IMPLEMENTATION OF CONTINUOUS POMDP ALGORITHMS ON AUTONOMOUS ROBOTS

by

Derya Sezen

BS, in Computer Engineering, Galatasaray University, 2003

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

Graduate Program in System and Control Engineering

Boğaziçi University

2008

ACKNOWLEDGEMENTS

First of all, I thank my thesis supervisor Prof. Dr. H. Levent Akın for his invaluable support during my thesis study as well as his guidance, patience, and motivation.

I would like to thank to Prof. Dr. Ethem Alpaydın and Doç. Dr. Borahan Tümer, my thesis committee members, for their kind participation.

Thanks to all AILAB people, especially Barış, Çetin, Ergin, Fuat, Nuri, Reyhan and Tekin, for their friendship and their valuable support to this work.

I am so grateful to Fatoş Hasgür for her endless and moral support. She showed a great patience throughout this study.

I am also so grateful to Ümit Karakaş, my manager, for his support and patience to this work.

I have to mention my family's support. Without their support, i couldn't finish this work.

This thesis was supported by Boğaziçi University Research Fund project 06HA102 and TUBITAK project 106E172.

ABSTRACT

IMPLEMENTATION OF CONTINUOUS POMDP ALGORITHMS ON AUTONOMOUS ROBOTS

Uncertainty is a fundamental problem for autonomous agents in a partially observable real world, where the sensors are not able to give the complete state of the environment. Although the outcomes of actions are not predictable, the agents must behave rationally. Furthermore, continuous nature of the environment makes the problem more difficult to model.

Markov Decision Process (MDP) is a way to model this kind of problems. Partially observable Markov decision process (POMDP) is an extension of MDP which can be used in environments which are not fully observable. In order to model the real world, the continuous states must be converted to discrete states.

The aim of this work is to model the real world environment and implement ARKAQ learning algorithm which is suitable for Partially observable Markov decision problems (POMDPs).

The experiments are realized with Sony AIBO four-legged robotic pets under Webots simulation environment. Two problems are studied: “Ball Approaching” and “Scoring Goal”. The predefined targets are achieved by the robots and the results in goal scoring show that ARKAQ is clearly much more successful compared to random actions. The optimality of the results are discussed and the parameters that affect the optimality are explained.

ÖZET

OTONOM ROBOTLAR ÜZERİNDE SÜREKLİ KGMKS ALGORİTMALARININ UYGULANMASI

Belirsizlik, otonom robotlar için, duyargaların gerçek dünya gibi tüm ortam durumunu tam olarak yansıtamadığı hallerde başedilmesi gereken temel problemlerden biridir. Gerçekleştirilen eylemlerin sonuçlarının ne olacağı önceden bilinmese dahi etmenlerin mantıklı davranışlar içinde bulunmaları beklenmektedir. Bunun yanında ortamın aralıksız dağılımlı zaman yapısı problemi daha da zor hale getirmektedir.

Markov Karar Yöntemleri bu tip ortamların modellenmesi için uygundur. Kısmen Gözlemlenebilir Markov Karar Süreçleri ise bütünüyle gözlemlenmesi mümkün olmayan ortamlar için tercih edilmektedir. Bu yöntemlerle ortamın modellenebilmesi için, gerçek dünyanın aralıksız dağılımlı yapısının aralıklı hale getirilmesi gerekmektedir.

Bu çalışmanın amacı gerçek dünyayı modelleyerek, Kısmen Gözlemlenebilir Markov Karar Süreçlerine uygun öğrenme algoritmalarının gerçek robotlar üzerinde uygulanmasıdır.

Deneyler Sony'nin dört bacaklı AIBO köpek robotları üzerinde, Webots simulasyon ortamı kullanılarak gerçekleştirilmiştir. İki farklı problem üzerinde algoritmalar uygulanmıştır: "Topa Yaklaşma" ve "Gol Atma". Robotların önceden verilen hedeflere ulaşmayı başardıkları ve gol skorlarında rastgele verilen kararlara nazaran daha başarılı oldukları gözlemlenmiştir. Sonuca ulaşırken gerçekleştirilen eylemlerin optimum olup olmadığı tartışılmış ve bunu etkileyen parametreler açıklanmıştır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS/ABBREVIATIONS

a	Action
A	Kalman state transition model
b	Belief
B	Kalman control-input model
H	Kalman observation model
I	Indication
o	Observation
O	Set of probability distributions over states
p	Probability
q	Observation of the current state
Q	Kalman process noise or Q-value
R	Kalman measurement noise or Reward
s	State or signal
T	State Transition Function
V	Value Function
α	Learning rate
γ	Learning factor or discount rate
π	Policy
Ω	Set of observations
ρ	Vigilance
AI	Artificial Intelligence
ART	Adaptive Resonance Theory
$CO-MDP$	Completely Observable Markov Decision Processes
DP	Dynamic Programming
LTM	Long Term Memory
MDP	Markov Decision Processes
ML	Match Learning

$POMDP$	Partially Observable Markov Decision Processes
$PWLC$	Piece-wise Linear and Convex
RL	Reinforcement Learning
SL	Supervised Learning
SPD	Stability-Plasticity Dilemma
STM	Short Term Memory
TD	Temporal Difference
V	Value Function
VI	Value Iteration

1. INTRODUCTION

Uncertainty is the main problem for an agent which interacts with the real-world. Unlike the early days in AI, planning under uncertainty and giving accurate decisions has become to play a crucial role, whenever the sensors are inaccurate, effectors are limited, the state is partially observable, the environment is dynamic and noisy. Without having some solutions for these dynamics, the agent cannot survive in a real-world.

A human being can be considered as an autonomous agent. Although the real-world is partially observable and has many uncertainty issues, the human-being can survive mostly by using its past experiences. As an example, for a small baby, a heater in the house is a partially-observable issue because, externally, it seems like a regular object. After touching the heater first time and having a pain, the baby learns this object and will not touch it again by taking into consideration the past experience.

In order to model the environment, we need an approach which, incorporating the past experiences, devises a solution for the uncertainty. Markov Decision Processes (MDP) [1, 2, 3] is a model which has two assumptions related to these dynamics: The first one is that the environment can be defined probabilistically and the second one, which is also referred to as first-order Markov assumption, is that the historical data how the agent has arrived to the current state is not taken into account; in other words, the future is independent of the past knowledge. MDP model implicitly exploits the past experience.

The main problem on implementing the Markov model to the real world problems is that, it assumes the environment to be fully-observable which, however, is not. So we have to consider the world as Partially Observable Markov Decision Processes (POMDP) [4, 5] and construct a model based on this approach.

The sensory environment provides an agent the information about the environment. The agent makes observations by using this information. These observations

helps the agent to infer the state that it is currently belong to. The continuous environment states, must be converted to discrete states in order the agent to decide which state it belongs to and give a decision upon to that. Adaptive Resonance Theory (ART) [6] is a convenient method to segment the continuous real-world environment.

The second problem is that, the data provided by the sensory environment can be incomplete and noisy which will make the ART network to produce undesired results. We'll be implementing Kalman Filter [7, 8] on top of the ART network, which will assure that the data provided to be more accurate.

The agent is expected to give decisions in order to achieve to its goal. These decisions are basicly the actions that the agent is able to perform. The role of the policy generator is to give the decisions by considering the goal and the current state. As a policy generator, we'll be using Q-Learning [9].

1.1. Aim of the Thesis

The aim of this work is to implement the ARKAQ Learning [10] which was proposed as a solution to POMDP problems. ARKAQ Learning model relies on an effective integration between Kalman Filter, ART-2 networks and Q-Learning.

The algorithms are implemented on AIBO [11] robots, coded under C++ on top of Cerberus [12] codebase.

There are two case problems that are studied: The first one is "Ball Approaching" in which AIBO tries to learn how to approach to the ball, the second is "Goal Achieving" in which AIBO tries to learn how to score a goal.

1.2. Thesis Outline

In chapter 2, we'll be describing the background elements of the thesis in details, which are Reinforcement Learning, Q-Learning, Markov Decision Processes (MDPs),

Partially Observable Markov Decision Processes (POMDPs), Adaptive Resonance Theory (ART) and Kalman Filter.

In chapter 3, we'll be defining the ARKAQ Learning model which relies on the elements that are described in chapter 2.

In chapter 4, we'll define the details of implementation of the algorithms, especially defining the different types of implementation in state segmentation.

In chapter 5, we'll be examining the two problems with different dynamics and we'll be giving the results that are taken.

2. BACKGROUND

2.1. Reinforcement Learning

Reinforcement Learning (RL) [13, 14, 15] is an unsupervised machine learning technique in which an agent tries to find the best actions with a trial-error process by considering the predefined environmental rewards.

RL tries to maximize the reward received. As the agent is not told which actions to take, it tries to find a mapping from states to actions to this effect. The actions taken may affect the next state rewards and all subsequent rewards as well. Trial-error period and delayed-reward are two fundamental features of reinforcement learning.

The main requisite of RL is a well characterized learning problem. Basically, the agent should sense the state to an extent, related to the problem, from the environment in order to achieve to the goal. Sensation, action and goal are the three aspects of RL.

In supervised learning (SL) the correct input and output pairs are given by a supervisor, which is the factor that differentiates it from unsupervised learning. SL is not an adequate method for learning from interaction because it is impractical and sometimes impossible to provide all possible situations and the desired behaviours. Especially, in an uncharted territory the agent must be able to learn from experience.

The trade-off between exploration and exploitation is a challenging problem in RL. The agent should try actions it has not selected before, which is called *exploration*, in order to make better action selections in the future. Beside that, the agent can consider its past experiences by performing the actions that it can obtain more reward which is called *exploitation*.

RL tries to achieve predefined goals in an uncertain environment. Even if the agent has an explicit goal, has sensations and decides which action to perform, it has

to deal with that uncertainty. In some cases RL involves planning and gives real-time decisions meanwhile.

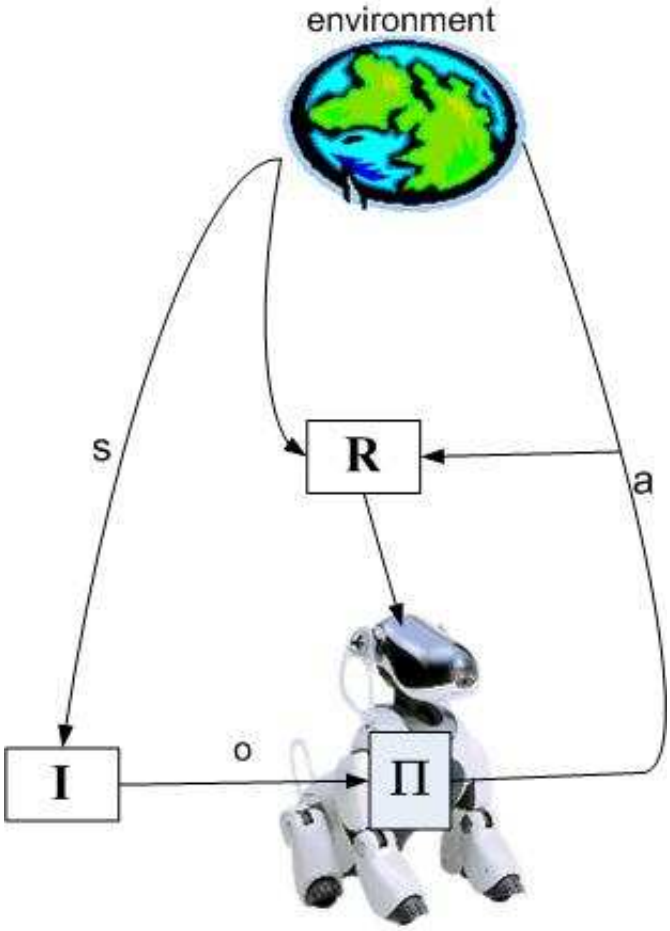

Figure 2.1. Reinforcement Learning, Agent World

Figure 2.1 describes a typical Reinforcement Learning agent world. The agent interacts with the environment through its sensors s and actions a. By considering the indication I, the agent makes an observation o and an appropriate action is choosen by the policy π. The same sensory inputs also provides information whether a reward or punishment is received from the environment by R.

2.1.1. Optimal Policy

Before implementing the policy, we must decide the optimality model which defines how the future will be taken into account before giving a decision. There are

three models for finding the optimal policy in Reinforcement Learning:

1. Finite-Horizon Model: The agent tries to find an optimal policy for the next k steps. Expected reward for the next k steps is defined by:

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

where r(t) is the scalar reward received in t steps.

This model can be implemented by giving a decision considering k next steps and in next state considering k-1 steps until 1 step remains or giving decisions always considering k next steps.

2. Discounted Infinite-Horizon: The agent tries to find the long-term reward. The future rewards are geometrically discounted according to the discount factor γ.

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

γ prevents the sum to be infinite.

3. Average Reward Model: The agent tries to optimize its long-term reward

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

Figure 2.2 is an example indicating these three optimal policies. In the figure, the circles are states and arrows are state transitions. Initial state is the one on the upper left. The labeled arrows produce the indicated rewards and unlabeled arrows produce zero reward. If the horizon length is 5, for finite-horizon model, the related rewards will be +6, 0, 0 so the upper first action should be the choice.

If the horizon is infinite, related rewards will be +16.2, +59, +58.5 so the infinite horizon model must be preferred.

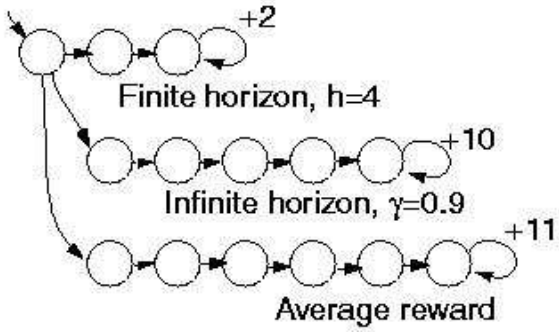

Figure 2.2. 3 models of Optimal Policy [14]

The finite-horizon model is suitable in the case where the agent's lifetime is known. The other two models are dependent on the horizon length but the average reward model does not need a discount factor which differentiates it from the infinite horizon model.

2.1.2. Measuring Learning Performance

The quality of learning can be evaluated by some measures:

1. Convergence to optimal behaviour: This measures the reliability of the algorithm in terms of eventual optimality. An agent that can reach up to 99 percent optimality in short time can be preferable to the agent that reaches to optimal behaviour in long time.

2. Speed of convergence: In order to measure the speed, the definition of the convergency must be well defined. An algorithm that tries to reach the optimality quickly may encounter penalties during the learning period. Generally, more time consuming but more guaranteed method is the choice.

3. Regret: Instead of applying the optimal policy initially, the punishments from the environment can be evaluated during the run. The expected decrease in reward

is called regret.

2.1.3. Exploration versus Exploitation

In Reinforcement Learning, the agent must explore the environment which is the major difference from supervised learning. By selecting the least selected actions, it can explore the environment.

N-armed bandit problem [16] is an example of a simple reinforcement learning case. There are k gambling machines and the agent is permitted h times to pull an arm and any arm may be pulled each time. The machine pays off one or zero after an arm i is pulled which is a probability p_i and each machine has its own probability. This problem shows the trade-off between exploration and exploitation. Exploration means trying different machines, but its risky and may have uncertain pay offs.

Briefly, exploration speeds up convergence but risks to encounter uncertain rewards. On the other hand, exploitation gets better rewards but risks to a late convergency to an optimal policy. There is not a definite answer to this problem, the aim should be making a balance between two.

2.1.4. Goals and Rewards

In reinforcement learning, the goal can be a state or a special reward signal to the agent from the environment. The agent aims to maximize long term rewards instead of maximizing immediate rewards.

For example, in a maze environment where an agent tries to get to a goal location, the reward is zero for every state different from the goal state and one for the goal state. Another approach gives -1 reward in every state different from the goal state and gives one for goal state.

Rewards in reinforcement learning are computed in the environment. If we take

an animal's behaviour into account, the goals are computed inside their bodies. In order to implement this to reinforcement learning, the related body sensors like food, hunger, pain and pleasure may be considered as enviromental signals. The main idea placing the goal signals outside the agent is that, the agent should not have any control over these signals.

2.2. Markov Decision Processes (MDP)

In Reinforcement Learning, the environment must rely on a model which provides the agent an accurate infrastructure in order to behave rationally. This infrastructure should contain environmental signals as states and other environmental properties. The state of the agent can be inferred from any signal it can receive from the environment. Particularly, the state signal must contain immediate sensations and all possible signals from the environment, but the state signal cannot include everything about the environment. For example, if the agent is waiting for a taxi, it cannot know whether a free taxi is approaching or not. As an another example, if the agent is answering to a analog-line phone, without a caller-id box, we should not expect it to know the caller. Eventually, there are hidden environmental signals that cannot be received but the agent is expected to utilize all possible non-hidden environmental signals. The Markov state is the signal that retains all the relevant information about the environment and should contain a summary of past sensations. As an example, the current position of the pieces on a checkers board is a Markov state because it has a summary of all past movements but not all the sequences one by one.

Markov Decision Processes (MDP) [1, 2, 3] is a discrete time non-deterministic process, which is characterized by a set of states and possible actions. The agent needs to make decisions (actions) in order to make transitions within the states. The transition function defines the transition probability to the next state. For each state, the agent receives reward from the environment. Briefly, MDP is defined as a tuple $\langle S, A, T, R \rangle$ where,

- S is the set of environment states. State can be defined as a vector that consists of

the environmental features. This is used to learn about the changing environment.

- A is the set of actions. For every state, a set of possible actions exists in order to make a transition to another state.
- $\Re$ is the reward function S x A $\rightarrow$ R . The measure how the action resulted is called reward. It is used for comparing the results of different actions.
- T is the state transition function SxA $\rightarrow \Pi(S)$ which is a probability distribution over states. An action may have different effects depending on the state that it is performed on. For each state, we have to define the results of all possible actions. Since MDP is a stochastic process, a result of an action may be probabilistic. That's why this function is defined as a probability distribution. In case where the actions are deterministic, T would be SxA $\rightarrow$ (S)

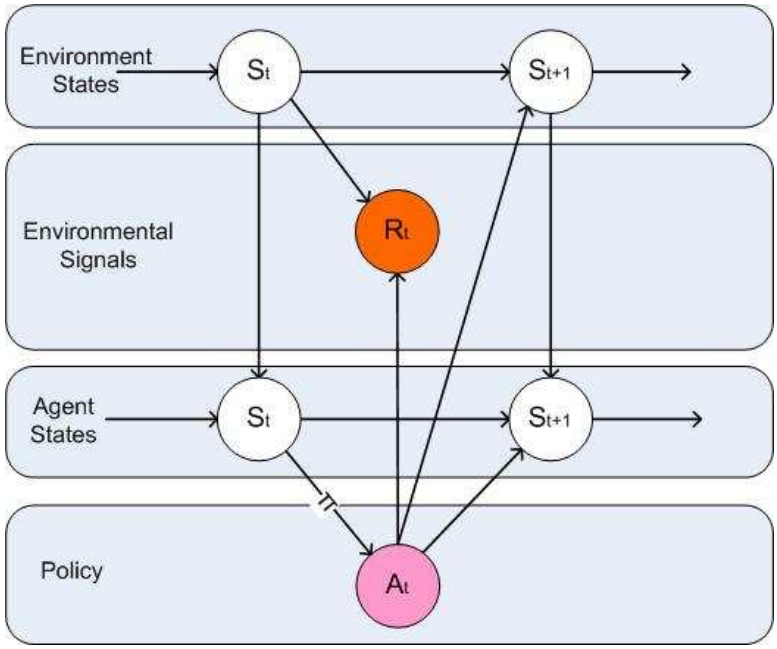

Figure 2.3. MDP Representation

Figure 2.3 is a graphical representation of a MDP. Since the environment is fully-observable, the states are given as input signals, therefore the agent states are the same with the environment states. R_t is immediate reward. After taking action A_t, the environment state becomes S_{t+1}.

Figure 2.4 illustrates the probabilistic state transition model in MDP. The values on the arrows represent the probability of the related transition. The sum of the

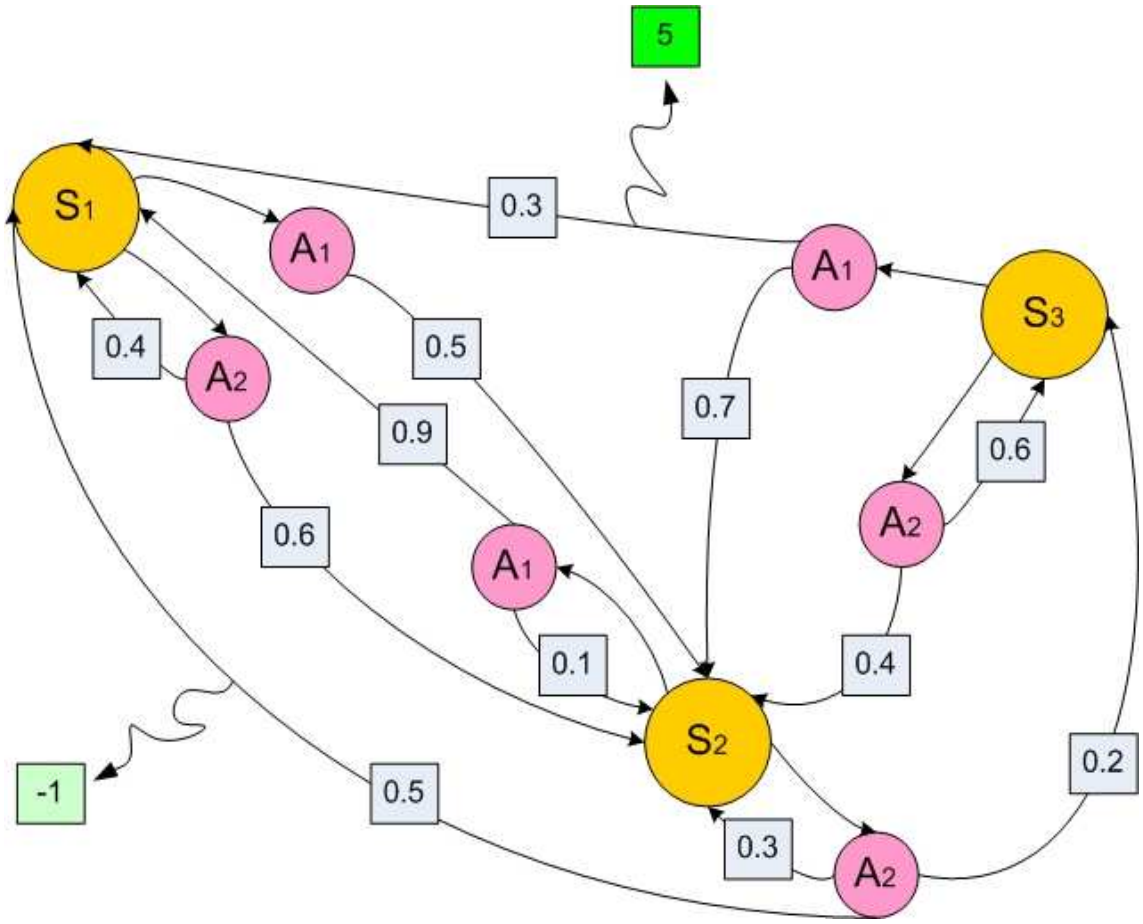

Figure 2.4. MDP State Transitions

probability of a transition, for instance performing action A_2 on state S_2, is equal to 1. The spiral arrows represent rewards or punishments.

In some cases, the resulting state of an action depends on the last n states. This is called n-th order Markov assumption. In general, n is accepted as 1 which is called first-order Markov assumption. As a result, the agent chooses an action based only upon the current state:

$$P(q_n|q_{n-1}, q_{n-2}, ..., q_1) = P(q_n|q_{n-1}) \tag{2.4}$$

where q is the observation of the current state.

Briefly, in MDP the goal is to maximize the cumulative discounted reward:

$$\sum_{t=0}^{\infty} \gamma^t R(s_t) \tag{2.5}$$

where γ is the discount rate and $0 < \gamma < 1$

$\prod$ is defined as policy, which is a mapping from S to A, giving which decisions(actions) to be taken:

$$\prod_{1}(x) = \underset{a}{\operatorname{argmax}}\, R(s,a), \prod : S \rightarrow A \tag{2.6}$$

MDP assumes the environment to be fully observable, which means the resulting state by performing an action will be definitely known by the agent.

2.2.1. Value Iteration Algorithm

MDP is defined as a tuple $\langle S, A, T, R \rangle$ as described in the previous section. The goal is to figure out the best actions for each state for the given horizon length.

The Value Iteration Algorithm begins with finding the values for horizon length 1. Since we have the immediate rewards, the action with the highest immediate reward is chosen for the related state. For the horizon length 2, the value of the horizon length 1 is added to value of the next action to be chosen. This process continues until we reach the desired horizon length.

Value Function (V) is the discounted sum of rewards of the related policy. V value for horizon length one is:

$$V_1(x) = \gamma max R(s,a) \tag{2.7}$$

Dynamic Programming (DP) [17] allows us to determine the Value Function for a given horizon length. The following equation is called as Bellman equation [17]:

$$V(s) = max\left(R(s,a) + \gamma \sum T(s,a,s')V(s')\right) \tag{2.8}$$

By iterating the procedure, we obtain V^* which converges to V(s). Given V^*, the optimal policy is defined as:

$$\overset{*}{\prod}(s) = argmax\left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V^*(s')\right) \tag{2.9}$$

2.2.2. Temporal Difference

Temporal difference (TD) learning [13] is another method for solving the MDP problems, which is a combination of Monte Carlo method [13] and DP. Value function is updated by using the following equation:

$$V(s_t) = V(s_t) + \alpha\left[R_t - V(s_t)\right] \tag{2.10}$$

where R_t is the reward for time horizon t and α is the step-size parameter

Simplest TD method is known as TD(0), which uses one step backup:

$$V(s_t) = V(s_t) + \alpha\left[r_{t+1} + \gamma R_t - V(s_t)\right] \tag{2.11}$$

where R_t is replaced with $r_{t+1} + \gamma R_t$, which indicates reward for one step.

2.2.3. Q-Learning

Reinforcement learning [18] is a learning technique in which the learner is an agent in an environment that receives reward or penalty in response to the actions the agent takes. After realizing a trial period, the best policy is expected to be learned. In this technique the goal is defined but the actions to the goal are not defined. In other words, what to do is defined but how to do is not. That differs reinforcement learning from supervised learning.

Q-Learning [9] is a reinforcement learning model, in which Q value of state and action pairs are calculated using formula:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma max Q(s_{t+1}, a) - Q(s_t, a_t) \right] \tag{2.12}$$

where s is the state, a is the action, r is the reward, α is the learning rate between 0 and 1, γ is the discount rate between 0 and 1 and Q(s,a) is the expected return of taking action a in state s.

The Q-Learning algorithm is guaranteed to converge to the correct Q-values if the environment is stationary, in other words do not change over time or position. The algorithm repeats itself after reaching to the goal state until Q-values converges up to some extend. This can be decided by calculating the delta changes in Q-values.

2.3. Partially Observable Markov Decision Processes

In the MDP model the current state is a signal from the environment, so it can be considered as Completely Observable-MDP (CO-MDP). In the case where the state cannot be obtained with complete reliability by the agent, the environment becomes partially-observable. Partially Observable Markov Decision Processes (POMDP) [4, 5] differentiates from MDP by its partially observable approach.

For example, chess is a completely-observable game because all positions of the

pieces are completely given by the environment. On the other hand, poker is a partially-observable game because the opponents cards cannot be directly observed but can be estimated.

Another example for POMDP is patient management [19]. The patient usually has several complaints but only a few of them are related to the illness. The doctor, as the decision-maker, has multiple possible actions: wait and see, demand additional medical analysis or surgery. Each decision has some risks as the situation of the illness can get worse. However, beginning a wrong treatment may lead to an unrecoverable situation. The result of the treatment is non-deterministic, which means we may encounter unexpected results. The decision-maker must carefully evaluate the cost and benefits of each action. Patient management is a typical problem with its cost-benefit trade-offs, non-deterministic results and partially-observable structure.

In POMDP, as the current state is not clear, we add obervations to the MDP model. The observations give us an idea about the current state and is a probability function over states.

Furthermore, in POMDP we need to keep track of the historical observations and actions. This is achieved by maintaining probability distributions over all of the states. Therefore, after each action this distribution must be updated. Briefly, in MDP our goal is to find a mapping from states to actions whereas in POMDP our goal is to find a mapping from probability distributions over states to actions.

POMDP is defined as a tuple $\langle S, A, T, \Re, \Omega, O \rangle$:

- S is the set of environment states.
- A is the set of actions.
- $\Re$ is the reward function S x A $\rightarrow$ R
- T is the state transition function SxA $\rightarrow \Pi(S)$
- Ω is a set of observations that agent can infer from environmental signals
- O is the set of probability distributions over states: O x S x A $\rightarrow$ *[0, 1]*

The goal of the agent is to learn the policy $\Pi(S)$ which is a mapping from observation history $H_t = ((O_1, A_1, R_1), ..., (O_{t-1}, A_{t-1}, R_{t-1}), (O_t, A_t, R_t))$ to actions A_t.

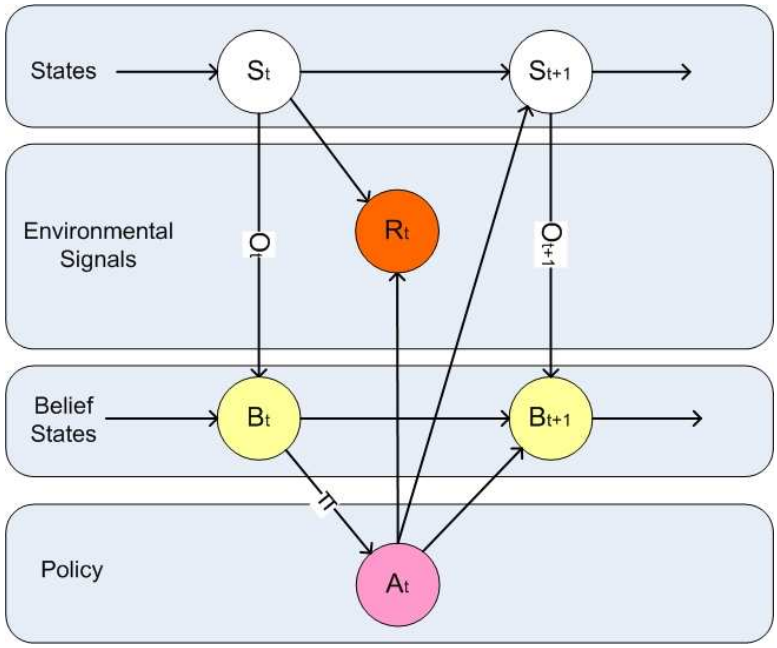

Figure 2.5. POMDP Representation

Figure 2.5 is a graphical representation of a POMDP. S_t is the environmental state, the agent gets an observation O_t from the environment, B_t is the belief state, defined in the next section, that the agent can infer, A_t is the action given by policy function, R_t is the reward. After taking action A_t, the environment state becomes S_{t+1}. The agent gets observation O_{t+1} and reward R_t. The belief state is updated to B_{t+1}.

2.3.1. Belief State

Belief State b, which was proposed by Aström [20], is the probability distribution over states S. The agent tries to infer the belief state by making observations and saves a finite history of past observations. In general, this may not be sufficient because in case the agent gets lost or confused, it should generate a stategy to survive like asking someone or using a map. These actions are not explicitly defined in the POMDP framework.

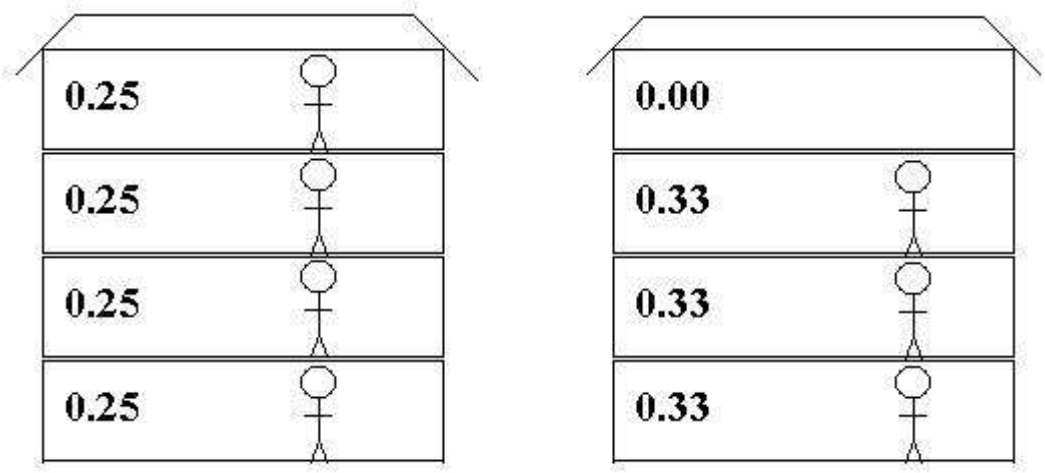

Figure 2.6. Belief State Example

Figure 2.7. Belief State Example

Figure 2.6 is a simple example [10] that shows how belief state values are computed. There are four floors in the building and two possible actions: move down, move up. If the agent takes the move-up action on the fourth floor or move-down act on the first floor, it remains on the same floor. Initially, the agent does not know on which floor it is, so its belief state is [0.25 0.25 0.25 0.25] as described on the left. After the agent performs a move down act and belief state becomes [0.00 0.33 0.33 0.33] because the agent knows that it cannot be on the fourth floor.

Figure 2.7 is a more complicated example [4]. It is more complicated because we add probability to our actions. There are four states and the state with asteriks is the goal state. There are two possible observations: non goal-state when the agent is in states 1,2,4 and goal state when the agent is in state 3. The possible actions are Left and Right. The actions succeed with probability 0.9 and fail with probability 0.1. Action failing means opposite direction action is performed. If no movement is possible to the direction, the agent remains in the same location.

The agent starts with a location other than the goal state. Thus, its belief state is [0.333 0.333 0.000 0.333]. Later, if the agent moves to the right, the belief state becomes [0.100 0.450 0.000 0.450]. Later if it moves again to right and still does not reach to the goal, the belief state becomes [0.100 0.164 0.000 0.736]. Until the agent reaches to the goal state, there will be the always non-zero beliefs.

As another example, a two state POMDP is presented in Figure 2.8. There are two states possible, since the sum of all probabilities is 1, if the probability of being in state s_1 is p, then the probability of being in the other state s_2 is 1-p. This belief state shows the probability distribution of being in state s_1. So, closer to left side means low probability of being in state s_1 but high probability of being in state s_2 and closer to right side means high probability of being in state s_2 but low probability of being in state s_1.

Figure 2.8. Two state POMDP [21]

2.3.2. State Estimator

Belief State b is a probability distribution over states S. $b(s)$ is the probability of being in s with belief b, which is a probability function and $0 \leq b(s) \leq 1$ where $s \in S$ and $\sum_{s \in S} b(s) = 1$. The state estimator computes the new state by using b_{t-1}, a_{t-1} and o_t. The new state b_t can be calculated with the following probability equation:

$$b_t(s_t) = Pr(s_t|o_{t-1}, a_{t-1}, b_{t-1}) \tag{2.13}$$

by using Bayes Rule [22] :

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2.14}$$

the equation of $b_t(s_t)$ becomes:

$$\begin{aligned} b_t(s_t) &= \frac{Pr(o_{t-1}|s_t, a_{t-1}, b_{t-1})Pr(s_t|a_{t-1}, b_{t-1})}{Pr(o_{t-1}|a_{t-1}, b_{t-1})} \\ &= \frac{Pr(o_{t-1}|s_t, a_{t-1})\sum_{s\in S} b(s)Pr(s_t|a_{t-1}, b_{t-1}, s_{t-1})Pr(s_{t-1}|a_{t-1}, b_{t-1})}{Pr(o_{t-1}|a_{t-1}, b_{t-1})} \end{aligned} \tag{2.15}$$

where $Pr(o_{t-1}|a_{t-1}, b_{t-1})$ is the normalizing factor

2.3.3. Optimal Policy

Policy function is a mapping from observation history to actions. The optimal policy is defined as follows:

- B : set of belief states
- A : set of actions
- T(b, a, b_t) : state transition function

$$\begin{aligned} T(b, a, b_t) &= Pr(b_t|a_{t-1}, b_{t-1}) \\ &= \sum_{o\in\Omega} Pr(b_t|a_{t-1}, b_{t-1}, o_{t-1})Pr(o_{t-1}|a_{t-1}, b_{t-1}) \end{aligned} \tag{2.16}$$

where

$$Pr(b_t|a_{t-1}, b_{t-1}, o_{t-1}) = \begin{cases} 1 & \text{if SE(b, a, o)} = b_t \\ 0 & \text{otherwise} \end{cases}$$

where SE(b,a,o) is the state estimation function which has new belief state as its output.

ρ(b,a) is the reward function on belief states:

$$\rho(b,a) = \sum_{s \in S} b(s)R(s,a) \tag{2.17}$$

Belief state represents probabilities for all states, so ρ represents the true expected reward.

2.3.4. Value Function

In MDP, the value function can be directly used to find the optimal policy. Value Function over belief state may be calculated using a value iteration algorithm.

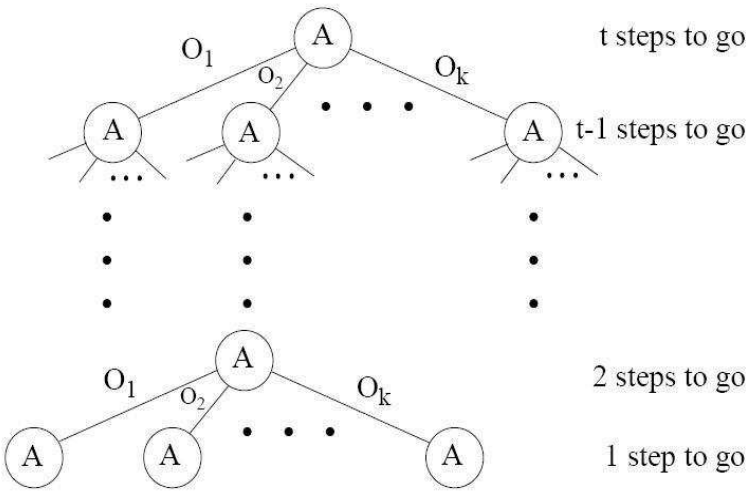

Figure 2.9. Policy Tree for POMDP [4]

The agent in a POMDP environment simply takes an action and makes an observation, maybe depending on previous observations, and goes on like that. Figure 2.9 represents a t-step policy tree of an agent. Initially, the agent performs the action on the top and depending to the observations $O_1, O_2 ... O_k$, it performs the next action.

If p is a 1-step policy tree, the value of performing the action a is:

$$V_p(s) = R(s, a(p)) \tag{2.18}$$

where a(p) is the action performed on the top of the policy tree. In the case, where p is a t-step policy, the value function becomes:

$$\begin{aligned} V_p(s) &= R(s, a(p)) + \gamma \langle \textit{value of the rest} \rangle \\ &= R(s, a(p)) + \gamma \sum_{s_t \in S} Pr(s_t | s_{t-1}, a_{t-1}(p)) \sum_{o_i \in \Omega} Pr(o_i | s_t, a_{t-1}(p)) V_{o_i(p)}(s_t) \\ &= R(s, a(p)) + \gamma \sum_{s_t \in S} T(s_{t-1}, a_{t-1}(p), s_t) \sum_{o_i \in \Omega} O(s_t, a_{t-1}(p), o_i) V_{o_i(p)}(s_t) \end{aligned} \quad (2.19)$$

where $o_i(p)$ is the (t-1)-step policy tree related with observation o_i. All possible next states are considered in order to compute the expected value.

The value of executing a policy tree p from a belief state s is:

$$V_p(s) = \sum_{s \in S} b(s) V_p(s) \quad (2.20)$$

$$\alpha_p = \langle V_p(s_1), ..., V_p(s_n) \rangle \quad (2.21)$$

$$V_p(s) = b.\alpha_p \quad (2.22)$$

This is the value of performing a policy tree p for every belief state s. In order to find the optimal t-step policy, policy trees with different initial belief states must be executed and the one with the maximum value is the optimal t-step policy:

$$V_t(s) = max_{p \in P} b.\alpha_p \quad (2.23)$$

where P is the set of t-step policy trees.

Each policy tree is linear over belief states and the optimal policy is the upper parts of these policy trees, indicating the maximum values. Thus, optimal policy V_t is

piece-wise linear and convex (PWLC). Figure 2.10 represents an optimal t-step PWLC value function. The world has two states, so the belief state vector is two dimensional: $\langle b(s_1), b(s_2)\rangle$. As the belief state is a probability function, the sum is 1, and a single value can represent a belief state. If the belief state is represented by x, this means the belief state vector is $\langle x, 1-x\rangle$. The figure consists of three policy trees: p_1, p_2 and p_3. the value function of p_1 is V_{p_1} is linear as shown in the figure. The bold line represents the upper part of 3 linear value functions, which is the optimal policy.

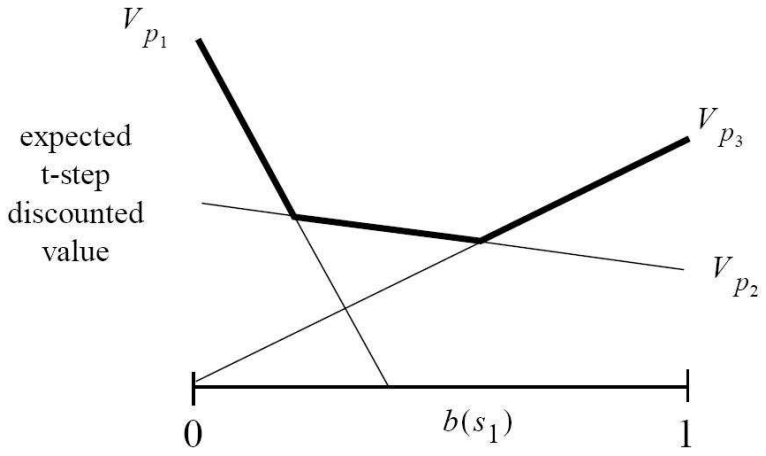

Figure 2.10. Optimal t-step Value Function [4]

Figure 2.11 represents the value function of a world with 3 states. Similar to the reason in the two-state world, the belief state can be determined with two values. Belief state is represented as a triangle with vertices (0,0), (1,0), (0,1). s_1 and s_2 are the dimensions. We can calculate the belief state vector by using the probability property: $s_1 + s_2 + s_3 = 1$. So supplementary value for any s_1 and s_2 in the triangle will be the value of s_3. The value function of a policy tree is a plane in three dimensional world. In the figure, seven policy tree planes are present. Similar to the two dimensional representation, the optimal policy is the upper part of the planes, which is a "bowl" like shape in the figure.

The states that are in the middle of the belief state, which have lower values, are states with higher uncertainty. In these states agent long-term reward is less. On the

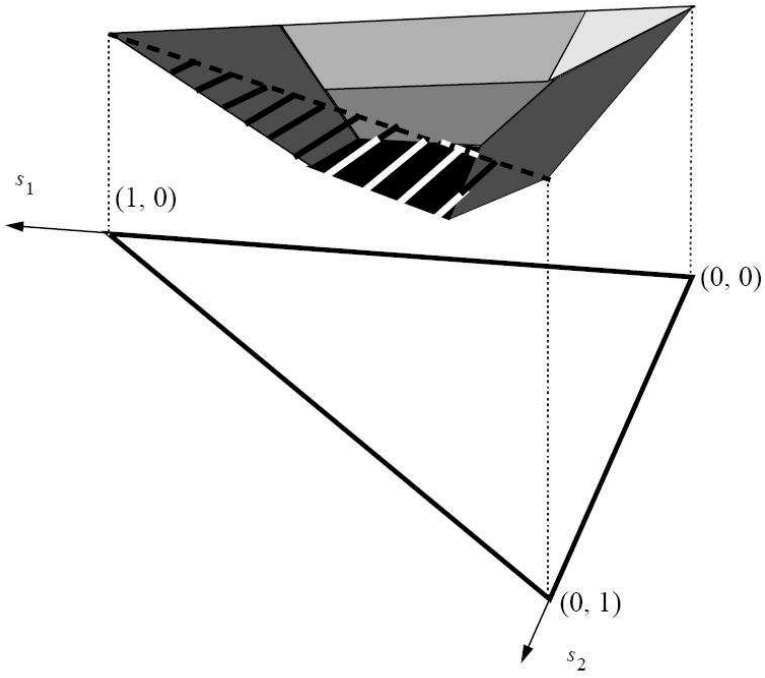

Figure 2.11. Optimal three dimensional t-step Value Function [4]

other hand, the states that are near to the corners are the ones that agent can have higher long-term reward.

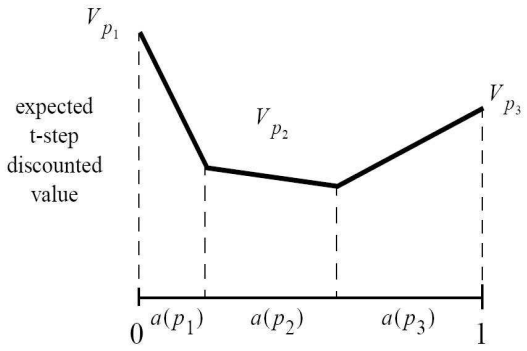

Figure 2.12. Optimal t-step action mapping [4]

As a result, the optimal value function is the projection of the upper lines on the belief space. Figure 2.12 represents the projections of three upper lines on the belief space of the two dimensional example in Figure 2.10. For each internal, the policy tree is different: p_1, p_2 and p_3. For each region, there exist a policy tree such that $V_p(s)$ is maximal over the entire region. Furthermore, *a(p)* on the horizontal axis means the

action on the root node of policy p.

In Figure 2.13 the black dots show the possible resulting belief states after taking actions a1 and a2. The bigger gray dot is the starting belief state and z1, z2, z3 are the observations.

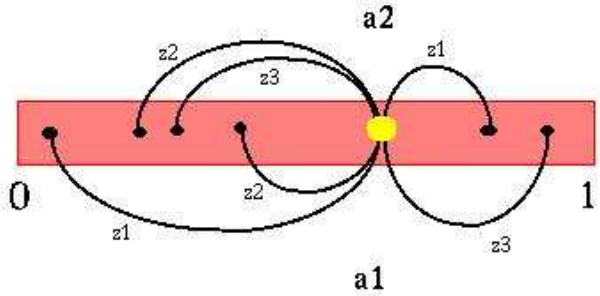

Figure 2.13. Two state POMDP, Belief State update [21]

Similar to the Value Iteration(VI) algorithm in MDP, this algorithm solves MDP problems for a given horizon length. For the POMDP case, the same algorithm can be adapted, the difference is that the space is now continuous. VI function can be represented like in Figure 2.14, where b is the belief space and V(b) is the value function over belief state.

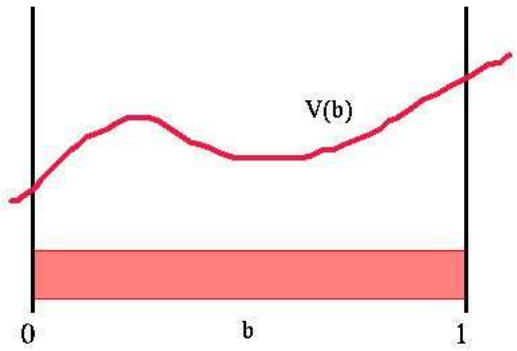

Figure 2.14. Value Function over Belief Space [21]

As discussed before, finite horizon value function is PWLC which means the value function is formed of a finite number of linear segments. Figure 2.15 shows a sample of POMDP Value Function with finite number of linear segments.

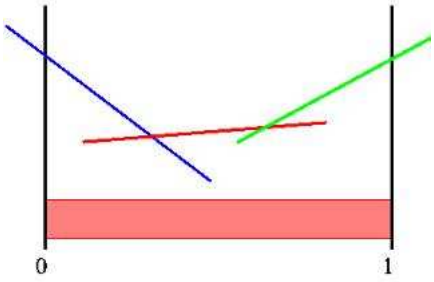

Figure 2.15. PWLC Value Function [21]

In order to find the value of the belief state, if we represent the value fuction as a set of vectors, we find the vector that has the highest dot product with the belief state.

2.3.5. POMDP Solution Example

We'll try to solve a POMDP for a horizon length of three, two states, two actions and three observations. As previously declared, we start with horizon length one, in which we need to take a single action.

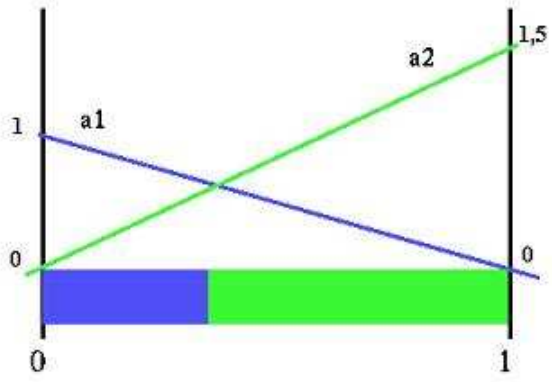

Figure 2.16. POMDP Value Iteration Example [21]

In Figure 2.16, a1 has a value of 0 in state s2 and 1 in state s1 and a2 has a value of 0 in state s1 and 1,5 in state s2. If the belief state is [0.25 0.75], V(a1) = 0.75 * 0 + 0.25 * 1 = 0.25 and V(a2) = 0.75 * 1.5 + 0.25 * 0 = 1.125. The left region on belief state represents the belief states where action a1 must be preferred and the region on the right consists the belief states where action a2 must be preferred.

The next step is to construct the horizon two value function. The value of horizon two is the value of immediate reward plus the value of the next action.

2.3.6. Policy tree domination

Each policy tree represents an optimal strategy in the belief space. This does not mean that every policy tree takes part in the optimal policy. Some policy trees may be dominated by one or more other policy trees. This type of policy trees does not contribute to the optimal value function. For example, in Figure 2.17 policy p_d is completely dominated by policy p_c. Similarly, policy p_c is dominated by p_a and p_b.

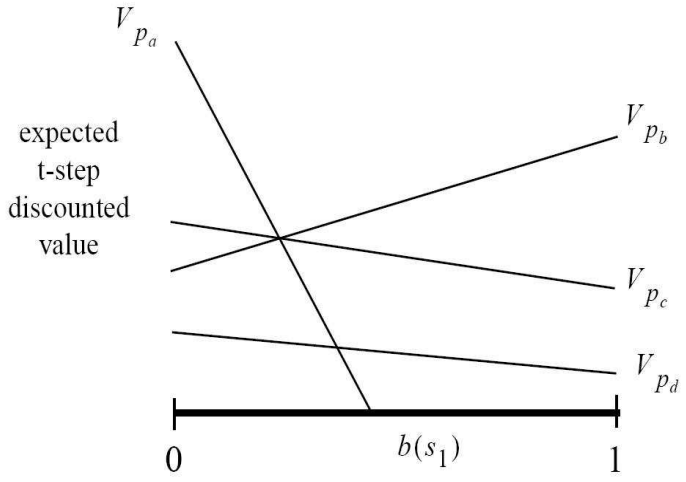

Figure 2.17. Policy tree domination [4]

2.4. Adaptive Resonance Theory

Adaptive Resonance Theory (ART) [23, 24] is a cognitive theory that proposes how the learning and recognition abilities of brain, related with categorization of objects and events, realize in sensory and cognitive neocortex. These abilities require a continuous adaptation to the changing world, thus includes a learning regarding to the expectations from already learned information to match with new input data.

Throughout our life, we experience the world continuously. The environmental signals are somehow processed to be meaningful patterns, serving for developing our

experience. By making use of our experience, we humans are able to develop a sense. This process is realized in a temporary memory called short term memory (STM). The brain makes a matching between previously stored experiences, which are present in long term memory (LTM), and the patterns in STM. The rapid learning capability of brain, without forgetting past experiences, is sometimes referred to as "stability-plasticity dilemma" (SPD). ART proposes that the resonant states, which means data worthy of learning, can provide rapid learning and this can solve SPD. Briefly ART has the self-stabilization property and prevents getting stuck into local-minima.

ART makes use of "match learning" (ML) within the cognitive and sensory domain. ML puts a novel pattern, which is in the sensory domain, into an existing recognition category if enough match exists between bottom-up data and top-down expectations; if not, a new recognition category is learned. The novel pattern is included in the new learned category. Nevertheless, the learned category may match more in existing category information, thus may include some existing information as well. These categories are often referred as "cluster templates" ; in other words "set of patterns".

Decision of whether a new cluster template is required for a new pattern or an existing one can be used is given depending on whether enough number of features in the pattern matches with any existing template or not. This is decided considering the vigilance value ρ [25]. If the number of features that match between a pattern and a template is l and the total number of features is n, the fraction $d{=}l/n$ gives us a value how that pattern is close to the template. As $0 \leq |d| \leq 1$, more d is closer to one, the pattern is closer to the template; more d is closer to zero, the pattern do not share so much common features with the template. If the value of $d > \rho$, a predefined vigilance parameter, that pattern can be assigned to the related template. If ρ is small, then a few features are enough for a match between the new coming patterns and existing template, which means small number of cluster templates will be enough. On the other hand, if ρ is large, then more common features are required in order the new coming pattern matches to an existing template, so higher number of cluster templates will be required.

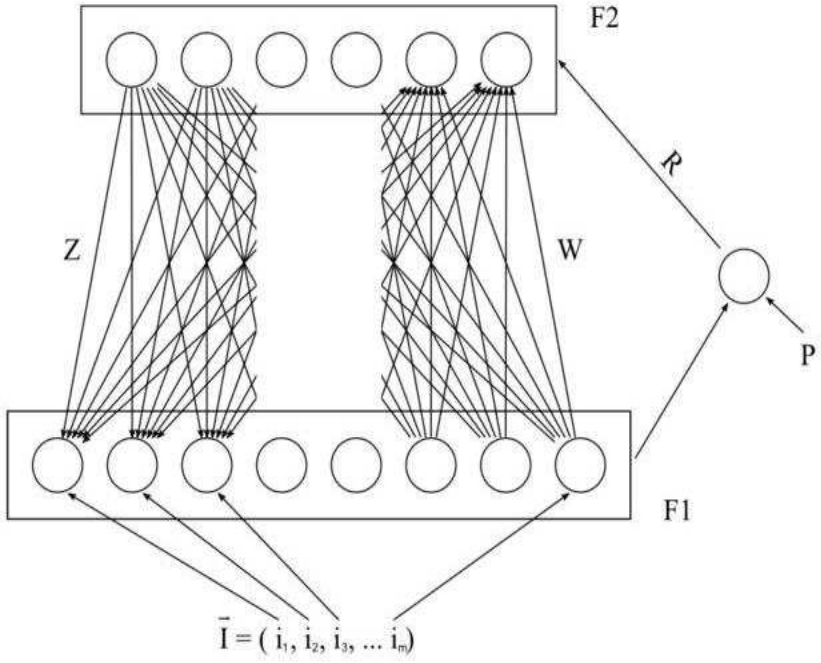

Figure 2.18. ART Neural Network Structure

ART network is basically formed of bottom-up and top-down adaptive filters containing of two layers: Feature representation (F1) and category representation (F2). There are two main forms of ART network: ART1 and ART2. The simplest form of ART network is ART1, which is a two-layer neural network (Fig. 2.18) that takes binary inputs. ART2 is more complicated, extending network capabilities to support analog (continuous) input patterns.

2.4.1. ART1 Structure

Figure 2.19 illustrates the structure of an ART1 network. Attentional subsystem contains F1 and F2 layers which encodes patterns of activation in short-term memory(STM). Bottom-up and top-down traces between F1 and F2 passes through an adaptive LTM. Gain 1 provides a modulation that provides F1 to distiguish or matching between bottom-up input and top-down template. Bottom-up and top-down patterns are matched according to a 2/3 rule which is necessary for self-stabilization in response to arbitrary input patterns.

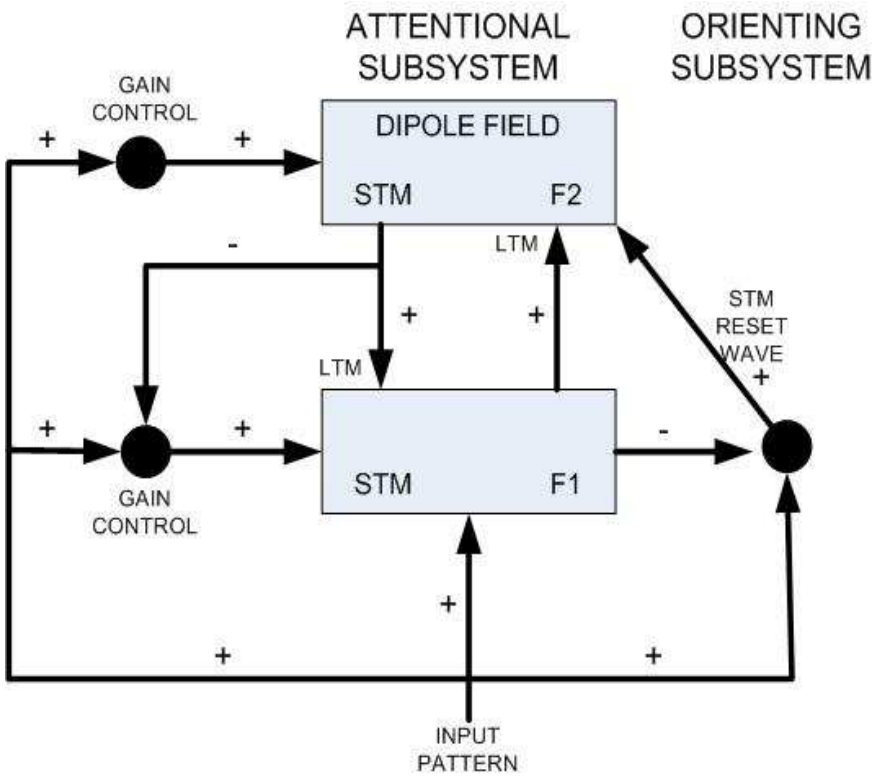

Figure 2.19. ART1 Structure

When a mismatch between bottom-up and top-down pattern happens at F1, the orienting subsystem generates a reset to F2. The processing at F1 terminates and offset of gain 2 is triggered which prepares F2 to encode next input by causing decay of STM at F2.

ART1 can be defined as a system of differential equations that determines STM and LTM in response to arbitrary input patterns. Beside that, ART1 designates a class of architectures rather than a single model.

2.4.2. ART2 Structure

ART2 [26] is designed to accept analog input patterns in addition to binary input patterns. Figure 2.20 illustrates the structure of an ART2 network in which open arrows are specific pattern inputs and filled arrows are nonspecific gain control inputs. In that architecture, feature presentation layer F1 has several processing levels and gain control systems. The bottom-up and top-down patterns are received in different

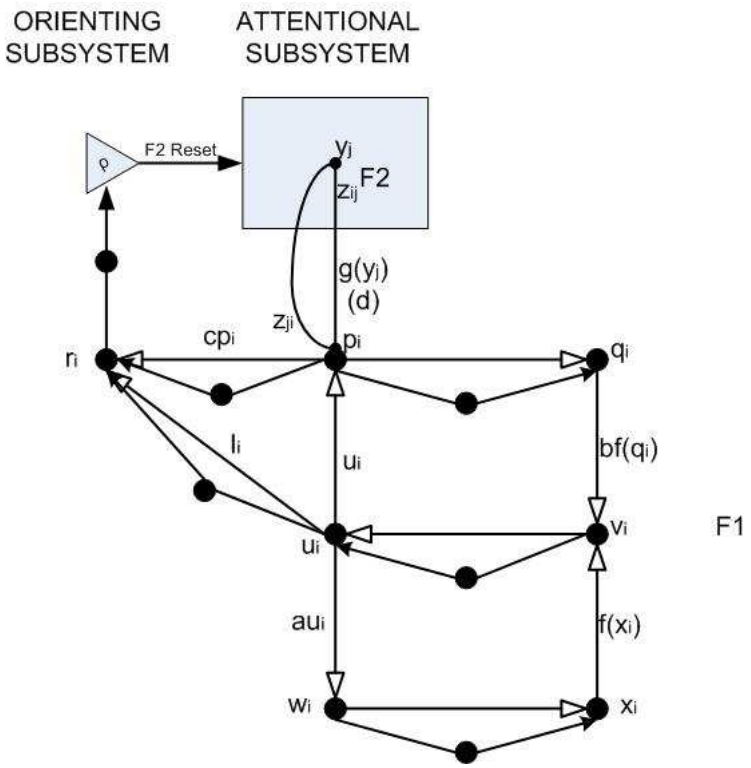

Figure 2.20. ART2 Structure

locations. Although F1 is more complex in ART2 compared to ART1, LTM equations are simpler in ART2 compared to ART1.

SPD of an ART2 system is maintained by learning stable code from arbitrary input and the way STM takes consider of historical data. The learning process is carried out by a parallel search in order to select appropriate recognition codes.

ART2 system is able to recognize an input pattern in F1 STM and find out the appropriate LTM pattern. In case the vigilance is high, F1 STM pattern by a bottom-up input is almost similar to a top-down F2 LTM established category. In case an uncommitted F2 pattern is first activated, although its top-down traces are set to zero, it must be able to encode the input. As F2 LTM has not previously learned a pattern, first STM bottom-up input should not result to a mismatch.

Invariance property of F1 provides transforming nonlinear input patterns in order to maintain the stability during learning process by augmenting the contrast or

eliminating the noise. STM invariance property prevents generation of a reset in case the LTM pattern perfectly matches by deactivating changes in STM at F1 levels. In order to achieve this property, additional input is defined in LTM top-down signals, so multiple F1 nodes exists. These additional levels normalizes the STM pattern before it is matched with top-down LTM at the middle F1 level. Furthermore, normalization of the activation patterns in F1 level is achieved by the black filled circles in Figure 2.20.

2.4.3. ART2 Equations

2.4.3.1. F1 STM Equations. If V_i is ith node, F1 STM equation is in the form:

$$\epsilon \frac{d}{dt} V_i = -AV_i + (1 - BV_i)\, J_i^+ - (C + DV_i)\, J_i^- \;\; (i = 1 \ldots M) \tag{2.24}$$

where M is the number of the nodes to be normalized, J_i^+ is total excitatory input, J_i^- is total inhibitory input and ϵ is the ratio between STM relaxation time and LTM relaxation time $0 < \epsilon << 1$

If B = 0, C = 0 and $\epsilon \to 0$

then

$$V_i = \frac{J_i^+}{A + DJ_i^-} \tag{2.25}$$

The following equations of $p_i, q_i, u_i, v_i, w_i, x_i$, which exists in Figure 2.20, characterize STM activities:

$$p_i = u_i + \sum g(y_j) z_{ji} \tag{2.26}$$

$$q_i = \frac{p_i}{e + ||p||} \tag{2.27}$$

$$u_i = \frac{v_i}{e + ||v||} \tag{2.28}$$

$$v_i = f(x_i) + bf(q_i) \tag{2.29}$$

$$w_i = I_i + au_i \tag{2.30}$$

$$x_i = \frac{w_i}{e + ||w||} \tag{2.31}$$

where w_i is the sum of an input vector I_i and internal feedback signal vector au_i, x_i represents the normalized w_i, x_i is transformed to v_i by a non-linear signal function, u_i represents the normalized v_i, p_i is the sum of interal F1 signal u_i and all F2 $\rightarrow$ F1 filtered signal, q_i represents the normalized p_i, $||V||$ indicates L_2 norm of a vector, y_j is the STM activity at the j_{th} node of F2 and f is nonlinear signal function in form:

$$f(x) = \frac{2\theta x^2}{x^2 + \theta^2} \; if 0 \leq x \leq \theta \tag{2.32}$$

$$f(x) = x \; if x \geq \theta \tag{2.33}$$

or if we linearize

$$f(x) = 0 \; if 0 \leq x \leq \theta \tag{2.34}$$

$$f(x) = x \; if x \geq \theta \tag{2.35}$$

2.4.3.2. F2 STM Equations. This level is the same with the one that exists in ART1. F2 is responsible from contrast enhancement of $F1 \rightarrow F2$ input and sending reset in case a pattern mismatch. F2 makes a choice when it receives maximum total input in all nodes. T_j is the sum of the F1 $\rightarrow$ F2 input to the j_{th} F2 node:

$$T_J = max \left\{ T_j = \sum p_i z_{ij} : j = M + 1 \ldots N \right\} \tag{2.36}$$

The orienting system send reset to F2 when a pattern mismatch occurs regarding the vigilance parameter. This reset causes the F2 nodes to block until all input from F1 to F2 finishes.

When F2 makes a choice, the gated dipole equation becomes:

$$g(y_J) = \begin{cases} d \; if T_J = max \{T_j\} \\ 0 \; otherwise \end{cases} \tag{2.37}$$

This reduce equation 2.26 as:

$$p_i = \begin{cases} u_i \; if F_2 \; is \; inactive \\ u_i + dz_{ji} \; if J_{th} F_2 \; node \; is \; active \end{cases} \tag{2.38}$$

2.4.3.3. ART2 LTM Equations. Top-down F2 to F1 equation is:

$$top - down(F_2 \rightarrow F_1) : \frac{d}{dt} z_{ji} = g(y_j)[p_i - z_{ji}] \tag{2.39}$$

Bottom-up F1 to F2 equation is:

$$bottom - up(F_1 \rightarrow F_2) : \frac{d}{dt} z_{ij} = g(y_j)[p_i - z_{ij}] \tag{2.40}$$

When F_2 makes a choice, assuming the J_{th} node is active, the equations become:

$$\frac{d}{dt} z_{ji} = d\,[p_i - z_{ji}] = d\,(1-d)\left[\frac{u_i}{1-d} - z_{ji}\right] \tag{2.41}$$

$$\frac{d}{dt} z_{ij} = d\,[p_i - z_{ij}] = d\,(1-d)\left[\frac{u_i}{1-d} - z_{ij}\right] \tag{2.42}$$

where $0 < d < 1$ if $j \neq J$, $\frac{d}{dt} z_{ji}$ and $\frac{d}{dt} z_{ij}$ is zero.

2.4.3.4. ART2 Reset Equations (Orienting Subsystem). The similarity between the pattern at F1 STM and LTM is determined via the vector r = $(r_1 \ldots r_M)$

Figure 2.21 shows the relation between $||r||$ and $||cdz_j||$. Pattern mismatch

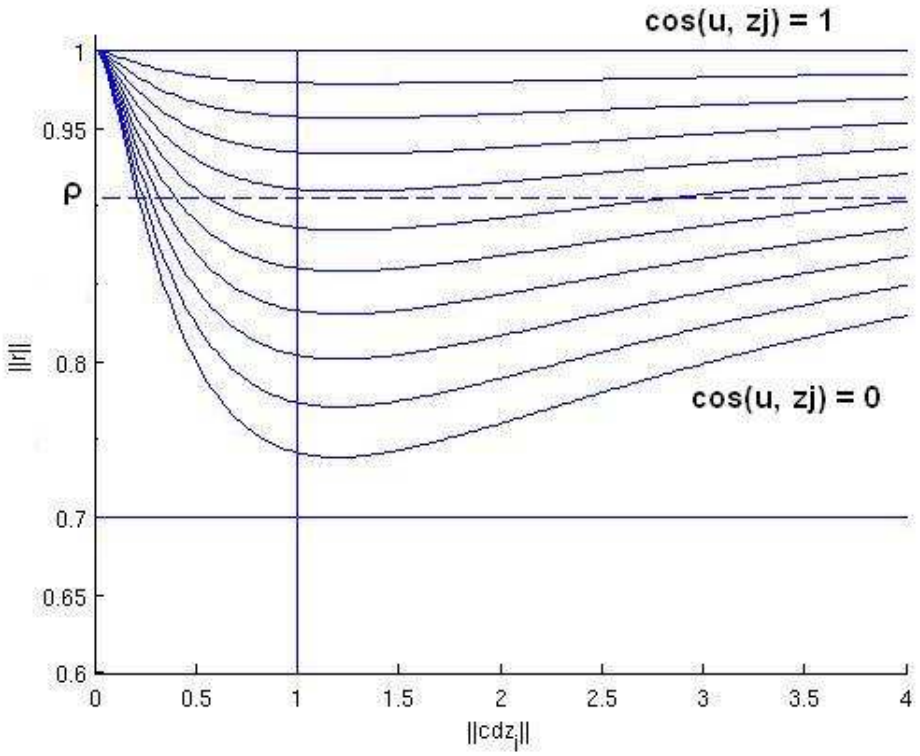

Figure 2.21. ART2 Reset Decision Indicator

(reset) occurs when $||r||$ is under the vigilance (ρ) threshold. r_i is defined as:

$$r_i = \frac{u_i + cp_i}{e + ||u|| + ||cp||} \tag{2.43}$$

where c < 0, reset occurs when

$$\frac{\rho}{e + ||r||} > 1 \tag{2.44}$$

where $0 < \rho < 1$

Top-down LTM Value

When J_{th} node of F_2 is active, equation 2.41 becomes:

$$||r|| = \frac{[1 + 2||cp||cos(u, p) + ||cp||^2]^{1/2}}{1 + ||cp||} \tag{2.45}$$

where cos(u,p) is the cosine of the angle between two vectors $p = u + dz_j$

Since $||u|| = 1$,

$$||p|| = cos(u, p) = 1 + ||dz_j||cos(u, z_j) \tag{2.46}$$

Finally

$$||r|| = \frac{[(1+c)^2 + 2(1+c)||cdz_j||cos(u, z_j) + ||cdz_j||^2]^{1/2}}{1 + [c^2 + 2c||cdz_j||cos(u, z_j) + ||cdz_j||^2]^{1/2}} \tag{2.47}$$

When $cos(u, z_j) = 1$, $||r|| = 1$
which means the angle between the vectors is 0 and they exactly match

J_{th} node of F_2 stays active while $\rho \leq ||r||$

ART2 ensures not to send a reset while a new category is being learned. When a new F_2 node becomes active, this is done by reducing the $||z_j||$ values:

$$z_{ji} = 0, i = 1...M \ and \ j = M + 1...N \tag{2.48}$$

2.5. Kalman Filter

Kalman Filter [7, 8], which is developed by Rudolf Kalman, can be defined as a "Optimal Recursive Data Processing Algorithm" where "Optimal" stands for its ability to interpret any kind of data that is provided, "Recursive" for its capability to remind and consider the previously processed data and "Data Processing Algorithm" simply indicates that it is a filter. It is capable of estimating the state of a system by analyzing incomplete and noisy data.

Kalman Filter provides a solution to discrete-data linear filtering problem. It tries to estimate the state x $\in \Re$ of the linear equation:

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1} \tag{2.49}$$

with a measurement z which is:

$$z_k = Hx_k + v_k \tag{2.50}$$

where w_k represents the processs noise and v_k represents the measurement noise. w_k and v_k supports the following probability distributions:

$$p(w) \cong N(0, Q) \tag{2.51}$$

$$p(v) \cong N(0, R) \tag{2.52}$$

where Q is the process noise covariance and R is the measurement noise covariance matrix. A is the state transition model which is a mapping from previous state to the current one, B carries the control-input u to state x and H is the observation model which is a mapping from states to measurements.

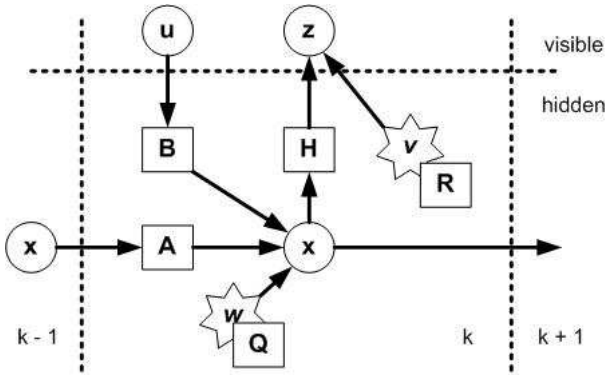

Figure 2.22. Kalman Filter Model

Figure 2.22 demonstrates Kalman Filter, indicating the related elements: Cir-

cles are vectors, squares are matrices, stars are gaussian noise with related covariance matrice at lower right.

2.5.1. Kalman Filter Computational Origins

In order to make an estimation of the current state, previous estimated state and current observation are needed. Two main variables indicate the state of the filter: $\hat{x}_{k|k}$: State estimation at time k and $P_{k|k}$: Error covariance matrice

Beside:

$\hat{x}_{k|k-1}$: State prediction given the state at time k-1

Estimate errors are:

$$e_{k|k-1} \equiv x_k - \hat{x}_{k|k-1}$$

$$e_{k|k} \equiv x_k - \hat{x}_{k|k}$$

Estimate error covariance matrices are:

$$P_{k|k-1} = E\left[e_{k|k-1}\ e_{k|k-1}^T\right]$$

$$P_{k|k} = E\left[e_{k|k}\ e_{k|k}^T\right]$$

State estimation is defined as:

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K(z_k - H\hat{x}_{k|k-1}) \tag{2.53}$$

where z_k is actual observation, $H\hat{x}_{k|k-1}$ is observation prediction and the difference $z_k - H\hat{x}_{k|k-1}$ is called observation innovation or residual. If this value is zero, that means the prediction is successful.

K is the optimal Kalman gain that minimizes error covariance:

$$K_k = \frac{P_{k|k-1}H^T}{HP_{k|k-1}H^T + R} \tag{2.54}$$

We can observe from Eq. 2.54 that

$lim_{R->0}K_k = H^{-1}$

$lim_{P_{k|k-1}->0}K_k = 0$

Briefly the main two variables in Kalman are defined as:

$E[x_k] = \hat{x}_{k|k}$

$E[(x_k - \hat{x}_k)(x_k - \hat{x}_k)^T] = P_k$

2.5.2. Kalman Filter Algorithm

Kalman Filter is a form of feedback control system that makes the estimations and takes noisy observations as feedback. Equations of Kalman is divided into two groups: Time update equations and measurement update equations. Time update makes prediction about the next step while measurement update is responsible for making the current state estimation regarding the prediction and observation. Measurement update works as a corrector while time update works as an predictor:

Time Update Equations:

$$\hat{x}_{k|k-1} = A\hat{x}_{k-1} + Bu_{k-1} \tag{2.55}$$

$$P_{k|k-1} = AP_{k-1}A^T + Q \tag{2.56}$$

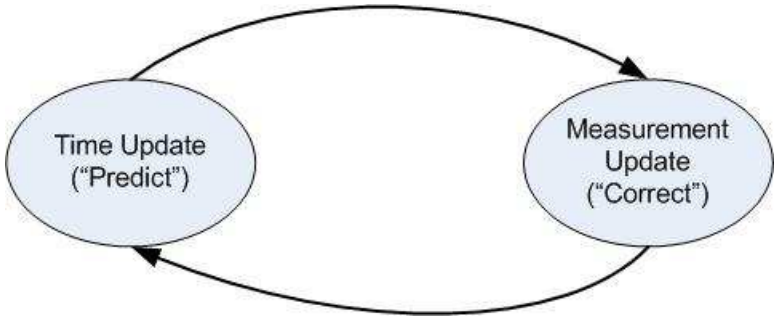

Figure 2.23. Kalman Filter Cycle

These equations give the state and covariance predictions from time k-1 to time k.

Measurement Update Equations:

$$K_k = P_{k|k-1}H^T(HP_{k|k-1}H^T + R)^{-1} \tag{2.57}$$

$$\hat{x}_k = \hat{x}_{k|k-1} + K_k(z_k - H\hat{x}_{k|k-1}) \tag{2.58}$$

$$P_k = (I - K_kH)P_{k|k-1} \tag{2.59}$$

The first step of measurement update is Kalman gain K_k in Eq. 2.57. The next step is making state estimation considering current state observation in Eq. 2.58. The last step is calculating error convariance via Eq. 2.59. After each time step, the values calculated from the previous state is used which forms the recursive structure of the filter.

3. ARKAQ Learning

ARKAQ Learning [10] is an unsupervised learning algorithm. It is suitable for the agents trying to survive in partially observable environments. Nevertheless, the continuous structure of the world makes it diffucult to rely on a model. Therefore, the world is segmented by the algorithm in order to apply the MDP model.

Noisy and incomplete perceptions is one of the features that distiguishes POMDP from MDP. Kalman Filter is a recursive filter which estimates the current state. Besides that, Kalman Filter keeps the past history implicitly, which is important for POMDP solutions. Furthermore, Kalman Filter is able to make state estimations from incomplete and noisy inputs.

The algorithm performs a state segmentation in order to convert the non-Markovian world to Markovian world, thus modeling the environment under POMDP becomes possible. ART2 [26] network is used for dividing continuous world into discrete states.

ARKAQ algorithm is based on Q-Learning [9]. Q-Learning is able to solve MDP problems [28] by iterative approximations of Q(s,a) which is an estimation of expected reward of an action a on state s. The optimal policy of MDP is

$$\prod(s) = argmax_{a \in A} Q(s, a) \tag{3.1}$$

In POMDP, since the state is no more given by the environment, the (state,action) pair becomes (observation,action).

Q-Learning (s,a) or (o,a) values are expected to converge to optimal policy after an episode. The uncertainty in the environment directly affects the speed of convergence of the Q values. There are various exploration-exploitation techniques proposed for the speed and optimality of Q convergency like Boltzman [14] which is used in ARKAQ.

Boltzman exploration has a factor T, called temperature, which decreases by exploration time. T value determines the exploration amount. It can be defined as the probability of choosing an action different than the one with the highest Q value. Initially T is high, exploration is high which means a random action is selected. By the time T decreases, exploitation probability gets higher and the action with the highest Q value will be selected. The probability of selecting an action according to Boltzman is:

$$Pr_s(a) = \frac{e^{Q(s,a)/T}}{\sum_{b \in A} e^{Q(s,b)/T}} \tag{3.2}$$

where A is the finite set of actions and $\sum_a Pr_s(a) = 1$

After an episode, T approaches 0, which means the action with the maximum Q value will be selected:

$$p_s(a) = \begin{cases} 1 \; if \; Q(s,a) = argmax_{a \in A} Q(s,a) \\ 0 \; if otherwise \end{cases} \tag{3.3}$$

Briefly, ARKAQ algorithm utilizes Kalman Filtering, ART2 Networks and Q-Learning.

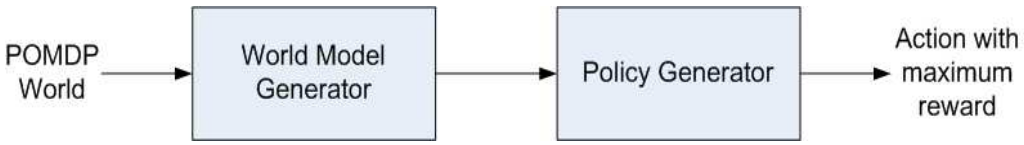

Figure 3.1. ARKAQ Structure

Figure 3.1 illustrates the architecture of ARKAQ consisting of two layers: World Model Generator consists of Kalman Filter and ART2 Network. This layer provides Markovian model to Policy Generator which uses Q-Learning to achieve the optimum policy.

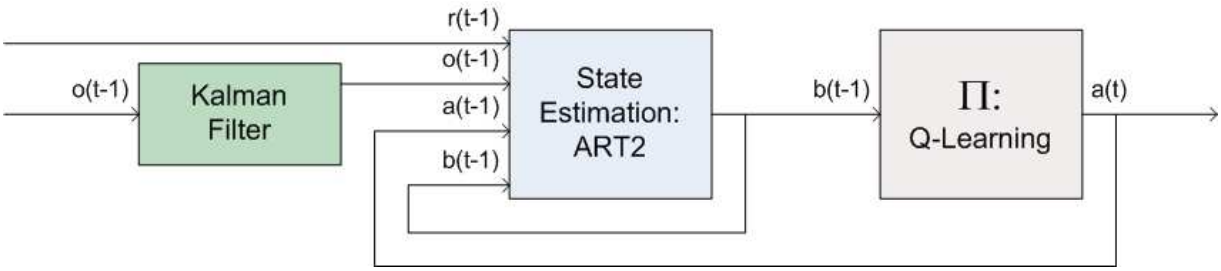

Figure 3.2. ARKAQ Structure

Figure 3.2 illustrates ARKAQ in detail. Policy Layer $\prod$ is activated after the state segmentation of ART2 is converged. The optimal action is the one with the highest Q value. For some states, more than one action may have high Q values, which indicates that more than one optimal action exists.

4. Implementation

4.1. Kalman Filter

Kalman Filter implementation requires various type of matrix operations like transpose, inverse, multiplication.

Boost [29] is a collection of peer-reviewed C++ libraries. Blas is the linear algebra library in Boost library tree which is implemented under Fortran programming language. Lapack is a high-level library that uses Blas at background and LaPackpp is the C version of Lapack. Boost is used while implementing Kalman Filter.

The main two fundamental parameters, dependant on the environment, of Kalman Filter are Q and R matrices. In the prediction phase of Kalman, Q is used as the process noise covariance which consists of state transition error parameter. In the correction phase, R is used as observation noise covariance consisting measurement error parameters. These matrices are defined when constructing Kalman filter instances.

4.2. ART2 Network

The vigilance value ρ is the parameter of ART2 that defines the similarity between two cluster centers. Depending on the dynamics of the problem, changing the vigilance value may increase or decrease cluster center number.

Beside that, ART2 may have different methods in order to measure the distance between two cluster centers. Two different methods are implemented: angular distance, euclidean distance.

4.2.1. Angular Distance

Angular distance calculates the cosine of the angle between two vectors in order to calculate the similarity. For example, if the angle between cluster center w_1 and a new node x_1 is α, $\cos(\alpha) > \rho$ means the angle is small enough for similarity.

Figure 4.1 illustrates the cluster center weight update when a new node is presented to the network. w_1 and w_2 are the cluster centers and x_1 and x_2 are the nodes. At t = 0, w_1 is the cluster center most aligned to x_1. When x_1 is presented several times, w_1 is updated towards x_1 as shown at t = 1.

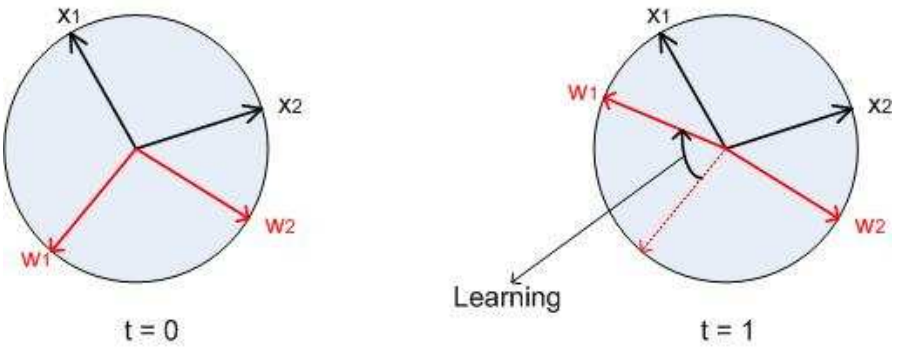

Figure 4.1. Angular Distance Update

Calculation of angular distance is achieved by dot product of two vectors, which is an operation that takes two vectors as input and returns the scalar quantity. Dot product of two vectors a and b with dimension n is:

$$a.b = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + ... + a_n b_n \tag{4.1}$$

While implementing the angular distance, since the different type of inputs have different intervals, inputs are normalised to the interval [-1, 1].

In Euclidean space, dot product of two vectors is:

$$a.b = |a||b|cos\theta \tag{4.2}$$

where θ is the angle between two vectors.

Since we have normalized the input vectors, $|a| = 1$ and $|b| = 1$, our equation becomes:

$$a.b = cos\theta \tag{4.3}$$

which is the value we need in order to compare two vector similarity by taking consider the vigilance.

4.2.2. Euclidean Distance

When we use the euclidean distance [15], if the closest cluster center distance is smaller than *vigilance*, the update rule is performed with online k-means [15]. Otherwise, in case the distance is larger, a new cluster center is created. Online k-means simply moves the closest cluster center towards to the node. The algorithm primarily points out the cluster center with the minimum euclidean distance:

$$argmin||x - m_i|| \tag{4.4}$$

Euclidean distance between two points x and y in Euclidean n-space is defined as:

$$\begin{aligned} x - y &= \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + ... + (x_n - y_n)^2} \\ &= \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \end{aligned} \tag{4.5}$$

Later, update to the cluster center is applied by formula:

$$m_i = m_i + \eta(x - m_i) \tag{4.6}$$

where η is the update factor

Figure 4.2 illustrates online k-means update. In the figure, the distance from x_1 to m_1 is close enough regarding to the vigilance value and cluster center m_1 is updated. Nevertheless, x_2 does not belong to a cluster center regarding to the vigilance, thus a new cluster center is created.

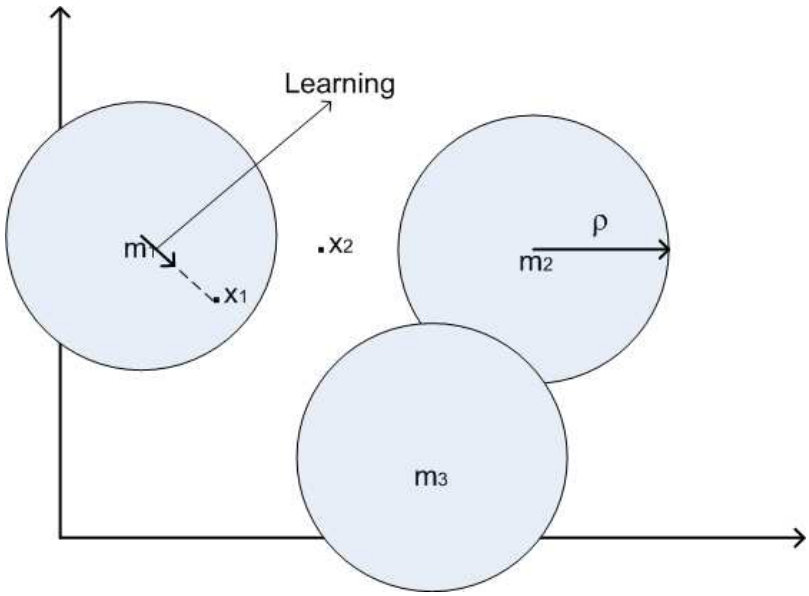

Figure 4.2. Euclidean Distance Update

4.3. Q-Learning

The Q-Learning phase includes exploration and exploitation phases. In the exploration phase, the agent performs random actions in order to explore the world. In the exploitation phase, the agent takes its actions by considering the existing Q values.

Q matrix convengence in the exploration phase is mostly related with performing all possible (s,a) pairs several times in order to get the Q matrix filled. On the other hand, convergence in the exploitation phase can be determined by variation delta of Q values.

The exploration phase is realized as in Figure 4.3.

```
procedure Q-Learning
begin
    t = 0;
    initialize Q[S][A] matrix;
    while Q Matrix is not converged
    select random action a;
    identify current state s;
        while current state remains same do
            perform action a
        end
        receive immediate reward r;
        receive new state s';
        evaluate Q[s][a];
    end
end.
```

Figure 4.3. Pseudo-code Q-Learning Exploration

The learning rate ($\alpha = 1$) can be decreased along with time. After learning process finishes, in other words Q matrix converges, exploitation phase is realized as in Figure 4.4.

```
procedure Q-Learning
begin
    t = 0;
    read Q[S][A] matrix;
    while Q Matrix is not converged
    select select action a within the ones with high Q value;
    identify current state s;
        while current state remains same do
            perform action a
        end
        receive immediate reward r;
        receive new state s';
        evaluate Q[s][a];
    end
end.
```

Figure 4.4. Pseudo-code Q-Learning Exploitation

5. Experiments

5.1. Experiment Environment

Implementation of the algorithms are realized under C++ using Eclipse [30] IDE under Linux Operating System on VMWARE with 512 MB. of RAM and 2.16 GHz. T2600 Intel CPU. AIBO [11] is used in our experiments, which is a four-legged robot manufactured by Sony. Webots [31] is a software which is capable of simulating mobile robots, including AIBO, and is used for implementing our algorithms on AIBOs.

5.1.1. Webots

Webots is a robot simulation software which is widely used in academic and education. It is able to model and simulate various kinds of robots, as well as AIBO. It includes a complete library of sensors and actuators and able to control all 20 servos of AIBO. Furthermore, it has a "supervisor" module which can be used when we need to modify the environment externally.

Simulator screenshot that shows the experimental setup is given in Figure 5.1.

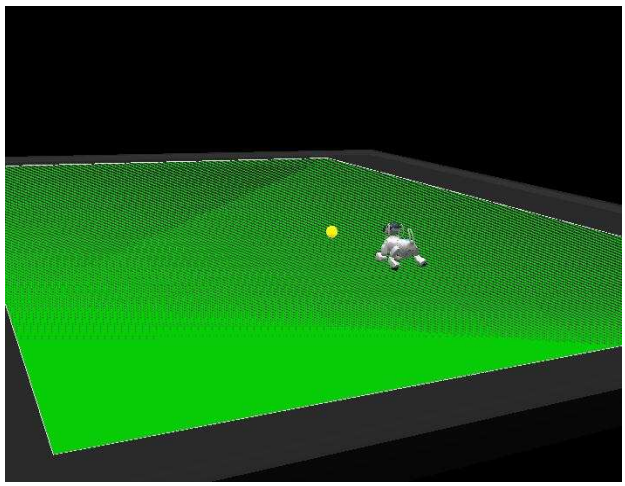

Figure 5.1. Webots simulation environment

The control panel for AIBO is given in Figure 5.2. We can control the actuators from this panel. This is necessary when we want to set the orientation of any actuator manually.

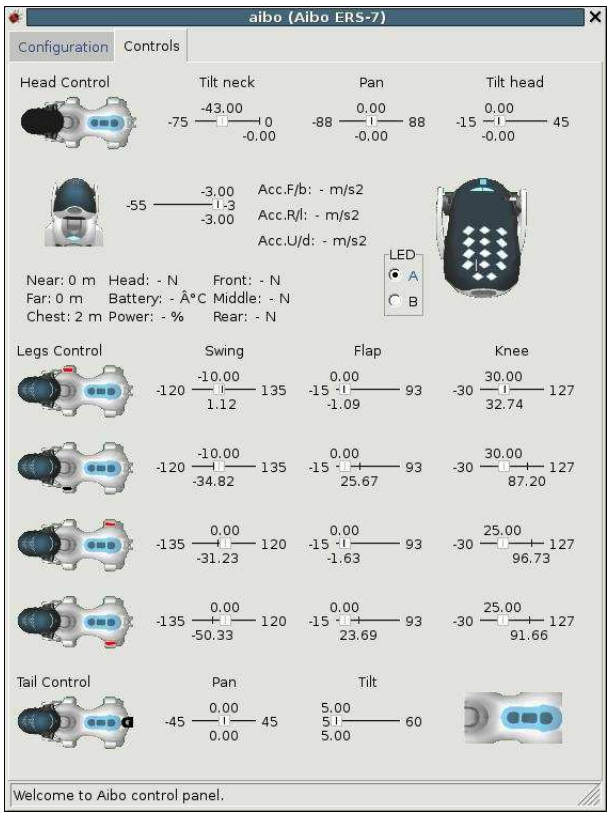

Figure 5.2. AIBO Control Panel

The camera vision of AIBO is given in Figure 5.3. This will be the sensor environment in our problems.

Figure 5.3. AIBO camera view

5.1.2. Cerberus Codebase

Cerberus codebase is used as a framework while interacting with AIBO. It is possible to execute Cerberus codebase on Webots with minor effort. It has various modules like vision, communication, localization and planner which is represented in Figure 5.4

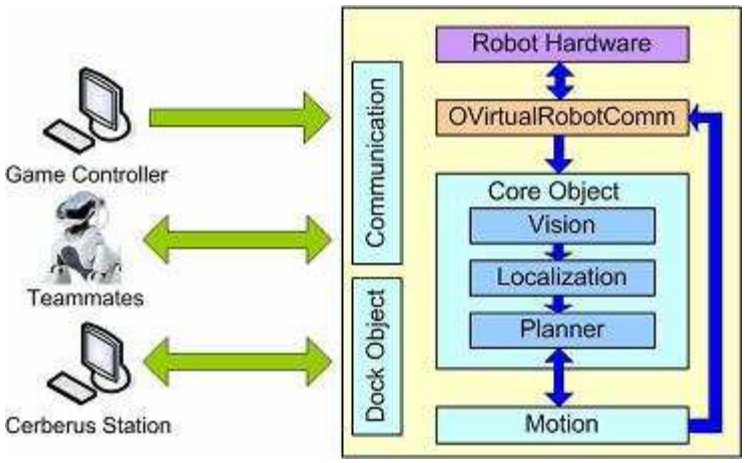

Figure 5.4. Cerberus Structure [32]

While implementing the algorithms, the localization and the planner modules are bypassed. So the following two modules are the main elements that our algorithms interact with:

5.1.2.1. Vision Engine. This module is capable of processing the data from the vision of the camera and can give us information about the environment. We'll be using the informations "distance to the ball", "orientation to the ball" and "orientation to the goal bar" in our experiments.

5.1.2.2. Motion Engine. This module has high-level functions for controlling the actuators. We'll be mainly using the walk and the turn functions.

5.1.3. Environmental Uncertainties

The main environmental uncertainties that are encountered during the implementation are given below.

5.1.3.1. Move forward. Moving straight forward under Webots is not possible. On a real AIBO, the battery is located in the left side of the body and that's why the center of gravity of AIBO is on the left. In order to handle this issue, motion engine has some methods but this creates undesired results in simulation environment. Figure 5.5 shows the path of the move forward action. We can see that AIBO slightly goes to left and rotates to right.

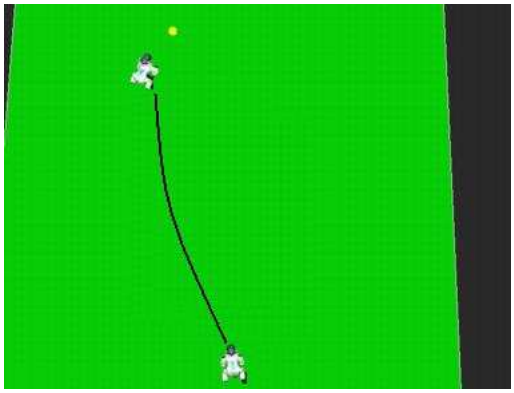

Figure 5.5. Uncertainties in simulation environment

5.1.3.2. Inaccurate vision calculations. While the servos of the head are moving, the vision calculations become continuously inaccurate. For that reason, the observations are not taken while the head is not static.

5.1.3.3. Grabbing and turning. Grabbing the ball is highly dependant to the jaw servo and also to the friction of the ball. In simulation environment, these two issues are not modeled as in the real environment, hence grabbing the ball and turning with the ball is not possible. In order to handle this problem, supervisor is used to set the location of the ball before the kick act.

5.2. Ball Approaching

In this problem, the goal of AIBO is to approach the ball by performing a sequence of the three possible actions: MOVE FORWARD, MOVE RIGHT, MOVE LEFT. These actions are realized by performing them during a predefined number of steps. Initially, AIBO does not know the consequences of its actions, however after performing an action, depending on the state, it receives a reward or a punishment. Environmental signals "ball distance" and "ball orientation" are obtained from the vision engine of Cerberus [12]. So our input vector is $\langle distance, orientation \rangle$. The ball distance input interval is [200, 4100] in millimeters and orientation interval is [-0.5, 0.5] in radians.

Kalman parameter R is defined as R=$\langle 100, 0.05 \rangle$ because in far distances, oscillations up to 100 mm. for distance and 0.05 rad. for orientation are observed. The other parameter is Q=$\langle 0.001, 0.0001, 0.0001, 0.000001 \rangle$, where the elements are distance, orientation, distance dx and orientation dy, respectively.

5.2.1. State Segmentation

The distance and orientation values are normalized before entering the ART network. Since the ball distance that is obtained from vision engine is in range [200, 4100], the value is normalized by *(distance - 2150)/1950*. The orientation value is normalized by *orientation*2*.

State segmentation is achieved via both angular distance and euclidean distance with various vigilance values. State segmentation learning phase is achieved by performing random actions on the field. Figure 5.6 illustrates the segmentation algorithm automaton.

Each pattern from the observation is fed to the ART2 Network. In the case, where the ball signals cannot be received, unknown state sends a "reset" request to the supervisor and environment is reset by the supervisor. Additionally, the supervisor sends the environment reset signal periodically. ART2 Network processes the input

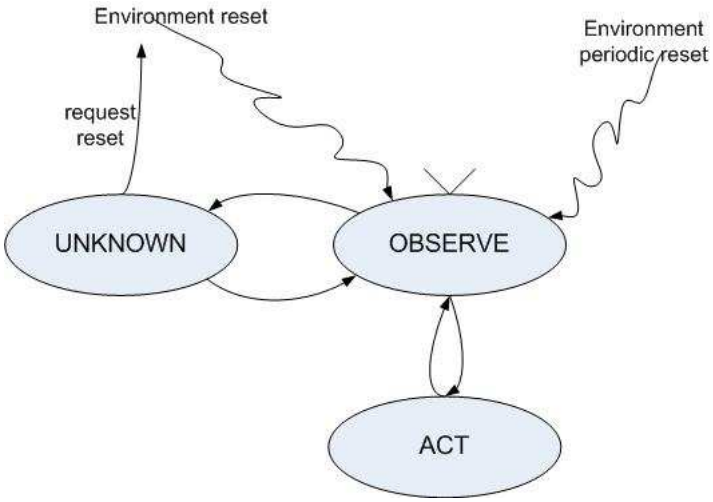

Figure 5.6. ART2 Segmentation finite automaton

vector to check whether it is close to an existing cluster center upon to the vigilance value. If it is close enough, the related cluster center is updated, otherwise a new cluster center is created.

In order for the segmentation to cover all possible points in the field, the supervisor resets the environment. The ball position is reset under a constraint: it should be placed inside the field of view of the robot; hence, the robot will not have to search for the ball. Additionally, the supervisor also resets the environment after an episode. Figure 5.7 is the screenshot showing an AIBOs field of view. After the reset, the ball position is randomly set within the sight lines.

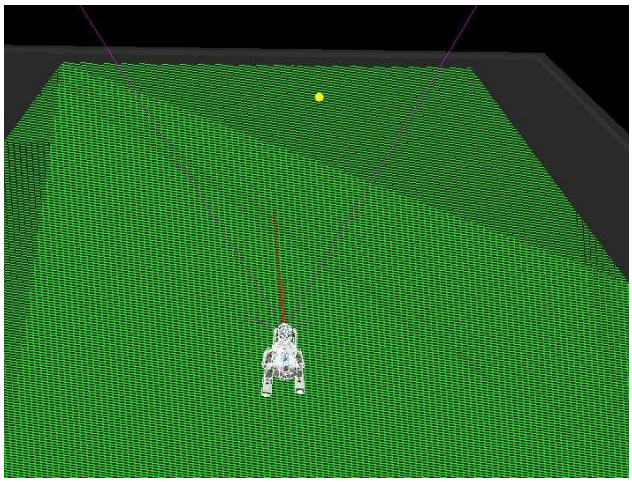

Figure 5.7. AIBOs field of view

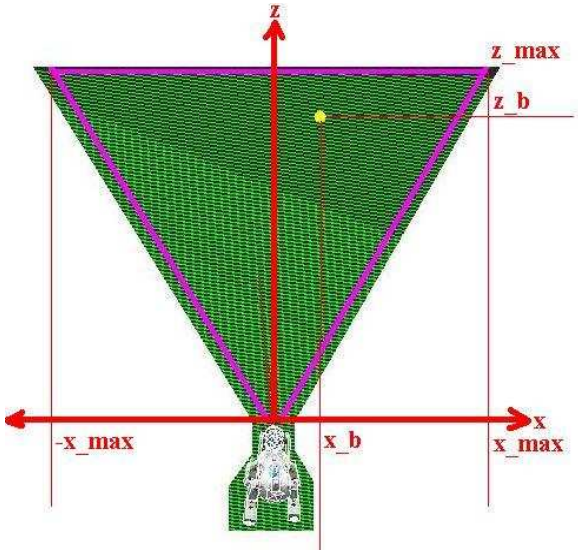

Figure 5.8. Random Ball Position

Figure 5.8 illustrates how the random ball position is calculated after each period. Even though the world has three dimensions, the y position of the ball is always zero and that is why it is not shown in the figure. Initially, a random value for z_b between [0, z_{max}] is calculated. Later, the value x_b on the x axis is calculated by

$$\frac{z_b}{z_{max}} = \frac{x_b}{x_{max}} \tag{5.1}$$

due to the triangle similarity. The value of x_b is the border of triangle, so we multiply it with a random number between zero and one. Finally, the sign of x_b is determined randomly.

Angular distance cluster centers(templates) with vigilance value 0.95 are shown in Figure 5.9. The point [0, 0] can be considered as the location of the AIBO and the ball can be on any of the point in the figure. The field is segmented according to the vectors that are presented in the figure. In angular distance, each vector template is a cluster center including the surrounding vectors and the origin point [2150, 0] comes from denormalization. The borders of the clusters are defined regarding to the vigilance parameter. For example, if the ball orientation is 0.3 and the distance is 2100, as indicated on the location 'x' that an arrow points in the figure, it will belong to the cluster center which is circled around.

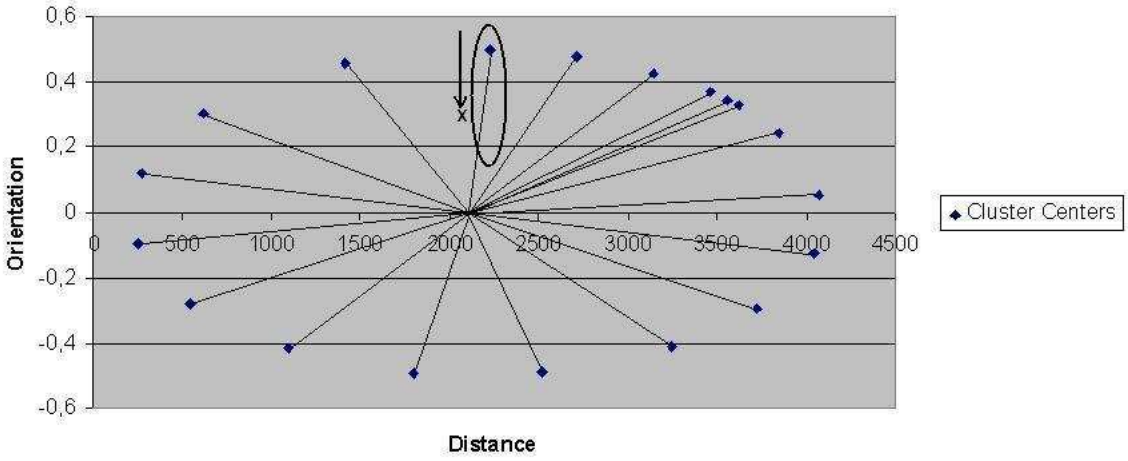

Figure 5.9. ART Network angular distance cluster centers

After the state segmentation phase is finished, the patterns state is calculated by determining the template which has the maximum dot product value with the cluster centers pattern vector.

According to the dynamics of the problem, euclidean distance ART network is a better choice than the angular distance clustering because state transitions in angular distance takes only the angular value into consider, which is not our goal. Figure 5.10, Figure 5.11, Figure 5.12, Figure 5.13 and Figure 5.14 shows euclidean distance cluster centers with various vigilance parameters. The dots are the cluster centers and the location of AIBO is on the point [0,0]. The ball can be on any of the point in the field and it belongs to the nearest cluster center.

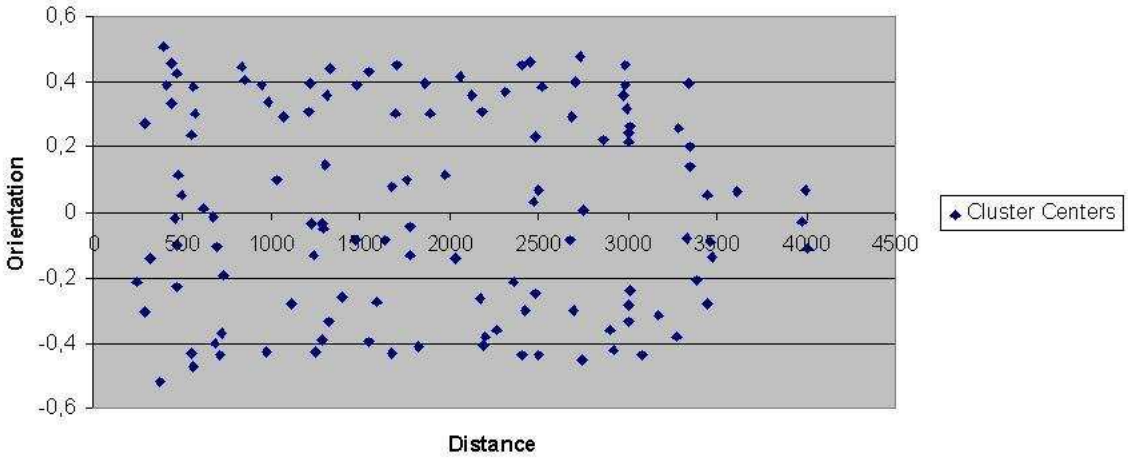

Figure 5.10. ART Network euclidean distance, 137 cluster centers with vigilance 0.15

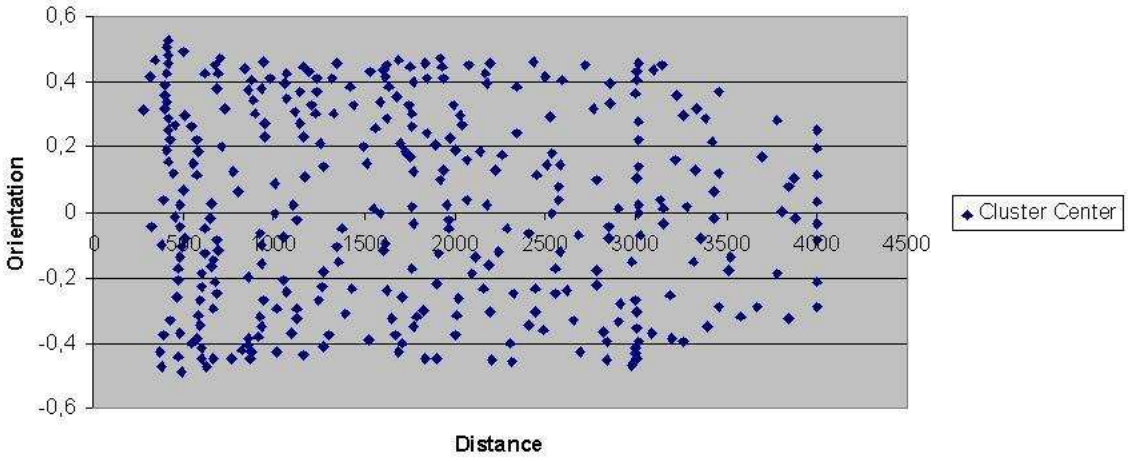

Figure 5.11. ART Network euclidean distance, 333 cluster centers with vigilance 0.1

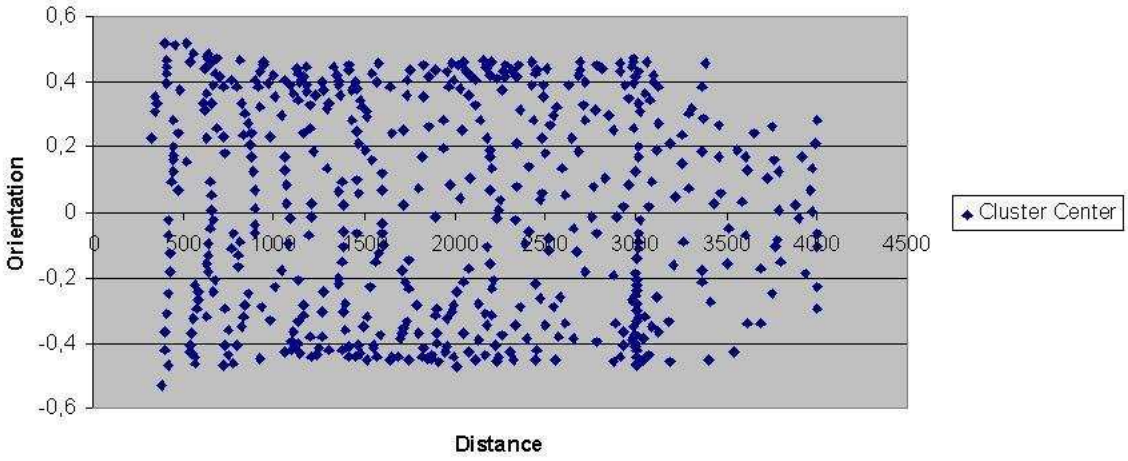

Figure 5.12. ART Network euclidean distance, 519 cluster centers with vigilance 0.075

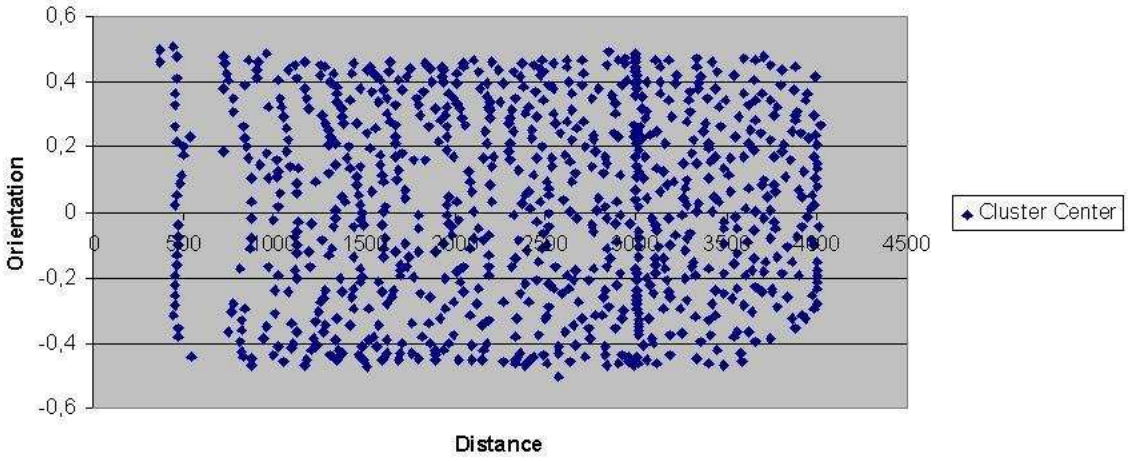

Figure 5.13. ART Network euclidean distance, 825 cluster centers with vigilance 0.05

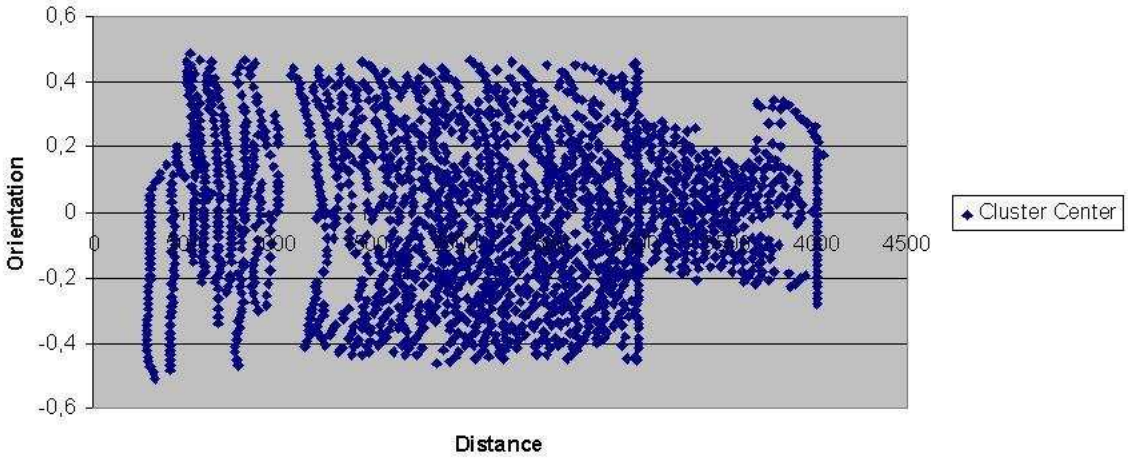

Figure 5.14. ART Network euclidean distance, 1933 cluster centers with vigilance 0.025

5.2.2. Q Learning Phase

The Q-Learning phase begins with the exploration of the world in order to cover all the (s,a) space. Similar to the state segmentation phase, the environment is periodically reset by the supervisor after a predefined episode. Ball distances smaller than 500 mm. are rewarded by +100 and in that case an extra reward +50(totally +150) is given if the orientation belongs to interval [-0.05, 0.05]. The exploration phase tries to execute unvisited (s,a) pairs as much as possible. The exploration automaton is given in Figure 5.15:

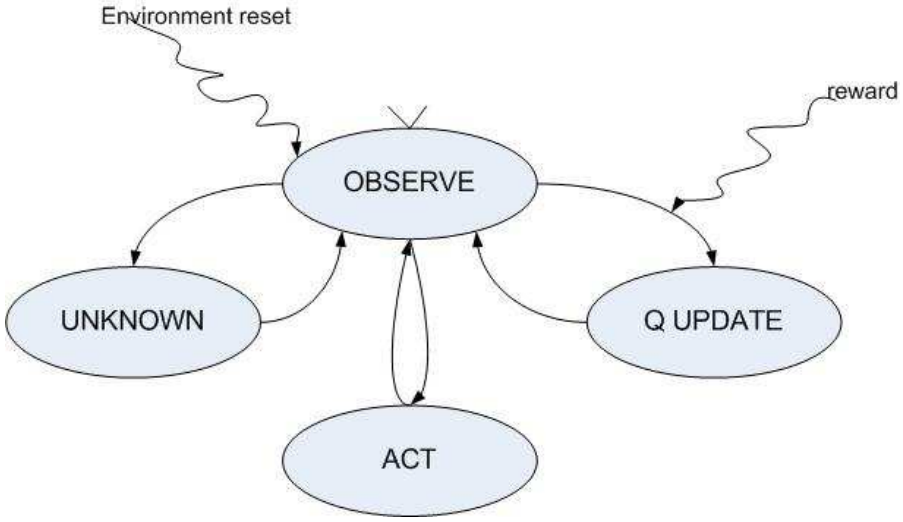

Figure 5.15. Exploration Phase Automaton

Resulting Q Matrix of related problem for the ART Network with 333 cluster centers and vigilance value 0.1 is illustrated in Figure 5.16. Since the location of the AIBO is the reference point in our problem, it is fixed as shown in the figure and the ball can be on any of the point. The arrows on the figure indicate the actions FORWARD, RIGHT or LEFT. Regarding to the location of the ball in the AIBOs field of view, the indicated action is selected. For example, if we consider the left border, AIBO mostly tries to move to the left in order to make the orientation of the ball to be closer to the center.

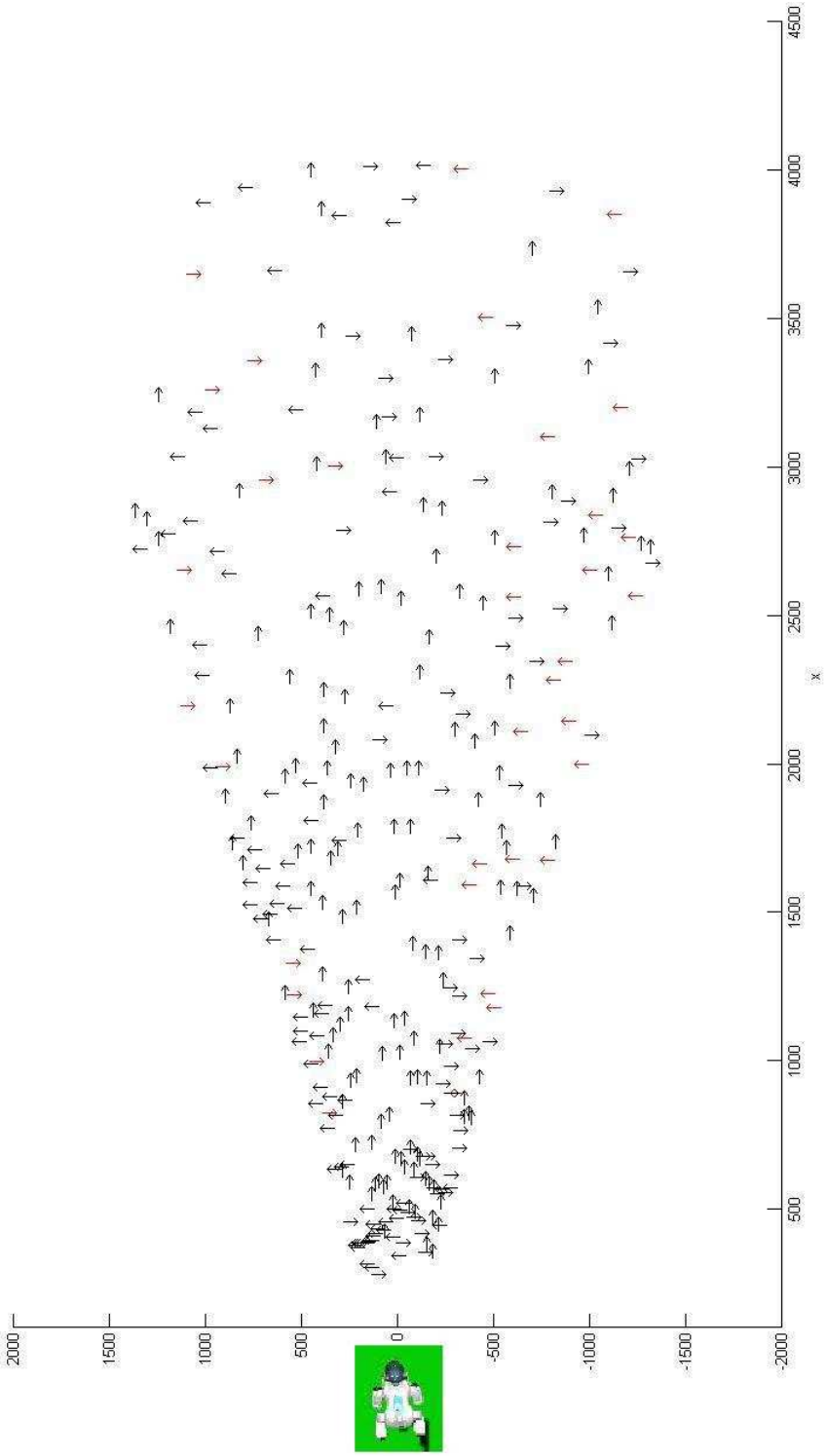

Figure 5.16. Resulting actions inferred from Q values

The following figures contains various screenshots of the trajectories of AIBO while trying to approach to the ball by performing the maximum Q valued actions in the Q matrix. The success of the ARKAQ algoritm depends to the vigilance parameter. The trajectory will be more straight if we increase the number of states, in other words decrease the vigilance parameter, or increase the number of actions.

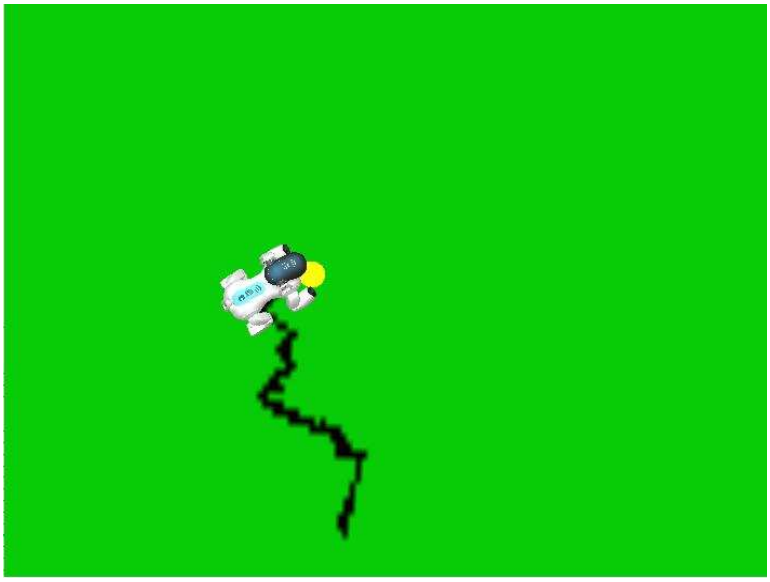

Figure 5.17. ARKAQ Ball Approaching Trajectory 1

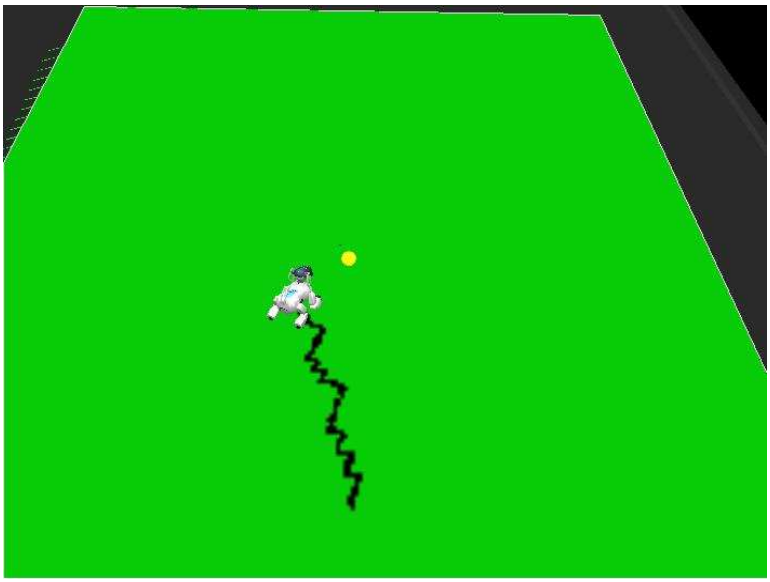

Figure 5.18. ARKAQ Ball Approaching Trajectory 2

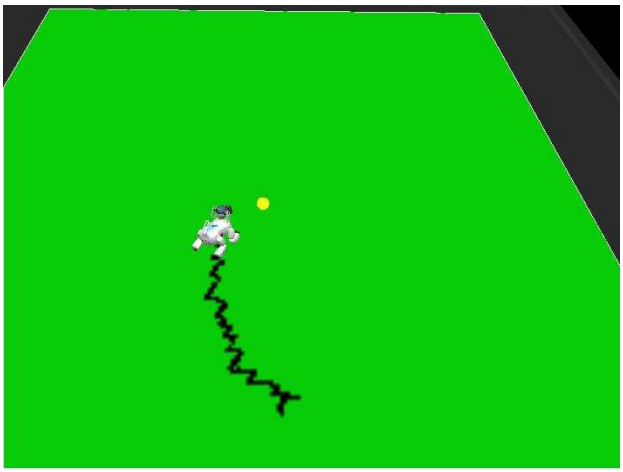

Figure 5.19. ARKAQ Ball Approaching Trajectory 3

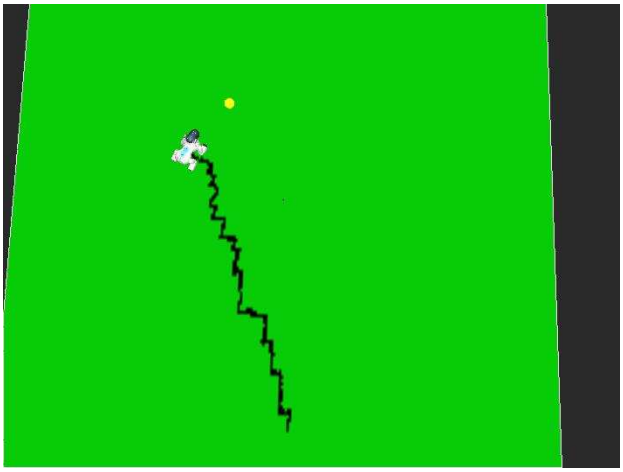

Figure 5.20. ARKAQ Ball Approaching Trajectory 4

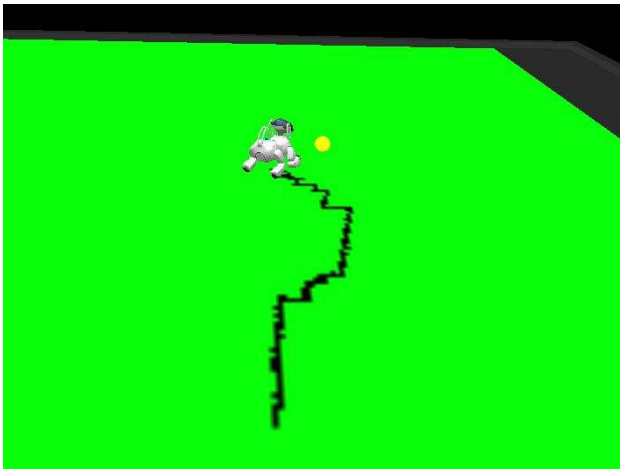

Figure 5.21. ARKAQ Ball Approaching Trajectory 5

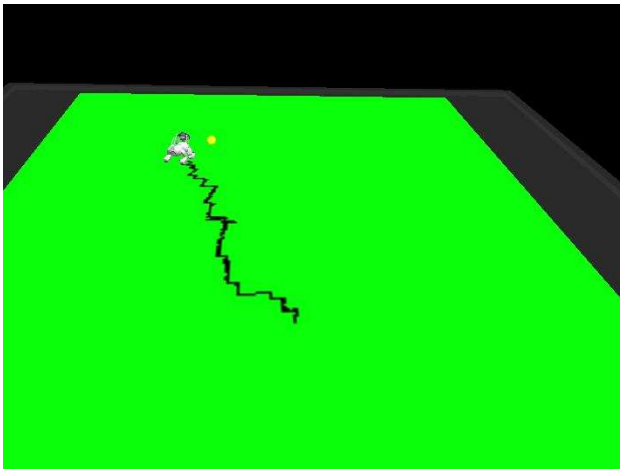

Figure 5.22. ARKAQ Ball Approaching Trajectory 6

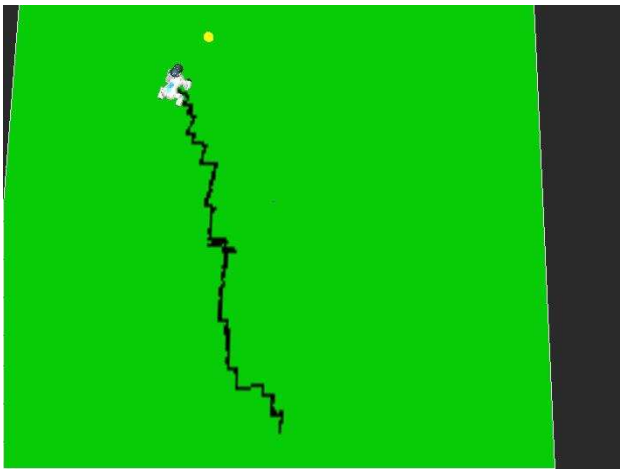

Figure 5.23. ARKAQ Ball Approaching Trajectory 6 (Upper View)

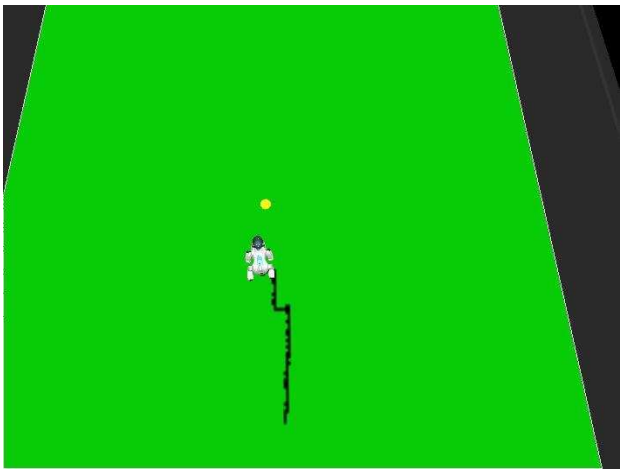

Figure 5.24. ARKAQ Ball Approaching Trajectory 7

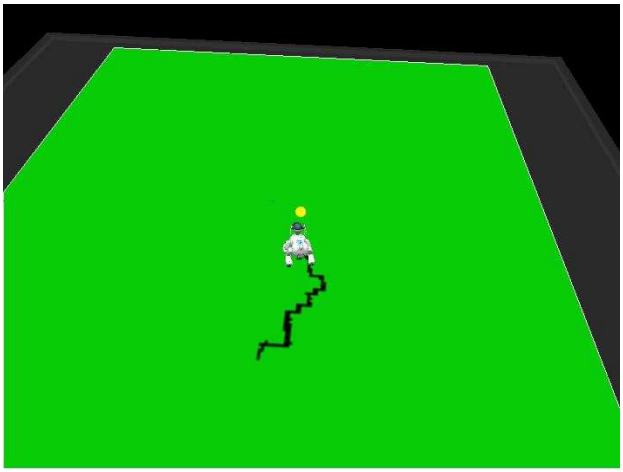

Figure 5.25. ARKAQ Ball Approaching Trajectory 8

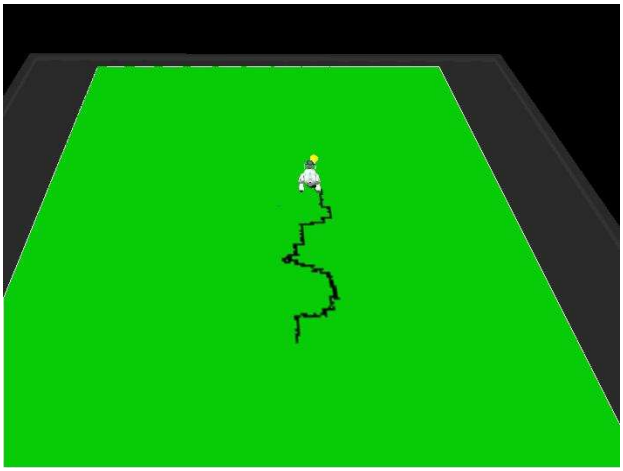

Figure 5.26. ARKAQ Ball Approaching Trajectory 9

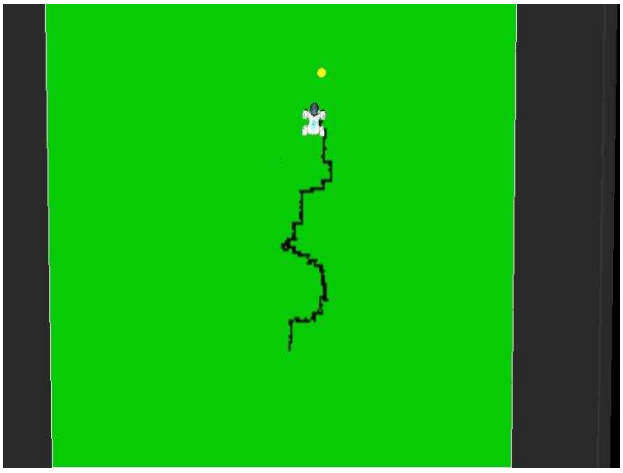

Figure 5.27. ARKAQ Ball Approaching Trajectory 9 (Upper View)

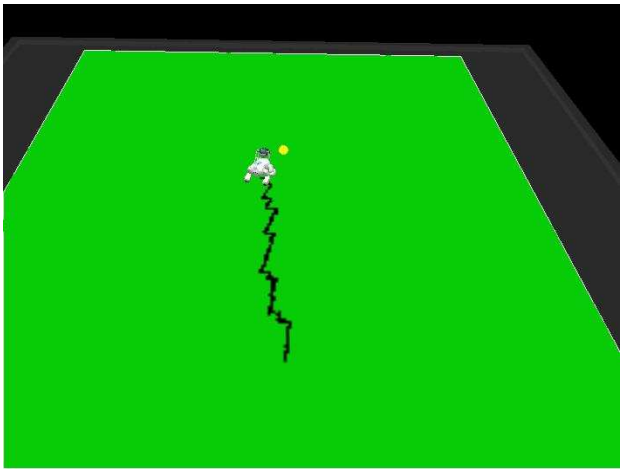

Figure 5.28. ARKAQ Ball Approaching Trajectory 10

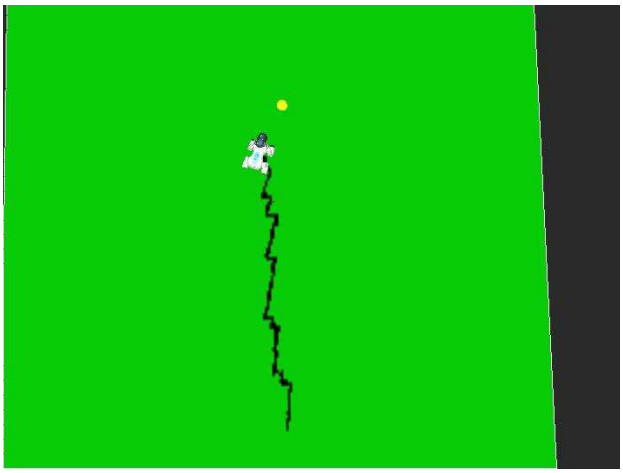

Figure 5.29. ARKAQ Ball Approaching Trajectory 10 (Upper View)

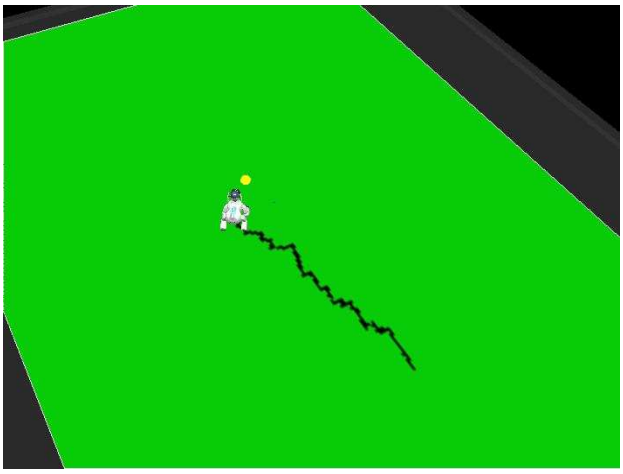

Figure 5.30. ARKAQ Ball Approaching Trajectory 11

5.3. Score a Goal

In this problem, AIBO tries to learn how to score a goal from a specified location. The possible actions are TURN-RIGHT, TURN-LEFT and KICK. The orientation to the goal bar is the environmental input, so our state vector is $\langle orientation \rangle$. The actions TURN-RIGHT or TURN-LEFT simply changes the orientation to the goal bar. After AIBO decides performing the action KICK, the supervisor sets the location of the ball to a suitable position for kicking. The supervisor gives a reward or punishment regarding to the final position of the ball: if its location is within the goal bar AIBO receives +100 reward, otherwise it receives -100 punishment. The following figures are the screenshots of the problem environment.

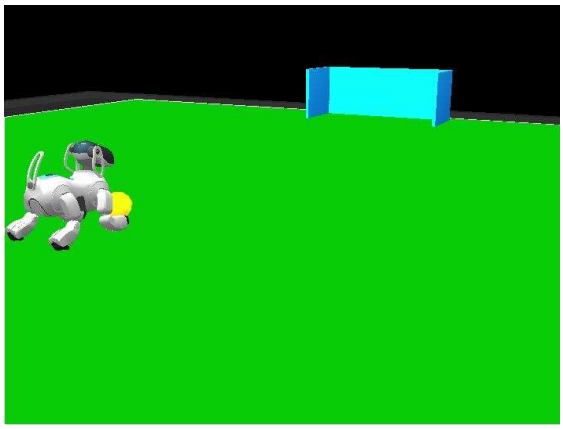

Figure 5.31. AIBO is preparing for a kick

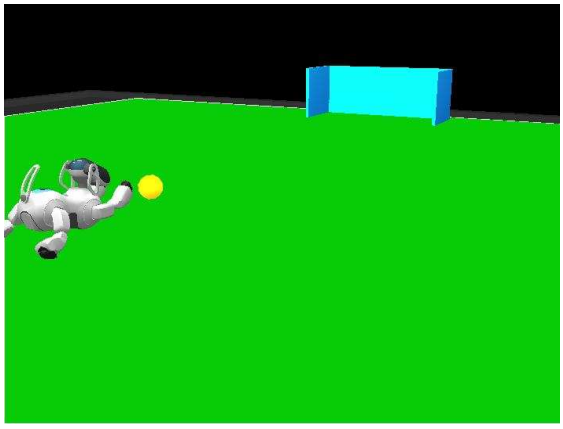

Figure 5.32. AIBO is performing a kick

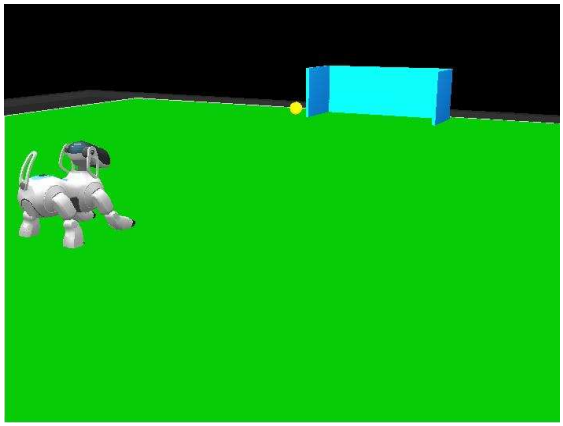

Figure 5.33. AIBO missed the goal chance

5.3.1. State Segmentation

The orientation value to the goal bar is normalized in a similar way to normalization of the orientation value to the ball in the previous problem. State segmentation is achieved with Online K-Means using various vigilance values like in the previous problem. Figure 5.34 demonstrates the segmentation algorithm automaton.

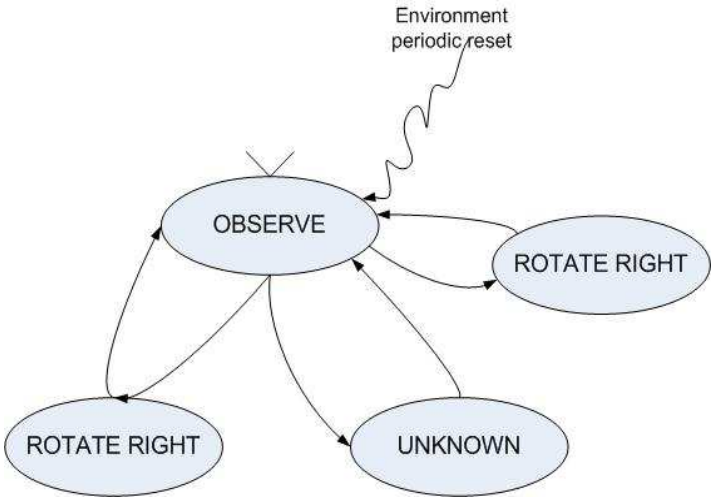

Figure 5.34. ART2 Segmentation finite automaton

State segmentation results are given in the Figure 5.35, Figure 5.36 and Figure 5.37. Since our vector is with one dimension, the results are observed linearly. The dots indicates the cluster centers which can be considered as the orientation to the goal bar. The state is the cluster center which has the closest orientation value compared

with the goal bar.

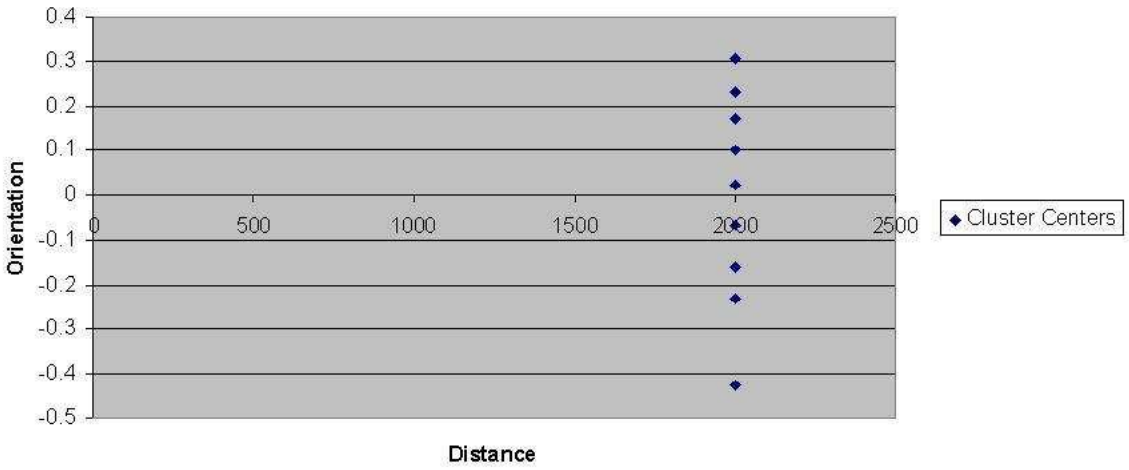

Figure 5.35. ART Network euclidean distance, 9 cluster centers with vigilance 0.075

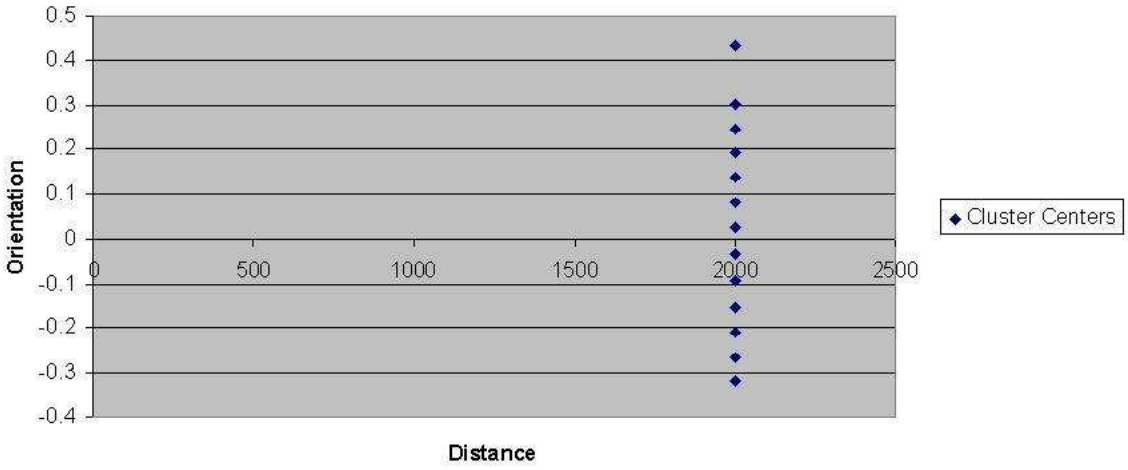

Figure 5.36. ART Network euclidean distance, 13 cluster centers with vigilance 0.06

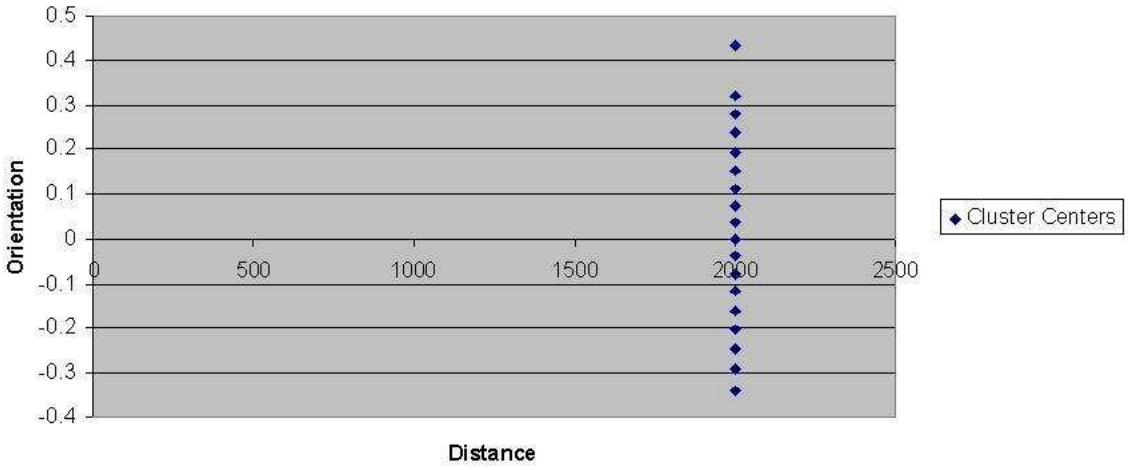

Figure 5.37. ART Network euclidean distance, 18 cluster centers with vigilance 0.05

5.3.2. Q Learning Phase

Q Learning exploration and exploitation phases are realized in a similar way to the previous problem. In the exploration phase, random actions are performed. After an episode, exploitation tries to perform the actions with maximum Q values. Figure 5.38 demonstrates the Q Learning algorithm automaton.

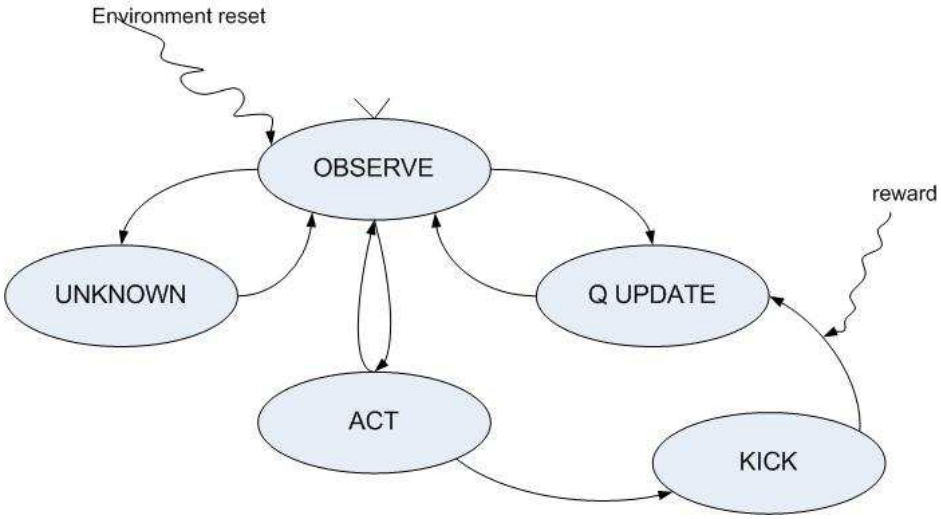

Figure 5.38. ART2 Segmentation finite automaton

Resulting Q Matrix of related problem for the ART Network with 13 cluster centers and vigilance value 0.06 is illustrated in Figure 5.39.

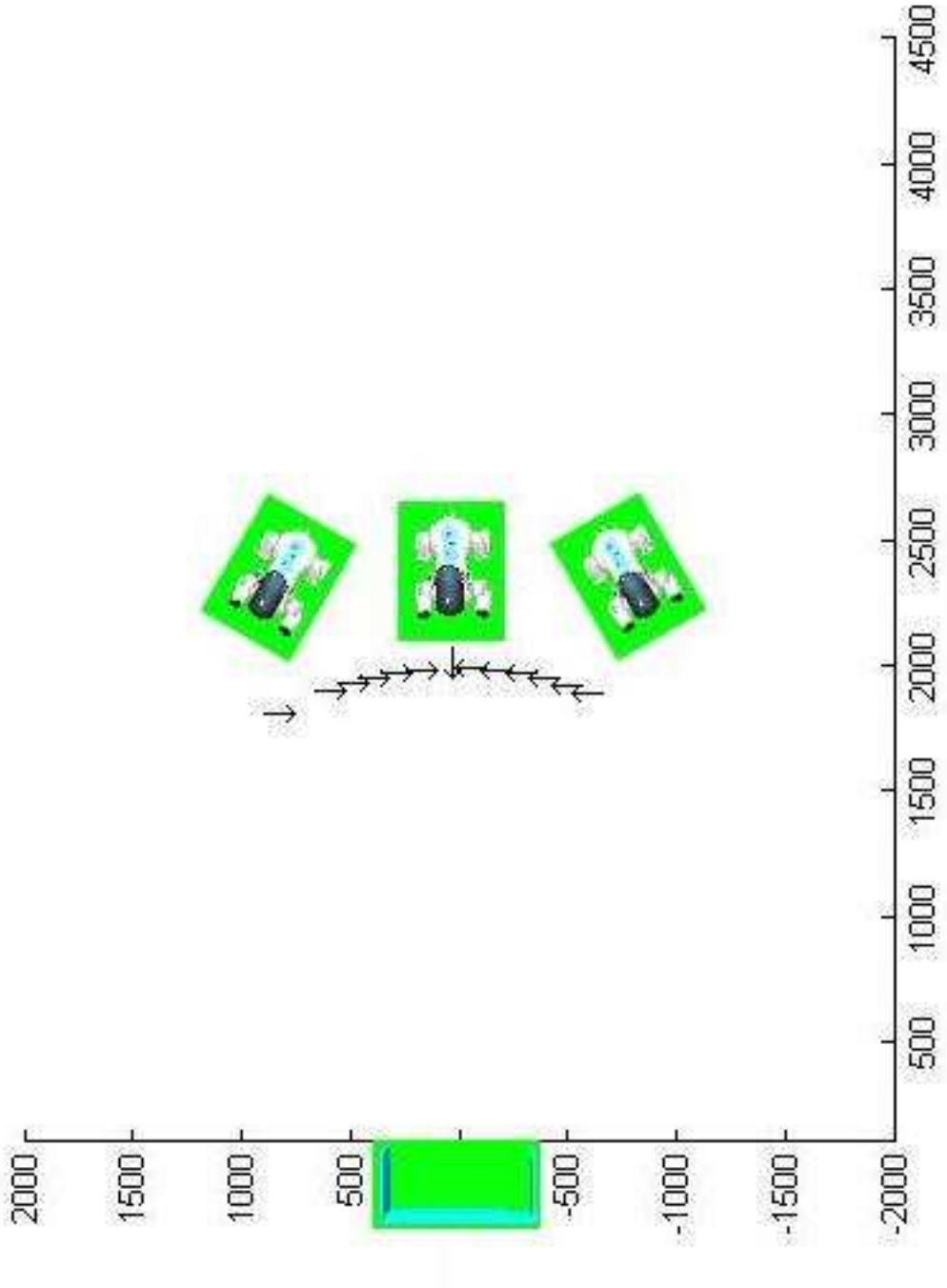

Figure 5.39. Resulting actions inferred from Q values

The comparison in goal scores between the ARKAQ Policy taken actions and randomly taken actions can be examined in Figure 5.40. We can observe that at the beginning the scores are very close, but ARKAQ starts scoring more over time. The success of scores for ARKAQ seems around 60 per cent and for random actions, it is around 35 per cent. The reason of the scores for random action is because the goal bar is reasonably close and even random actions can result with scoring. If we apply the same experiment in a higher distance to the goal bar, the scores for random actions will decrease. Furthermore the kick action does not have high accuracy, we also affect from this in our experiments.

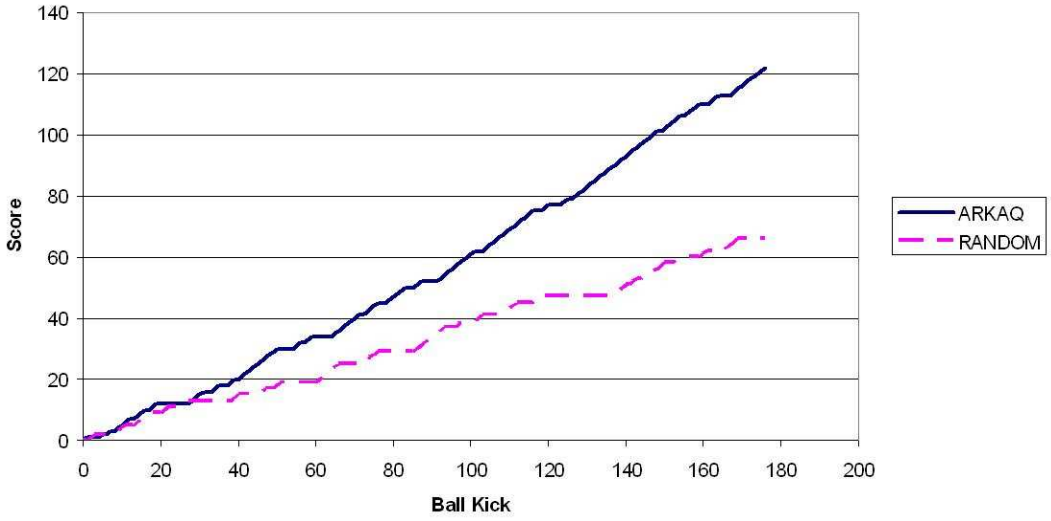

Figure 5.40. Goal scores comparison between ARKAQ and Randomly taken actions

We can observe the success of ARKAQ in Figure 5.41, where the distance to the goal bar is increased, orientation is not straight and with vigilance 0.05.

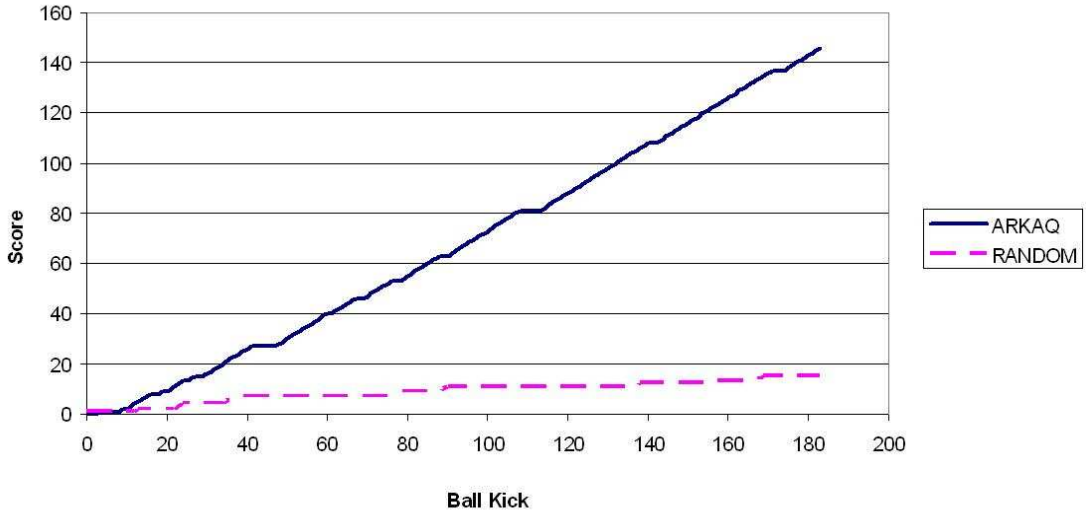

Figure 5.41. Goal scores comparison between ARKAQ and Randomly taken actions in high distance

6. Conclusion

In robotics, the main concentration in planning under uncertainty is the problem of dealing with the inaccurate sensory and actuator environment. This is crucial in order to generate robust policies towards to the goal.

The POMDP framework well defines the problem and there are some propositions on value iteration for policy trees. These techniques proposed are impractical since the time required is not reasonable and also because of the curse of dimensionality in continuous space. Furthermore, the world model is assumed to be given, which is impractical in most cases.

ARKAQ model proposes a solution which is reasonable in time manner. Moreover, the algoritm is online and in worst case has complexity $O(AS^2)$. This model relies on the convergency to the robust policy after an episode.

ARKAQ does not assume the world model to be given. It has a world model generator inside which is capable of segmenting the real-world Markovian states. On the top this world model, ARKAQ has a policy generator which tries to find out the actions which maximizes the discounted sum of rewards to the goal.

ARKAQ algorithm can be applied to a wide range of problems, from path planning to maze domain problems. The success of the algorithm depends on the vigilance parameter which is predefined in world model generator layer.

During the implementation of ARKAQ, various parameter values are applied, dependant to their consequences, the ones that are more suitable to our needs are selected. These parameters are the vigilance for the state segmentation, Q and R matrices for the Kalman filter and convergence parameters for the Q-Learning matrice.

During the state segmentation, for some regions it is observed that the ART

network could not be able to produce cluster centers. This is mostly because the ball cannot be detected on those regions, which can be because of the simulation environment. Beside that, low vigilance parameter caused high number of cluster centers to be produced. This resulted to some unvisited segments which can be because the AIBO performs the actions step by step. The minimum "move forward" action for AIBO means a footstep. So, high number of the cluster centers does not mean better results. The optimum number of cluster centers can be around the multiplication of the number of footsteps in horizontal and in vertical direction. Even though AIBO performs random actions, environmental random reset is necessary for speeding up the segmentation and covering all the field.

Kalman Filter was very useful, even in the simulation environment, because in high distances oscillations up to 100 mm. from distance calculation and 0.05 rad. from orientation calculation of vision engine are observed.

During the Q-Learning phase, the exploration and exploitation phases are decided by manually observing the delta changes in Q values and with Boltzman as well.

As a result, the algorithm produced Q-Learning values that were sufficient to reach the goals. However, the results may be optimised by applying more convenient values for vigilance, Kalman and Q-Learning convergency, dependant to the dynamics of the problem.

REFERENCES

1. M. L. Puterman, "Markov decision processes", *In Handbook in Operations Research and Management Science*, pp. 331-434, 1990.

2. R. Bellman, "A Markovian Decision Process", *Journal of Mathematics and Mechanics*, 6, 1957.

3. R.A. Howard, "A Markovian Decision Process", *Dynamic Programming and Markov Processes*, MIT. Press, 1960.

4. L.P. Kaelbling, M.L. Littman, A.R. Cassandra, "Planning and acting in partially observable stochastic domains", *Artificial Intelligence*, Volume 101, pp. 99-134, 1998.

5. R. Jaulmes, J. Pineau, D. Precup, "Active learning in partially observable Markov decision processes", In *Lecture Notes in Artificial Intelligence*, 3720: pp. 601-608, 2005.

6. S. Grossberg, "Adaptive Resonance Theory", *Technical Report CAS/CNS-2000-024*, September 2000.

7. G. Welch, G. Bishop, "An Introduction to Kalman Filter", *SIGGRAPH 2001*, Course 8 , 2001.

8. P.S. Maybeck, "The Kalman Filter: an Introduction to Concepts", *Stochastic Models, estimation and control*, Volume 1 , pp. 3-7, 1979.

9. C. Watkins, "Learning from Delayed Rewards", *Thesis*, University of Cambridge, 1989.

10. A. Sardağ, H.L. Akın, "ARKAQ-Learning: Autonomous State Space Segmentation and Policy Generation", *Lecture Notes in Computer Science*, Volume 3733, pp. 512-

523, 2005.

11. SONY AIBO, "http://support.sony-europe.com/aibo/", 2006.

12. Cerberus Four-Legged Robot Soccer Team, "http://robot.cmpe.boun.edu.tr/aibo", 2006.

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

14. L.P. Kaelbling, M.L. Littman, A.W. Moore, "Reinforcement Learning: A Survey", *Journal of Artificial Intelligence Research 4*, pp. 237285, 1996.

15. E. Alpaydın, *Introduction to Machine Learning*, MIT Press, 2004.

16. R.S. Sutton, "Learning to predict by the methods of temporal differences", *Machine Learning*, 1988.

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

18. M.E. Harmon, "Reinforcement Learning", *a Tutorial*, Wright State University, 1996.

19. N. Peek, "A specialised POMDP form and algorithm for clinical patient management", *Utrecht University: Information and Computing Sciences*, External research report, 1999.

20. K.J. Aström, "Optimal control of Markov decision processes with incomplete state estimation", *Journal of Mathematical Analysis and Applications*, v10, pp. 174-205, 1965.

21. Anthony R. Cassandra, "Partially Observable Markov Decision Processes", http://www.pomdp.org/, 2005.

22. J. Joyce, "Bayes Theorem", *Stanford Encyclopedia of Philosophy*, 2003.

23. S. Grossberg, "Competitive learning: From interactive activation to adaptive resonance", *Cognitive Science*, 11, pp. 23-63, 1987.

24. G.A. Carpenter, S. Grossberg, "Adaptive Resonance Theory", *In M.A. Arbib (Ed.), The Handbook of Brain Theory and Neural Networks*, Second Edition (pp. 87-90). Cambridge, MA: MIT Press, 2003.

25. K. Gurney, *An Introduction to Neural Networks*, Ucl Press, 1997.

26. G.A. Carpenter, S. Grossberg, "ART 2: Self-organization of stable category recognition codes for analog input patterns", *Applied Optics*, Vol.26 No.23, pp. 4919-4930, 1987.

27. G.A. Carpenter, S. Grossberg, D. Rosen, "ART 2-A: An adaptive resonance algorithm for rapid category learning and recognition", *Neural Networks*, 4, pp. 493-504, 1991.

28. A. Dutech, "Solving POMDPs using selected past events", *Proceedings of the 14th European Conference on Articial Intelligence*, pp. 281-285, 2000.

29. Boost, http://www.boost.org/, 2007.

30. Eclipse IDE, http://www.eclipse.org/, 2007.

31. Webots 5, http://www.cyberbotics.com/, 2006.

32. Cerberus Team Report, http://robot.cmpe.boun.edu.tr/aibo/files/Cerberus06Report.pdf, Feb 13, 2007.