.6)	EAST

with two valid hypothesis p_{21} with higher probability and p_{22}. Then the agent moves to *EAST*. Now, at the third move we have two hypothesis, either p_{11} and p_{12}. The hypothesis p_{12} is rejected since its weight has fallen below the threshold and thereafter continues with the unimodal policy where the agent will navigate to *EAST*, *NORTH* and *EAST*.

In case of the agent is really at p_{33} and moves to the south, we end up with two valid hypothesis p_{21} and p_{22} with higher probability. The hypothesis p_{21} is rejected since its weight has fallen below the threshold and thereafter continues with the unimodal policy where the agent will navigate to *EAST*, *NORTH* and *EAST*.

6. COMPARISON OF ALGORITHMS

Although all of the algorithms are based on reinforcement learning, ARKAQ-learning and KAFAQ-learning are based on explorative TD learning techniques. On the other hand, Kalman based value iteration techniques are based on Monte Carlo estimation of expected future reward. The major differences between TD learning and Monte Carlo estimation type of methods are:

- TD learning methods take much longer time to converge, and the policy is very dependent on the initial policy and exploration-exploitation technique. Monte Carlo methods do not need an exploration phase. Besides the policy convergence horizon is the step or time needed to obtain the approximate optimal policy and is very low compared to the time needed for TD learning exploration phase.
- With TD learning methods, multiple runs and tuning are necessary to obtain a valid and stable solution. Monte Carlo methods do not require a tuning phase.
- Experiments showed that the expected discounted reward for Monte Carlo methods are higher than TD learning methods.

To compare the belief simplex segmentation and 1-NN policy generation in terms of space complexity, belief simplex segmentation requires three times as much as 1-NN policy generation space requirements. On the other hand belief simplex segmentation obtained higher expected discounted reward in classical POMDP problems and kept the pole balanced much longer than 1-NN. The comparative expected discounted rewards for the tiger and the one dimensional problems are given in Table 6.1.

Table 6.1. Comparison of Expected Rewards Belief Simplex and 1-NN Segmentation

	Tiger	1D Maze
Optimal Policy (MDP)	9	1
Belief Simplex Segmentation	6.92	0.93
1-NN	5.89	0.47

Moreover, in the continuous case the value function is not PWLC yet is a non-

linear function over the belief simplex, the policy change point as shown in Figure 6.1 cannot be calculated from α-vectors (Porta et al 2005) intersections as point based methods of the discrete case. It seems more logical for the continuous case to keep point backups with the corresponding Q-values and state segmentation methods based on point backups or belief simplex segmentation method will reveal an approximate policy change point. Moreover, KBVI can solve multi-hypothesis tracking problems.

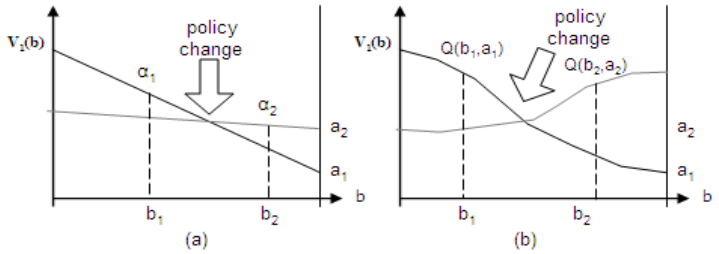

Figure 6.1. Policy change points for (a) discrete (b) continuous case.

In the cart-pole problem ARKAQ gave a stack overflow error, i.e. the state size exceeded the memory capacity. On the other hand, thanks to state representation and pruning phase, KAFAQ-learning converged to 162 states on the average and successfully stabilized the pole.

ARKAQ-learning has to wait the world model generator in order to calculate the optimal policy. In other words, the decision layer tuning begins only when the state segmentation phase is over. On the other hand, KAFAQ-learning handles state segmentation and policy generation in parallel which reduces time to reach the optimal policy.

Applied to the same problem, with common world parameters, it took ARKAQ to converge, about three to four times longer than KAFAQ.

Both algorithms have space complexity of $O\ (|A||S|^2)$ in the worst case. But the pruning phase of KAFAQ-learning prevents it to reach the boundary conditions. Moreover, since the policy and world model generator works in parallel in KAFAQ-learning is better in time complexity. Even without prior knowledge such as vigilance

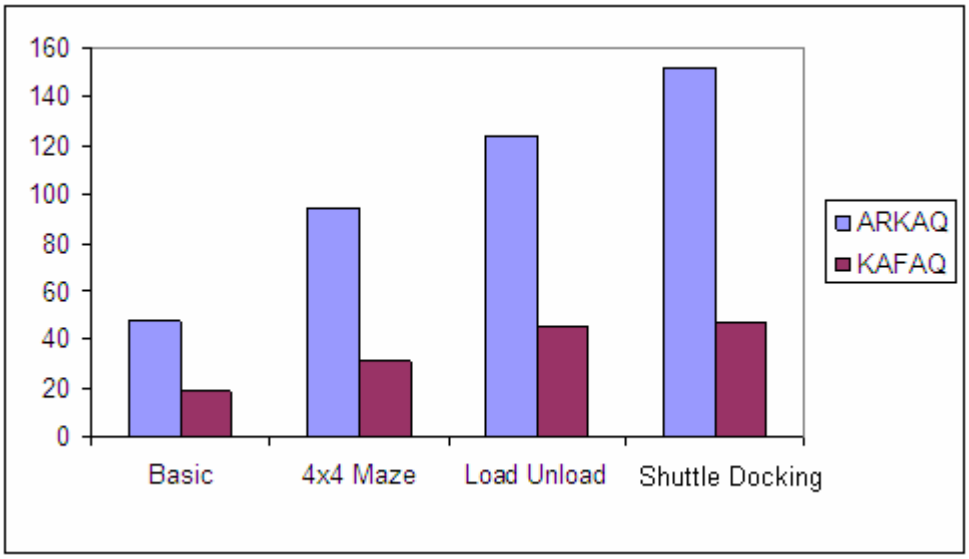

Figure 6.2. Comparison of ARKAQ and KAFAQ time of convergence values

parameter KAFAQ-learning worked better in terms of number of states and time to converge for each experiment than ARKAQ-learning.

The Q-Values obtained from the experiments, for both ARKAQ and KAFAQ Learning techniques, have converged to a value. Non oscillating feature of these results show the resulting policies are stable.

7. CONCLUSIONS

The problem of planning under uncertainty is relevant to a large number of fields, from manufacturing to robotics to medical diagnosis. In the area of robotics, it is generally understood to mean the ability to produce action policies that are robust to sensory noise, imprecise actuators and so on. This is imperative for robot systems deployed in real-world environments.

POMDP offers a rich framework for performing planning under uncertainty. It can be used to optimize sequences of actions with respect to a reward function, while taking into account both action and state uncertainty. POMDPs can be used to model a large array of robot control problems. However, finding a solution in reasonable time is often impossible, even for very small problems. One of the key obstacles to increased scalability of POMDPs is the curse of history, namely the fact that the number of information states grows exponentially with the planning horizon. Another obstacle is the curse of dimensionality that involves learning from few data samples in a continuous feature space. Moreover, POMDP solution techniques assume that the world model is given, which is an unrealistic assumption.

It is the focus of this thesis to develop computationally tractable solutions for continous state space POMDP problems, where there are infinitely many dimensions. We have offered efficient approaches that overcome the two obstacles mentioned above. These three new approaches are named ARKAQ-learning, KAFAQ-learning and KBVI. ARKAQ makes use of ART2-A Networks augmented with Kalman Filters and Q-learning, KAFAQ is a finite state automaton using Kalman Filters and Q-learning. Both are online algorithms and have space and computational complexity of order $O(|A||S|^2)$ in the worst case. KAFAQ-learning performed better in the simulation steps to reach the optimal policy. Moreover, the success of the optimal policy generated by ARKAQ-learning, depends on the vigilance parameter. KAFAQ-learning and ARKAQ-learning are based on TD learning and therefore depends on a time consuming exploration phase. On the other hand KBVI is based on Monte Carlo methods has

no exploration phase.

ARKAQ-learning architecture consists of two layers. The first layer is a world model generator that incrementally segments internal belief simplex for a given task. The second layer is a policy generator that proposes actions that deal with each state segment in order to maximize expected discounted reward. But tuning at the second layer has to wait for the first layer finish the state segmentation. In KAFAQ-learning these two tasks are interleaved reducing the required time to obtain a policy.

Throughout this study, due to the following reasons we came up with another approach, namely KAFAQ-learning:

- When a large amount of noisy data are inherently involved, ARKAQ may create irrelevant clusters continuously depending on the vigilance parameter choice.
- Using a static vigilance parameter, means that the states have equal ranges. This increases the number of states discovered, depending on the chosen vigilance parameter.
- The only choice for the experimenter to find the optimal vigilance parameter is to repeat the experiment each time with different vigilance value until the optimal solution is obtained as the result of the experiment.
- In continuous environments, parameters which causes long exploration phase combined with the unnecessary categorization caused overflow in the number of discovered states.
- It is very difficult to define a merging or pruning criteria for an algorithm that keeps the cluster centers instead of their range. This means there is a serious problem of state representation.

At the previous algorithm single previous state is fed as one component of the input vector. As a result the ARKAQ-learning categorized separately a single state, which can be reached from many previous states, which caused an overflow in the number of states in continuous domain problems

KAFAQ is an online algorithm that works on continuous states POMDP problems where the world model is not given. It is an approximate approach consisting of two layers, where it yields a stationary policy in infinite horizon in finitely transient problems. The first layer uses Kalman filter to estimate the current belief state by recursively conditioning on previous estimates. The second layer is responsible for state segmentation and policy generation. The final result is a finite state automaton.

To overcome the problem of continuous state space, KAFAQ-learning segments the continuous space into closed connected regions, where one best action dominates. As the agent explores, these regions may be extended, divided or merged according to their dominant actions. If the dominating action remains the same although the agent travels outside the region, the state boundaries are extended. Instead of representing states with cluster centers rather we moved to cluster range representation with a common dominating action, which is common in real world problems. The algorithm iterates until a finite transient policy is obtained, i.e. until the policy representation over states does not change for several iterations. The internal state and policy representation is a finite state automaton (FSA) where states are compactly represented compared to ARKAQ-learning algorithm where a single state which can be reached from many previous states categorized separately which caused an overflow in the number of states in continuous domain problems.

Both algorithms are effective for solving continuous state space POMDP problems. They can address a wide range of problems, with varying levels of uncertainty, from the localization uncertainty exhibited by the maze domains, to the miscellaneous control problems like inverted pendulum. They are based on TD learning, where KBVI depends on forward view technique.

Continuous state space POMDPs where world model parameters are not given, necessary for good robot control, are very challenging to be solved exactly. However, many problems naturally exhibit strong structural properties. By designing algorithms that exploit such structure, it is possible to produce high quality approximate solutions in reasonable time.

All approaches consider the leveraging of structural constraints in POMDPs from many perspectives, from sparse belief space sampling, to explicit policy generation, to automatic belief simplex segmentation and observation abstraction. These provide powerful approximation possibilities for POMDP solving. Taken together, these techniques offer a new perspective to the design and development of planning and control systems that are scalable, modular, and robust to uncertainty. Moreover, multi-hypothesis KBVI can handle the challenging agent wake-up problems under uncertainty.

Vigilance parameter in ARKAQ is a priori knowledge. On the contrary we required our agent to learn autonomously, without any prior knowledge if possible. A further research may be done to make the ARKAQ-learning automatically learn the optimum vigilance parameter. Maybe we may come up to a dynamic vigilance parameter that the agent may change its mode if lost. According to the vigilance parameter, agent may have revealed too many states sharing common optimal action. T overcome this problem we plan to add pruning phase to ART Networks. We will also investigate the two algorithms with $Q(\lambda > 0)$.

The comparison results for TD learning and Monte Carlo forward view methods show that Monte Carlo forward view methods performs better in terms of time to converge and expected discounted rewards. Likewise, belief simplex segmentation method yields higher expected discounted reward compared to 1-NN policy generation. Moreover, slightly modified KBVI algorithm can handle informed kidnapped agent problem (wake-up agent problem). KBVI algorithm uses Kalman filters, but most real world environments are nonlinear.

As future work we plan to extend this algorithm for nonlinear domains, where the convolutions and integrations will be planar rather than linear. We also plan to incorporate time variant reward mechanisms to POMDP problems and also model the agent's internal dynamics by incorporating time variant parameters to the belief state update layer.

APPENDIX A: Cart Pole Steady State Q-Values

KAFAQ algorithm for the cart pole problem revealed 162 states. The pole could be balanced for more than 4 hours, where each simulation step is about 0.02 seconds. The Q-Values for these states are given in Table A.1.

Table A.1: Cart Pole Q-Values in KAFAQ.

State	**Action: Push left**	**Action: Push right**
1	-0.01087	0.00377
2	0.00465	-0.01565
3	0.00062	0.00729
4	-0.01678	-0.01781
5	0.00896	0.00416
6	0.00713	-0.01716
7	-0.01036	0.00871
8	0.00786	0.00253
9	-0.01187	0.00847
10	-0.01584	-0.01161
11	0.00671	0.0067
12	-0.0148	-0.01984
13	0.00917	-0.01926
14	0.00991	0.00556
15	-0.01847	-0.01642
16	-0.01321	0.00127
17	-0.01886	0.00991
18	0.00541	-0.01361
19	-0.01084	0.00766
20	0.00416	0.00309
21	0.00763	-0.01475
22	0.00842	-0.01157

Table A.1 – continued

State	Action: Push left	Action: Push right
23	-0.01642	-0.01266
24	-0.01756	-0.01883
25	0.00286	-0.01667
26	-0.01351	0.00959
27	-0.01021	-0.01772
28	0.00385	-0.01169
29	-0.01487	-0.01723
30	0.00978	0.00594
31	0.00712	-0.01517
32	0.00004	-0.01815
33	-0.01577	0.0054
34	-0.01346	0.00014
35	0.00383	-0.01763
36	-0.0176	-0.01227
37	-0.01461	0.00517
38	0.00874	0.00222
39	0.00442	0.00597
40	0.00468	0.00863
41	-0.01295	0.00543
42	0.00746	0.00354
43	-0.0195	-0.01605
44	-0.01375	0.001
45	0.00588	0.00904
46	-0.01916	-0.01657
47	-0.01718	-0.0159
48	0.00453	0.00798
49	0.00303	-0.01458
50	-0.01631	0.00373
51	0.00639	0.00317

Table A.1 – continued

State	Action: Push left	Action: Push right
52	-0.01194	0.00001
53	0.00418	-0.01024
54	-0.01133	-0.01932
55	0.00979	-0.01516
56	-0.01016	0.00916
57	0.00453	0.00383
58	0.00705	0.00244
59	0.00722	-0.01073
60	-0.01037	0.00578
61	-0.01279	-0.01445
62	-0.01216	-0.01434
63	-0.01815	0.00163
64	0.00519	0.0009
65	-0.01727	-0.01266
66	0.00612	0.00284
67	-0.01358	-0.01834
68	-0.01194	-0.01288
69	-0.01588	0.00039
70	-0.01771	-0.01298
71	0.00298	0.00561
72	0.00043	-0.01302
73	0.00328	0.00617
74	-0.01653	-0.01189
75	0.0002	-0.0134
76	-0.01974	0.00509
77	0.00247	0.00553
78	0.00848	0.00912
79	0.00319	-0.01729
80	-0.01981	-0.01206

Table A.1 – continued

State	Action: Push left	Action: Push right
81	0.00881	-0.01503
82	0.00314	0.00792
83	0.00382	0.00297
84	0.00413	0.00169
85	-0.01227	0.00902
86	0.00859	0.00228
87	0.00604	0.00527
88	0.00003	0.00868
89	-0.0139	0.00372
90	0.00281	-0.01768
91	-0.01953	0.00016
92	-0.01374	-0.01796
93	-0.01903	-0.01304
94	0.00999	0.00941
95	0.00172	-0.01025
96	-0.01811	0.00434
97	-0.01041	0.00645
98	-0.01881	0.00476
99	-0.01546	0.00242
100	0.00511	-0.01529
101	0.00633	0.00734
102	-0.01503	-0.01659
103	-0.01121	0.00213
104	-0.01427	-0.01401
105	-0.01851	-0.01869
106	0.00301	-0.01783
107	-0.01402	-0.01949
108	0.00754	-0.01678
109	-0.01575	-0.01768

Table A.1 – continued

State	Action: Push left	Action: Push right
110	0.00303	-0.01517
111	-0.01048	-0.01488
112	0.00484	-0.01692
113	0.00859	0.00118
114	-0.01335	0.00672
115	0.00491	-0.011
116	-0.01421	-0.01373
117	-0.01051	0.00775
118	-0.01654	-0.01741
119	-0.01	0.005
120	0.00244	-0.0174
121	0.0029	0.00024
122	-0.01745	-0.01022
123	0.00342	0.00028
124	0.00909	0.00346
125	0.00455	0.00377
126	-0.01746	0.00577
127	0.00552	0.00608
128	-0.01809	-0.01972
129	0.00007	0.00223
130	-0.01002	0.00292
131	-0.0167	0.00261
132	-0.01729	-0.01054
133	0.00541	-0.01864
134	0.00282	0.00295
135	0.00492	-0.01508
136	-0.01912	0.0098
137	-0.01664	0.00864
138	-0.01082	0.00048

Table A.1 – continued

State	Action: Push left	Action: Push right
139	0.00435	-0.01839
140	0.00863	0.00791
141	0.00251	-0.01951
142	0.00245	-0.0128
143	0.00462	0.00033
144	-0.0136	-0.0101
145	-0.01314	-0.01647
146	0.00671	0.00662
147	0.00859	-0.01167
148	0.00652	-0.01617
149	0.00118	-0.01833
150	0.00869	0.00621
151	0.00871	0.00543
152	-0.01986	-0.01928
153	-0.01501	-0.01184
154	0.00517	0.0009
155	-0.01147	-0.01982
156	0.00249	-0.0163
157	-0.01382	-0.01707
158	0.00157	0.00814
159	0.00409	0.00886
160	-0.01893	-0.01273
161	-0.01452	-0.01432
162	-0.01634	-0.01377

REFERENCES

Aström, K. J., 1965, Optimal control of markov decision process with incomplete state estimation. *Mathematical Analysis and Applications*, Vol. 10, pp. 174–205.

Baird, L. C., 1993, *Advantage updating*, Wright Laboratory Technical Report WL–TR–93–1146.

Baird, L. C., 1995, "Residual algorithms: Reinforcement learning with function approximation", In *Proceedings of the Twelfth International Conference on Machine Learning*, pp. 30–37.

Bartlett, P. L. and J. Baxter, 2000, "Estimation and approximation bounds for gradient based reinforcement learning", In *Thirteenth Annual Conference on Computational Learning Theory*, pp. 133–141.

Baxter, J. and P. L. Bartlett, 2001, "Infinite-horizon policy-gradient estimation", *Journal of Artificial Intelligence Research*, Vol. 15, pp. 319–350.

Bellman, R., 1957, *Dynamic Programming,* Princeton University Press, Princeton, NJ, USA.

Berry, D. A. and B. Fristedt, 1985, *Bandit Problems: Sequential Allocation of Experiments,* Chapman and Hall.

Bertsekas, D., 1987, *Dynamic Programming: Deterministic and Stochastic Models,* Prentice-Hall, Englewood Cliffs, NJ.

Bonet, B. and H. Geffner, 2000, "Planning with incomplete information as heuristic search in belief space", in *Artificial Intelligence Planning Systems,* pp. 52–61.

Boutilier, C. and D. Poole, 1996, "Computing optimal policies for partially observable decision processes using compact representations", In *Proceedings of AAAI-96*, pp.

1168–1175.

Boutilier C., T. Dean and S. Hanks, 1999, "Decision-theoretic planning: Structural assumptions and computational leverage", *J. of Artificial Intelligence Research*, Vol. 11, pp. 1–94.

Boyce, S. J., 2000, "Natural spoken dialogue systems for telephony applications", *Communications of the ACM*, Vol. 43, pp. 29–34.

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

Bracewell, R., 1999, *Two-Dimensional Convolution,* Vol. 1. McGraw-Hill.

Brafman, R. I., 1997, "A heuristic variable grid solution method for POMDPs", in *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI '97).*

Brooks, A., A. Makarenko, S. Williams and H. Durrant-Whyte, 2005, "Planning in Continuous State Spaces with Parametric POMDPs", In: *IJCAI Workshop Reasoning with Uncertainty in Robotics*, Edinburgh.

Brown, R. G. and P. Y. C. Hwang, 1992, *Introduction to Random Signals and Applied Kalman Filtering*, Second Edition, John Wiley & Sons, Inc.

Carpenter, G. A. and S. Grossberg, 1987a, "A Massively parallel architecture for a self organizing neural pattern recognition machine", *Computer Vision, Graphics, and Image Processing*, Vol. 37, pp. 54–115.

Carpenter, G. A. and S. Grossberg, 1987b, "ART2. Self-Organization of Stable Category Recognition Codes for Analog Input Patterns.", *Applied Optics*, pp. 4919–30.

Carpenter, G. A., S. Grossberg and D. B. Rosen, 1991a, "ART 2-A: An Adaptive Reso-

nance Algorithm for Rapid Category Learning and Recognition", *Neural Networks*, Vol. 4., pp. 493–504.

Carpenter, G. A., S. Grossberg and D. B. Rosen, 1991b, "Fuzzy ART: Fast stable learning and categorization of analog patterns by an adaptive resonance system", *Neural Networks*, Vol. 4, pp. 759–771.

Carpenter, G. A., S. Grossberg and J. H. Reynolds, 1991c, "ARTMAP: Supervised real-time learning and classification of non-stationary data by a self-organizing neural network", *Neural Networks*, Vol. 4, pp. 565–588.

Cassandra, A. R., L. P. Kaelbling and M. L. Littman, 1994, "Acting optimally in partially observable stochastic domains", in *Proceedings of the Twelfth National Conference on Artificial Intelligence,* pp. 1023–1028.

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

Cassandra, A., 1998, *Exact and approximate algorithms for partially observable Markov decision process*, PhD dissertation, Brown University.

Cassandra, A., 1999, *Tony's POMDP File Repository Page*. http://www.cs.brown.edu /research/ai/pomdp/examples/index.html.

Cheng, H. T., 1998, *Algorithms for partially observable Markov decision process*, PhD dissertation, University of British Columbia.

Duda, R. O., P. E. Heart and D. G. Stork, 2001, *Pattern Classification*, Wiley-Interscience.

Engel, Y., and S. Mannor, 2001, "On Finding Good State Aggregation Functions", ICML workshop on Hierarchy and Memory in Reinforcement Learning.

Grewal, A. and C. Andrews, 1993, *Kalman Filtering*. Prentice Hall.

Hansen, E. A. and Z. Feng, 2000, "Dynamic programming for POMDPs using a factored state representation", In *Fifth International Conference on Artificial Intelligence Planning and Scheduling*, pp. 130–139.

Hartigan, J. A., 1975. *Clustering Algorithms*, NY: Wiley.

Hauskrecht, M., 1997, "Incremental methods for computing bounds in partially observable markov decision processes", In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97)*, pp. 734–739.

Hauskrecht, M., 2000, "Value-function approximations for partially observable markov decision processes", *J. of Artificial Intelligence Research*, Vol. 13, pp. 33–94.

Haykin, S., 1999, *Neural Networks, A Comprehensive Foundation*, New Jersey, Prentice-Hall.

Hochreiter, S. and J. Schmidhuber, 1997, "Long Short-Term Memory", Neural Computation, Vol. 9, pp. 1735–1780.

Hoey, J. and P. Poupart, 2005, "Solving pomdps with continuous or large discrete observation spaces", In: *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pp. 1332–1338.

Jacobs, O. L. R., 1993, *Introduction to Control Theory*, 2nd Edition. Oxford University Press.

Kaelbling, L. P., M. L. Littman and A. P. Moore, 1996, "Reinforcement learning: A survey", *Journal of Artificial Intelligence Research*, Vol. 4, pp. 37–285.

Kalman, R. E., 1960, "A New Approach to Linear Filtering and Prediction Problems", *Transactions of the ASME–Journal of Basic Engineering*, Vol. 82, Series D, pp.35–45.

Littman, M., 1994, "Memoryless policies: theoretical limitations and practical results", In *Proceedings of the Conference on Simulation of Adaptive Behavior*, pp. 297–305.

Littman, M. L., A. R. Cassandra and L. P. Kaelbling, 1995, "Learning policies for partially observable environments: scaling up", In *Proceedings of the Twelfth International Conference on Machine Learning*, pp. 362–370.

Long–Ji, L. and T. Mitchell, 1992, *Memory approaches to reinforcement learning in non-markovian domains*, Technical Report CMU-CS-92-138, School of Computer Science, Carnegie Mellon University.

Lovejoy, W. S., 1991, "A survey of algorithmic methods for partially observed Markov decision processes", *Annals of Operations Research*, Vol. 28, pp. 47–65.

Madani, O., S. Hank and A. Condon, 1999, "On the undecidability of probabilistic planning and infinitehorizon partially observable Markov decision problems", In *Proc. of the National Conference on Artificial Intelligence*, pp. 541–548.

MathWorks, 1994, *Example: Modeling an Inverted Pendulum.* http://www.library.cmu.edu/ctms/ctms/examples/pend/invpen.htm.

Maybeck, P. S., 1979, *Stochastic models, estimation and control*, Academic Press.

McCallum, A. R., 1995, "Instance-based util distinctions for reinforcement learning with hidden state", In *Proceedings of the Twelfth International Conference on Machine Learning, San Francisco*, pp. 387–395.

McCallum, A. R., 1996, "Learning to use selective attention and short-term memory in sequential tasks", In *From animals to animats 4: Proceedings of the fourth international conference on simulation of adaptive behavior*, pp. 315–324.

Michel, O., 2004, "Cyberbotics ltd - Webotstm: Professional mobile robot simulation", *International Journal of Advanced Robotic Systems*, Vol. 1, pp. 39–42.

Minsky, M. L., 1967, *Computation: Finite and infinite machines*, Englewood Cliffs, NJ: Prentice-Hall.

Mitchell, T. M., 1997, *Machine Learning*, New York: McGraw-Hill.

Monahan, G. E., 1982, "A survey of partially observable Markov decision process: Theory, models, and algorithms", *Management Science*, Vol. 28, pp. 1–16.

Nourbakhsh, I., R. Powers and S. Birchfield, 1995, "DERVISH an office-navigating robot", *AI Magazine*, Vol. 16, pp. 53–60.

Orr, G., N. Schraudolph and F. Cummins, 1999, CS-449: Neural Networks. http://www.willamette.edu/∼gorr/classes/cs449/intro.html.

Papadimitriou, C. H. and J. N. Tsitsiklis, 1987, "The complexity of markov decision processes", *Mathematics of Operations Research*, Vol. 12, pp. 441–450.

Peshkin, L., N. Meuleau and L. P. Kaelbling, 1999, "Learning policies with external memory", In: *Proceedings of the Sixteenth International Conference on Machine Learning,* pp. 307–314.

Peshkin, L. and G. R. Shelton, 2002, "Learning from scarce experience", *Proceedings of the Nineteenth International Conference on Machine Learning*, pp. 498–505.

Porta, J. M., M. T. J. Spaan and N. Vlassis, 2004, *Value Iteration for Continuous-State POMDPs*, IAS Technical Report, University of Amsterdam, Tech. Rep. IAS–UVA–04–04.

Porta, J. M., M. T. J. Spaan and N. Vlassis, 2005, "Robot planning in partially observable continuous domains", In: *Robotics: Science and Systems I* (G. S. Sukhatme, S. Thrun and S. Schaal, Eds.). MIT Press, pp. 217–224.

Ron, D., J. Singer and N. Tishby, 1994, "The power of amnesia", *Advances in Neural Information Processing Systems*, pp. 176–183.

Roy, N. and S. Thrun, 2001, "Integrating the value functions and policy search for continuous Markov decision processes", In *Proceedings of Neural Information Processing Systems.*

Rumelhart, D., G. Hinton, and R. Williams, 1986, *Parallel distributed processing*, Cambridge, MA.: MIT Press.

Russell S. J. and P. Norvig, 2003, *Artifical Intelligence: A Modern Approach*, 2nd edition, Prentice Hall.

Sallans, B., 2000, "Learning factored representations on partially observable markov decision process", In: *Neural Information Processing Systems.* MIT Press, pp. 1050–1056.

Siegelmann, H. T. and E. D. Sontag, 1991, "Turing Computability with Neural Nets", *Applied Mathematics Letters*, Vol. 4, pp. 77–80.

Simmons, R. and S. Koenig, 1995, "Probabilistic robot navigation in partially observable environments", In *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 1080–1087.

Singh, S., T. Jaakkola, and M. I. Jordan, 1994, "Learning without state estimation in partially observable Markov decision processes", In *Machine Learning: Proceedings of the Eleventh International Conference*, pp. 284–292.

Singh, S. P., T. Jaakkolah and M. I. Jordan, 1995, "Reinforcement learning with soft state aggregation", *Advances in Neural Information Processing Systems*, pp. 361-368.

Smallwood, R. D. and E. J. Sondik, 1973, "The optimal control of partially observable Markov decision processes over a finite horizon", *Operations Research*, Vol. 21, pp. 1071–1088.

Sondik, E. J., 1971, *The optimal control of partially observable Markov decision process*,

PhD dissertation, Stanford University.

Sondik, E. J., 1978, "The optimal control of partially observable markov processes over the infinite horizon: Discounted costs", *Operations Research*, Vol. 26, pp. 282–304.

Sutton, R. S., 1988, "Learning to predict by the methods of temporal differences", *Machine Learning*, Vol. 3, pp. 9–44.

Sutton, R. S. and A. G. Barto, 1998, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA.

Sutton, R. S., S. Singh, S. D. Precup and B. Ravindran, 1999, "Improved switching among temporally abstract actions", In *Proceedings of the 1998 conference on Advances in neural information processing systems II*, pp. 1066–1072.

Szita, I. and A. Lõrincz, 2004, "Kalman filter control embedded into the reinforcement learning framework", *Neural Computation*, Vol. 16, pp. 491–499.

Tesauro, G., 2002, "Programming Backgammon Using Self-Teaching Neural Nets", *Artificial Intelligence*, Vol. 134, pp. 181–199.

Watkins, C., 1989, *Learning from delayed rewards*, PhD. Thesis, King's College, Cambridge.

Williams, R. J. and J. Peng, 1990,"An efficient gradient based algorithm for on-line learning of recurrent network trajectories", *Neural Computation*, Vol. 2, pp. 490–501.

REFERENCES NOT CITED

Aberdeen, D., 2003, *A (revised) survey of approximate methods for solving Partially Observable Markov Decision Processes*, Technical report, National ICT Australia, Canberra Austalia.

Agre P. and D. Chapman, 1987, "Pengi: An implementation of a theory of activity", In*Proceedings of the Sixth National Conference on Artificial Intelligence*, pp. 268–272.

Carpenter, G. A. and S. Grossberg, 1990, "ART 3: Hierarchical search using chemical transmitters in self-organizing pattern recognition architectures", *Neural Networks*, Vol. 3, pp. 129–152.

Murphy, K. P., 2000,"A survey of pomdp solution techniques", Technical Report, Department of Computer Science, U.C. Berkeley.

Ott, N. and H. G. Meder, 1972, "The Kalman filter as a prediction error filter", *Geophys. Prosp.*, Vol. 20, pp. 549–560.

Suematsu, N. and A. Hayashi, 1999, "A reinforcement learning algorithm in partially observable environments using short-term memory", In *Proceedings of Neural Information Processing Systems*, pp. 1059–1065.

Thrun, S., 1992, *Efficient exploration in reinforcement learning*, Technical Report CMU-CS-92-102, School of Computer Science, Carnegie Mellon University.

Thrun, S., 1993, "Exploration and model building in mobile robot domains", In *Proceedings of the IEEE International Conference on Neural Networks*, pp. 175–180.

Thrun, S., M. Bennewitz, W. Burgard, A. B. Cremers, F. Dellaert, D. Fox, D. Hähnel, V. Rosenberg, N. Roy, J. Schulte and D. Schulz, 1999, "MINERVA: A second

generation mobile tour-guide robot", In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1999–2005.

Ullman, S., 1984. "Visual Routines", *Cognition*, Vol. 18, pp. 97—159.