Oppositional Reinforcement Learning with Applications

by

Maryam Shokri

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Systems Design Engineering

Waterloo, Ontario, Canada, 2008

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Maryam Shokri

Abstract

Machine intelligence techniques contribute to solving real-world problems. Reinforcement learning (RL) is one of the machine intelligence techniques with several characteristics that make it suitable for the applications, for which the model of the environment is not available to the agent.

In real-world applications, intelligent agents generally face a very large state space which limits the usability of reinforcement learning. The condition for convergence of reinforcement learning implies that each state-action pair must be visited infinite times, a condition which can be considered impossible to be satisfied in many practical situations.

The goal of this work is to propose a class of new techniques to overcome this problem for off-policy, step-by-step (incremental) and model-free reinforcement learning with discrete state and action space. The focus of this research is using the design characteristics of RL agent to improve its performance regarding the running time while maintaining an acceptable level of accuracy. One way of improving the performance of the intelligent agents is using the model of environment. In this work, a special type of knowledge about the agent actions is employed to improve its performance because in many applications the model of environment may only be known partially or not at all. The concept of opposition is employed in the framework of reinforcement learning to achieve this goal. One of the components of RL agent is the action. For each action we define its associate opposite action. The actions and opposite actions are implemented in the framework of reinforcement learning to update the value function resulting in a faster convergence.

At the beginning of this research the concept of opposition is incorporated in the components of reinforcement learning, states, actions, and reinforcement signal which results in introduction of the oppositional target domain estimation algorithm, OTE. OTE reduces the search and navigation area and accelerates the speed of search for a target. The

OTE algorithm is limited to the applications, in which the model of the environment is provided for the agent. Hence, further investigation is conducted to extend the concept of opposition to the model-free reinforcement learning algorithms. This extension contributes to the generating of several algorithms based on using the concept of opposition for $Q(\lambda)$ technique.

The design of reinforcement learning agent depends on the application. The emphasize of this research is on the characteristics of the actions. Hence, the primary challenge of this work is design and incorporation of the opposite actions in the framework of RL agents. In this research, three different applications, namely grid navigation, elevator control problem, and image thresholding are implemented to address this challenge in context of different applications. The design challenges and some solutions to overcome the problems and improve the algorithms are also investigated. The opposition-based $Q(\lambda)$ algorithms are tested for the applications mentioned earlier. The general idea behind the opposition-based $Q(\lambda)$ algorithms is that in Q-value updating, the agent updates the value of an action in a given state. Hence, if the agent knows the value of opposite action then instead of one value, the agent can update two Q-values at the same time without taking its corresponding opposite action causing an explicit transition to opposite state. If the agent knows both values of action and its opposite action for a given state, then it can update two Q-values. This accelerates the learning process in general and the exploration phase in particular.

Several algorithms are outlined in this work. The $OQ(\lambda)$ will be introduced to accelerate $Q(\lambda)$ algorithm in discrete state spaces. The $NOQ(\lambda)$ method is an extension of $OQ(\lambda)$ to operate in a broader range of non-deterministic environments. The update of the opposition trace in $OQ(\lambda)$ depends on the next state of the opposite action (which generally is not taken by the agent). This limits the usability of this technique to the deterministic environments because the next state should be known to the agent. $NOQ(\lambda)$

will be presented to update the opposition trace independent of knowing the next state for the opposite action. The results show the improvement of the performance in terms of running time for the proposed algorithms comparing to the standard $Q(\lambda)$ technique.

Acknowledgements

I am very grateful for the guidance and supports of my advisers, Dr. H.R. Tizhoosh and Dr. M.S. Kamel.

Professor Tizhoosh has been my supervisor since the beginning of my Master's studies. I am grateful of him for his invaluable guidance, advice and the great discussions we had during my research. His ideas and guidance has formed my way of thinking about this thesis and his mentorship was the paramount advantage that I had through these years. His encouragements always generate a positive and uplifting spirit for me to seriously pursue my goals and overcome the challenges during my research. He is an enthusiastic professor who inspires and empowers by his teaching.

Professor Kamel is the director of our research group, Pattern Analysis and Machine Intelligence (PAMI). In the PAMI lab we always admire him for his strength, leadership, and insights. I was fortunate to pursue my PhD studies under his co-supervision. He is very supportive and his guidance and encouragements have helped me since the beginning of my research. He has a great ability to build a strong and active research team. I have had the privilege of working with him during these years.

I would like to thank the other members of my thesis committee, Dr. Karray, Dr. Basir, Dr. Ponnambalam, and Dr. Berenji who have provided me the important advice and guidance to improve this work.

I would like to sincerely thank Dr. H. Berenji for accepting to be part of my thesis committee. It is a great honor to have him in my committee. I would like to thank him for his time and vital comments to improve this work.

I would like to thank my professors, Dr. Tizhoosh, Dr. Kamel, Dr. Ponnambalam, Dr. Karray, Dr. Cohen, and Dr. Jernigan for the valuable lessons that I learned from their courses during my graduate studies. I am thankful of all PAMI lab professors for the great

opportunity they have provided me to pursue my research.

I would like to thank my friends in PAMI lab and Opposition-Based Learning (OBL) research group for their friendships, discussions, and collaborations. It was a great pleasure for me working with them.

At the end I would like to thank my husband Masoud Khabiry. His love, support, and encouragements have helped me to finish this research. His passion and dedication to the research, his vision and determination always inspire me to continue this journey.

This research is supported by Learning Object Repository Network (LORNET), Ontario Graduate Scholarship (OGS), President's Graduate Scholarship (PGS), and research assistantships.

To my husband Masoud and my parents
for their love, supports, and encouragements

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

Artificial intelligence aims to develop systems that are able to think and act rationally. Currently, there are artificial intelligence methods that are, to different degrees, involved in learning, understanding, adapting, interacting, achieving goals or objectives, reasoning, predicting, recognizing, or acting rationally.

Reinforcement learning (RL) is one of the techniques in artificial intelligence that can be considered a goal-directed method for solving problems in uncertain and dynamic environments. The RL agent learns by receiving reinforcement signals (reward or punishment) from its environment. One of the advantages of using reinforcement learning is its independence from the need to specify a priori knowledge. The learning is rather performed based on trial and error. This behavior is useful for all user-dependent cases where it is difficult to obtain sufficiently large training data.

One of the established approaches of reinforcement learning is temporal difference (TD) which does not require a model of the environment and is based on step-by-step, incre-

mental computation. Q-learning is an off-policy TD control and is one of the most popular methods in reinforcement learning. In off-policy techniques the agent learns a greedy policy and also applies an exploratory policy for action selection. One of the characteristics of Q-learning is model-freedom which makes it suitable for many real-world applications.

Dynamic programming, Monte Carlo algorithms, and temporal-difference learning (TD) are three elementary classes of techniques for solving problems using reinforcement learning. Some of the methods of RL are based on the concept of an eligibility trace which provides a bridge from TD techniques to Monte Carlo methods. The idea is that only eligible states or actions will be assigned a credit or blamed for an error.

1.2 Objective

Most real-world applications constitute large environments which are dynamic, stochastic and/or only partially observable. One of the theoretical conditions for convergence of reinforcement learning methods such as Q-learning [97] implies that each state-action pair must be visited infinite times. Hence, tabular and naive RL algorithms (e.g. Q-learning) benefit from a technique that decreases their computation time in the case of large state spaces.

This research presents the newly proposed opposition-based RL algorithm for accelerating the learning process in off-policy, step by step, incremental and model-free reinforcement learning with discrete state and action space. We integrate the concept of opposition [53, 54, 55, 74, 95] within a new scheme to make some tabular RL algorithms perform better with regard to the running time and number of learning iterations. The general idea behind these algorithms is that in Q-value updating, the agent updates the value of an action in a given state and if it knows the value of the opposite state or the

opposite action, then instead of one value, the agent can update two Q-values at the same time without taking its corresponding opposite action or going to the opposite state. This accelerates the learning process, particularly the exploration stage. The most challenging part of this scheme is defining the opposite state or the opposite action. For some applications, the concept of opposition can be established heuristically. A variety of algorithms can be generated based on the concept of opposition to improve learning and facilitate a faster convergence [73, 90, 91, 92]. The objective of this research is to increase the speed of some of RL-techniques using the concept of opposition by maintaining the same accuracy.

1.3 Thesis Organization

The thesis consists of six chapters. Chapter 2 introduces the reinforcement learning techniques as well as a survey of the techniques for accelerating reinforcement learning. Chapter 2 also describes the opposition-based computing. The purpose of this research is to introduce a technique to accelerate the reinforcement learning methods for off-policy, step-by-step (incremental) and model-free reinforcement learning with discrete state and action space. Hence, the design characteristics of RL agent as well as the concept of opposition are employed to improve the performance of some of the RL algorithm regarding the running time.

At the beginning of this research (Chapter 3) the concept of opposition is applied to the components of reinforcement learning, namely states, actions, and reinforcement signal which yields to the introducing the oppositional target domain estimation (OTE) algorithm. OTE is not constructed upon learning, it rather searches for a reduced state space which includes the target. However, the model of the environment should be provided for the OTE algorithm.

In order to extend the opposition-based computing to the model-free RL techniques a special type of knowledge about the agent actions is used. Therefore, in Chapter 4 a class of new opposition-based RL algorithms is introduced by employing opposite actions to update the value function. Major techniques based on opposition are described in this chapter.

Chapter 5 investigates the oppositional agents in dynamic environments for more challenging applications. Three major applications, namely navigation, elevator control, and image thresholding are discussed based on the concept of opposition and the design issues for implementing this concept is discussed in details. Chapter 6 contains the conclusions with some thoughts on the future works.

Chapter 2

Reinforcement Learning and Opposition-Based Computing

2.1 Reinforcement Learning

Reinforcement learning (RL) can be considered as a class of goal-directed intelligent techniques for solving problems in uncertain and dynamic environments. By definition, an agent is "something that acts" or "something that perceives and acts in an environment" [66]. More specifically by *intelligent agent* we mean an intelligent software that performs a task for a human user. A rational agent attempts in a way to maximize the expected value of a performance/quality measure, given the percept sequence it has seen so far [66]. Agents should have the ability to manage themselves, optimize task performance and increase the level of security. An agent can be a function or a piece of software that maps the states of an environment to a set of actions. A rational agent aims to maximize the profit according to a performance measure. Intelligent robots and softbots (software agents) are examples of artificial intelligent agents.

A reinforcement learning agent is autonomous [58] meaning that its behavior is determined by its own experience. What is outside the agent is considered the environment. The states are parameters (features) describing the environment. An RL agent has the ability to sense the environment and learn the optimal policy (or a good policy) for taking optimal/good actions in each state of the environment to achieve its goal. As a result of taking an action, the agent has the ability to influence the state of the environment (and can map the states to appropriate actions). The agent must be aware of the states of the environment by interacting with the environment, and learns from receiving reinforcement feedback as reward or punishment from its environment. The reward function represents, directly or indirectly, the goal of the reinforcement learning problem. RL agents try to maximize the reward or minimize the punishment [3, 16, 83]. The reward values could be objective or subjective. In the subjective case, the agent will receive reward and punishment directly from the interactive user (in some cases by using an interface). In the objective case, the reward is defined based on some optimality measures or desired properties of the results [66, 69]. Actions could affect the next situation and subsequent rewards and have the ability to optimize the environment's state [31].

As mentioned earlier, one of the advantages of using reinforcement learning is independency of some of the RL techniques from a priori knowledge by learning based on trial and error. Some RL agents learn from their own experience without relying on a teacher or training data. This kind of learning is not supervised, but because of using a reward function and exploitation of rewarding actions a weak supervision can be assumed. The agent can learn online by the ability of continuously learning and adapting through interaction with the environment while performing the required task and improving its behavior in real time. This characteristic is useful, among others, for all user-dependent cases where a set of sufficiently large training data is difficult or impossible to obtain.

Learning is the core characteristic of any intelligent system. Learning can be described as "modification of a behavioral tendency by experience" [44]. For an RL agent, learning by trial and error has mainly two stages, namely exploration and exploitation. Exploration means that the agent tries to discover which actions yield the maximum reward by taking different actions repeatedly. Exploitation, on the other hand, means that the agent takes those actions that yield more reward.

The history of RL research has two major parts: the study of animal learning, and the solution of optimal control problems using value functions and dynamic programming [83]. Value functions are functions of states or functions of state-action pairs that estimate how good it is to perform a given action in a given state. Watkins developed the Q-learning algorithm in 1989 [96, 97] in such a way that the agent maintains a value for both state and action, which represents a prediction of the value of taking that action in that state.

2.1.1 General Framework

The design of an RL agent is based on the characteristics of the problem at hand. First of all the problem must be clearly defined and analyzed and the purpose of designing the agent must be determined. Figure 2.1 illustrates the components, which constitute the general idea behind reinforcement learning.

The RL agent, which is the decision-maker in the process, takes an action that influences the environment. The agent acquires knowledge of the actions that generate rewards and punishments and eventually learns to perform the actions that are the most rewarding in order to attain a certain goal. In the RL model presented in Figure 2.1 the process is as follows [58]:

- Observe the state of the environment

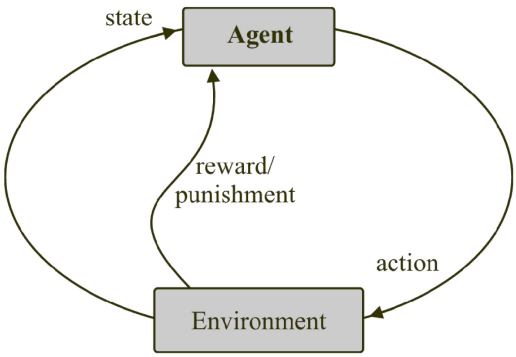

Figure 2.1: Basic components of reinforcement learning

- Take an action and observe reward and punishment
- Observe the new state

2.1.2 Action Policy

Another key element of reinforcement learning is the action policy which defines the agent's behavior at any given time. It maps the perceived states to the actions to be taken [83]. There are three common policies, softmax, ϵ-greedy, and greedy. ϵ-greedy or near-greedy policy is based on the idea of selecting greedy actions most of the time but a few times selecting a random action with probability of ϵ. Based on the near-greedy policy an agent has a chance to explore more and balance exploration with exploitation, in contrast with greedy policy that always selects rewarding actions. In this situation all actions may not be explored. Generally, choosing the appropriate policy depends on the application but it must be considered that the greedy policy sometimes leads to sub-optimal situations which is a common problem in any intelligent system. On the other hand, one of the disadvantages of using ϵ-greedy is that this technique uses random action

selection by considering equal probabilities for each action based on a uniform distribution for the exploration. An alternative solution is considering the estimated value of each action to vary the probability of action selection. This technique is called softmax action selection [83]. The Boltzmann policy is the most common softmax method, and uses a Gibbs, or Boltzmann distribution for defining the policy. The probability of taking each action is presented based on Boltzmann policy in Equation 2.1, where $\mathrm{Q}(s, a)$ is the Q-matrix [96, 97] representing the state-action values, τ is a parameter called temperature, s is the state, and a is a candidate action.

$$P(a) = \frac{e^{\frac{Q(s,a)}{\tau}}}{\sum e^{\frac{Q(s,:)}{\tau}}}. \tag{2.1}$$

In the Equation 2.1 the parameter τ is used to control the intensity of exploration. At the beginning of learning τ has higher values which contributes to more exploration. By increasing the time τ decreases and consequently the intensity of exploitation increases [26].

2.1.3 Markov Decision Process (MDP)

Reinforcement learning is learning from interaction of the agent with the environment to achieve the goal, i.e. maximizing the accumulated rewards over the long run [31]. Another concept that should be mentioned here is the Markov property for reinforcement learning. If the environment has the Markov property, then the state at the time step $t+1$ depends on the state and action at time t. If the reinforcement learning task satisfies the Markov property it is called a finite Markov decision process (MDP) [83]. Regarding RL problems as MDPs, it is assumed that the next state depends on the finite history of previous states [66]. The environment can be modeled as a Markov decision process (MDP). The

first order Markov property is used to predict the probability of a possible next state as follows:

$$P^a_{ss'} = Pr\{s_{t+1}|s_t = s, a_t = a\}, \tag{2.2}$$

where $P^a_{ss'}$ is the transition probability to the next state $s' = s_{t+1}$ given any state s and action a [83].

2.1.4 Different RL Schemes

The solution of the reinforcement learning problem is the policy that maximizes the reward over several learning episodes. There are three elementary classes of techniques for solving reinforcement learning problems: dynamic programming, Monte Carlo algorithm, and temporal-difference learning (TD). There are also methods based on "eligibility traces" that can be considered as a bridge between TD methods and Monte Carlo techniques. The idea of eligibility traces is that only eligible states or actions will be assigned a credit or blame for the error [83].

Optimization techniques are suitable for solving the sequential decision problems when knowledge of the environment is provided for the agent. The model of the environment can be applied to predict the next state and next reward by using the current state and action. Models generally can be employed for planning by considering possible future situations before they occur to decide the best possible action. A model, similar to integration of prior knowledge, can include the transition probabilities for the states of the environment and the expected reward for the actions.

Dynamic programming is one of the techniques for solving reinforcement learning problems if a complete model of the environment is available. The model of environment can

be applied to predict the next state and next reward by using the given state and action. Dynamic programming is a mathematically well developed method but the complete and accurate model of the environment must be provided for this technique.

Monte Carlo techniques are not based on using the model of the environment. However, they are not appropriate for step-by-step learning because of episodic nature [83].

Another alternative approach for solving a reinforcement learning problem is temporal difference (TD) methods which do not require a model of the environment and are based on step-by-step incremental computation [31]. The TD technique is flexible because it does not require a model. However its implementation is more complex.

Another property of some of the elementary RL techniques is bootstrapping, which refers to the idea of updating estimated values of states based on estimated values of successor states without waiting for final outcomes [2]. Dynamic programming (DP) and TD are bootstrapping methods, in contrast to Monte Carlo methods which wait for a final outcome. In this research, the focus is on TD approaches.

2.1.5 Temporal-Difference Learning

Temporal-difference learning is a combination of Monte Carlo and dynamic programming ideas. If at time t a nonterminal state s_t is visited, TD methods estimate the value of that state, $V(s_t)$, based on what happens after that visit. TD methods wait until the next step $(t+1)$ to determine the increment to $V(s_t)$ as opposed to Monte Carlo that must wait until the end of the learning episode. The simplest TD method, called $TD(0)$, is presented in Equation 2.3 where α is a step-size parameter ($0 < \alpha \leq 1$), r is the result[1] of taking an action, γ is a discount-rate parameter ($0 \leq \gamma \leq 1$) and $V(s_t)$ is the value function in each

[1]Generally, in all RL algorithms r represents the result of the action which can be reward or punishment. The variable r represents the reward and a variable p represents the punishment.

time step t:

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]. \tag{2.3}$$

The tabular $TD(0)$ for estimating the value function is presented in Algorithm 1 [83] where V is the value function and π is a policy that must be evaluated.

Algorithm 1 Tabular $TD(0)$ for estimating value function

```
Initialize V(s) arbitrary, π the policy to be evaluated
for each episode do
  Initialize s
  for each step of episode do
    a ← action given by π for s
    Take action a: observe reward, r, and next state, s'
    V(s) ← V(s) + α[r + γV(s') − V(s)]
    s ← s'
  end for
end for
```

Sarsa

Sarsa (State-Action-Reward-State-Action) is an on-policy method and is one of the TD techniques. For an on-policy method the state-action value $Q^{\pi}(s, a)$ must be estimated for the current policy π, and all states s and actions a. In the on-policy Sarsa, the learned policy for the action-value function is the same as the policy which is applied for action selection. The general steps of Sarsa are presented in Algorithm 2 [83].

Q-Learning

Q-learning is off-policy TD control and one of the most popular methods in reinforcement learning. In an off-policy technique the learned action-value function, $Q(s, a)$, directly ap-

Algorithm 2 Sarsa: An on-policy TD control algorithm

```
Initialize Q(s, a) arbitrary
for each episode do
  Initialize s
  Choose a from s using policy derived from Q (e.g.,ε-greedy)
  for each step of episode do
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ε-greedy)
    Q(s, a) ← Q(s, a) + α[r + γQ(s', a') − Q(s, a)]
    s ← s', a ← a'
  end for
end for
```

proximates the optimal action-value function Q^* independent of the policy being followed. In other words, the agent learns a greedy policy as well as applying exploratory policy for action selection [83]. It simplifies the algorithm and facilitates convergence. The agent learns to act optimally in Markovian domains by experiencing sequences of actions. The agent takes an action at a particular state and uses immediate reward and punishment and estimates the state value. By trying all actions in all states multiple times, the agent learns which action is best overall for each visited state [97]. Therefore, the agent must determine an optimal policy and maximize the total discounted expected reward. Equation 2.4 must be employed for updating the action-value function, Q, where s is state, a is action, α is learning step, γ is discount factor, r is the reward (reinforcement signal), a' is the next action, and s' is the next state:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]. \tag{2.4}$$

The task of Q-learning is determining an optimal policy π^*. The values of the Q matrix are the expected discounted reward for executing action a at state s, using policy π [97].

The (theoretical) condition for convergence of the Q-learning algorithm is that the sequence of episodes which forms the basis of learning must visit all states infinitely. It must be mentioned that based on Watkins' and Dayan's Theorem in [97] the rewards and learning rate are bounded ($|r_n| \leq R_{max}$, $0 \leq \alpha_n < 1$). The Q-learning algorithm is presented in Algorithm 3 [83].

Algorithm 3 Q-Learning: An off-policy TD control algorithm

```
Initialize Q(s, a) arbitrary
for each episode do
  Initialize s
  for each step of episode do
    Choose a from s using policy derived from Q {e.g., ε-greedy}
    Take action a, observe r, s'
    Q(s, a) ← Q(s, a) + α[r + γ max_a' Q(s', a') − Q(s, a)]
    s ← s';
  end for
end for
```

Q-learning is model-free and off-policy, and it uses bootstrapping. The bootstrapping property helps the algorithm to use step-by-step reward, which is the result of the taking an action in each step. This reward is a useful source of information which guides the agent through the environment. Q-learning is considered a general and simple technique for learning through interaction in unknown environments.

Initialization of parameters can be considered as a priori knowledge for reinforcement learning. Choosing the appropriate parameters may influence the convergence rate [51, 83]. The (theoretical) condition for convergence of the Q-algorithm is that the sequence of episodes which forms the basis of learning must visit all states infinitely. It must be mentioned that based on Watkins and Dayan's Theorem in [97] the rewards and learning rate are bounded ($|r_n| \leq R_{max}$, $0 \leq \alpha_n < 1$). The focus of opposition-based reinforcement learning is on accelerating the speed of the updating process for Q-learning which is a

major problem in real-world applications and a result of the hyper-dimensionality of states and actions.

Q(λ): A Bridge Between Monte Carlo and Q-Learning

Sutton et al. [83] introduced the eligibility trace as a bridge between TD techniques and Monte Carlo methods. The idea behind the eligibility traces is that only eligible states or actions will be assigned a credit or blamed for the error. $TD(\lambda)$ is based on a weighted averaging of n-step backups, λ^{n-1} $(0 \leq \lambda \leq 1)$. This weighted averaging is presented in Equation 2.5 and Equation 2.6 (λ-return) where R_t^n is the n-step target at time t [83],

$$R_t^n = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... + \gamma^{n-1} r_{t+n} + \gamma^n V_t(s_{t+n}), \tag{2.5}$$

$$R_t^\lambda = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} R_t^n. \tag{2.6}$$

The alternative choice for non-Markovian tasks with long delayed rewards is using the eligibility traces. "Eligibility trace is a temporary record of occurance of an event" and "marks the memory parameter associated with the event as eligible for undergoing learning changes" [83]. Eligibility traces require more computation but yield faster learning especially for applications with delayed rewards (result of many steps) [83]. The parameter λ $(0 \leq \lambda \leq 1)$ must be adjusted to place the eligibility somewhere between TD and Monte Carlo. If $\lambda = 1$ then the algorithm behaves like the Monte Carlo technique. In contrast, if $\lambda = 0$, then the overall backups (for value V) behave like one-step TD backup. Sutton et al. suggest that there is not sufficient theoretical investigations on determining the suitable location for placing the eligibility [83].

The idea of eligibility traces can be applied to TD techniques such as Sarsa and Q-

learning. Watkins' $Q(\lambda)$ unlike the $TD(\lambda)$ and Sarsa(λ) looks ahead until the next exploratory action. $TD(\lambda)$ and Sarsa(λ) need to look ahead until the end of the episode. In contrast, in $Q(\lambda)$, if an agent takes an exploratory action then the eligibility traces will become zero. During the exploration process in early learning the traces will be cut off [83]. Peng et al. [49] introduced a solution for this problem by using a mixture of backups. The implementation of Peng's technique is more difficult than the $Q(\lambda)$ [49, 83]. It is based on a combination of $TD(\lambda)$ and Q-learning. This technique is "experimentation-sensitive" when $\lambda > 0$. Rewards associated with "non-greedy action" will not be used for evaluating "greedy policy" [49].

Sutton et al. proposed a third variation of $Q(\lambda)$ [83], Naive $Q(\lambda)$, which is like Watkins' $Q(\lambda)$ with the difference that in this version the traces related to exploratory actions are not set to zero. In this chapter the Watkins' Q(λ) has been implemented (presented in Algorithm 4). The reason for this is that in Watkins' Q(λ) the representation of the update for the value function is based on using $Q(s, a)$, contrary to Peng's technique which is based on a combination of backups based on both $V(s)$ and $Q(s, a)$. Since the opposition-based RL technique benefits from the concept of opposite actions, therefore the updates should be presented for $Q(s, a)$ and $Q(s, \breve{a})$ where $\breve{a}$ denotes the opposite action.

2.1.6 Benefits and Shortcomings

Reinforcement learning has the ability of learning through interaction with a dynamic environment and using reward and punishment independent of any training data set. One of the advantages of using some of the reinforcement learning techniques is independency from a priori knowledge. RL agents learn from their own experience without relying on teachers or training data. The agent does not need a set of training examples. Instead, it learns online by continuously adapting its knowledge through interaction with the environ-

Algorithm 4 Tabular Watkins' Q(λ) algorithm [83]

Initialize $Q(s, a)$ with arbitrary numbers and initialize $e(s, a) = 0$ for all s and a
for each episode **do**
  Initialize s and a
  for each step of episode **do**
    Take action a, observe r and next state s'
    Choose next action a' from s' using policy
    $a^* \longleftarrow argmax_b$ Q(s', b) {if a' ties for max, then $a^* \leftarrow a'$}
    $\delta \longleftarrow r + \gamma$ Q(s', a^*)−Q(s, a)
    $e(s, a) \longleftarrow e(s, a) + 1$
    for all s, a **do**
      Q$(s, a) \longleftarrow$ Q$(s, a) + \alpha\delta e(s, a)$
      if $a' = a^*$ **then**
        $e(s, a) \longleftarrow \gamma\lambda e(s, a)$
      else
        $e(s, a) \longleftarrow 0$
      end if
    end for
    $s \longleftarrow s'$;$a \longleftarrow a'$
  end for
end for

ment while performing the required task and improving its behavior in real time. In some of the RL techniques the model of the environment can be applied to predict the next state and next reward by using the given state and action. Model-based reinforcement learning methods have the advantage of yielding more accurate value estimation but they require more storage and may suffer from the curse of dimensionality. The other major problem is that the model of the environment is not always available. The strengths and shortcomings of RL techniques can be summarized as follows [30, 37, 50, 75]:

Strengths:

- Online learning through interaction with environment
- No training data is required for some of the RL techniques
- No model is required for some of the RL techniques
- Independency of a priori knowledge for some of the RL techniques

Shortcomings:

- Exploring the state space is computationally expensive for some of the RL techniques specially tabular methods
- The large number of actions also makes the RL techniques more computationally expensive
- Design of RL agents is not straightforward for all applications

The condition for convergence of Q-learning implies that each state-action pair must be visited infinite times. This requirement is impossible to fulfill in the real-world, but in

practice nearly optimal action policies can be achieved before all actions have been taken several times for all states [57]. In real-world applications generally the agent faces a very large state or action space which limits the usability of reinforcement learning. There are several solutions in literature. Some general solutions to this problem are briefly described in following.

- Using a priori knowledge - It is argued that RL is not always a tabula rasa and additional knowledge can be provided for the agent [41, 57]. The knowledge can be gained through imitation of other agents or transformation of knowledge from previously solved problems [41]. Ribeiro proposed a technique of using knowledge about rates of variation for action values in Q-learning and updating temporal information for the visited states and neighboring states which have similarities with the visited states [57]. Knowledge can be transfered to the agent by a human user through giving advice to the agent. Maclin et al. [40] propose the Preference Knowledge-Based Kernel Regression algorithm (Pref-KBKR). Their algorithm uses human advice as a policy based on if-then rules.

- Using hierarchical models and task decomposition - The idea of a hierarchical model is presented by Mahadevan and Kaelbling in [41] as a way of increasing the speed of learning by decomposing the task into a collection of simpler subtasks. Goel provides a technique of sub-goal discovery based on learned policy for hierarchical reinforcement learning [18]. In this technique, hierarchies of actions are produced by using sub-goals in a learned policy model. Hierarchies of actions can then be applied for more effective exploration and acceleration of the process [41]. Goel emphasizes that for finding the sub-goals the states with certain structural properties

must be searched [41]. Kaelbling [29] introduces hierarchical distance to goal learning (HDG) by using decomposition of the state space. She argues that this modified version achieves the goal more efficiently. Shapiro et al. [68] combine the hierarchical RL with background knowledge. They propose Icarus, an agent architecture that embeds hierarchical RL within a programming language representing an agent's behavior, where the programmer writes an approximately correct plan including options in different level of details to show how to behave with different options. The agent then learns the best options from experience and the reward function given by the user.

- Parameters optimization - Potapov and Ali mention that choosing the appropriate parameters may influence convergence rate of reinforcement learning [51]. They investigate the problem of selecting parameters for Q-learning method.

- Function approximation - Function approximation techniques such as neural networks can also be considered as a technique for reducing the large stochastic state space, especially in continuous domains [41, 84]. The function approximation techniques are providing solutions for approximating the value function [41].

- Offline training - The RL agent could be trained offline. The demands and needs of a user can be learned offline in order to minimize the online learning time [88].

- Gereralization - Agent generalizes by learning similar or closely related tasks. The generalization techniques "allow compact storage of learned information and transfer

of knowledge between 'similar' states and actions" [31].

- Hybrid techniques - As mentioned earlier, neural networks can be applied in the framework of the RL as function approximators [41, 84]. In the case of model learning the Bayesian network can represent state transition and reward function [41].

- Fuzzy techniques - Berenji [6] introduces fuzzy Q-learning (FQ-learning) which can be applied in the decision process with fuzzy goals or constraints. FQ-learning is an extension of Q-learning applicable for fuzzy environments. Berenji [7] introduces the combination of the fuzzy Q-learning and the generalized approximate reasoning-based intelligent control ($GARIC$) for generalizing reinforcement learning. In this technique a group of agents are controlled by fuzzy Q-learning "at top level" but each agent learns locally based on $GARIC$ architecture. Hence, the "generalization of input space" by using "fuzzy rules" expedites and improves fuzzy Q-learning in the $GARIC - Q$ method.

- Using macro actions - Mc Govern and Sutton [42] introduce the macro actions as temporally extended actions by combining smaller actions. They argue that macro actions can affect the speed of learning based on the task at hand.

- Relational state abstraction - Morales proposes relational state abstraction for reinforcement learning [46]. In this technique Morales describes that the states can be presented as a set of relational properties which yields to abstraction and simplification of the state space. Then, the agent can learn over abstracted space to

produce reusable policies which can be transferred and applied to similar problems or problems with the same relational properties [46].

Relational learning can be considered as "learning from examples and background knowledge" and can be presented using inductive logic [14]. Driessens et al. describe the problem of "integrating guidance and experimentation in reinforcement learning", particularly relational RL technique [14].

As it is described, a large number of techniques exist in the literature and many of them are involved in using function approximation techniques or integration of knowledge of the environment. In this research the novel idea of opposition will be introduced to expedite the RL-techniques. The idea of explicitly employing opposition in machine intelligence has been introduced before [91].

There are "Diverse forms of opposition" everywhere around us but the "nature and significance of oppositeness" are not investigated and applied directly in the field of computational intelligence [93]. The goal of this research is understanding and establishing the concept of opposition within reinforcement leaning techniques in order to improve the performance of existing methods in terms of running time.

Another emphasis of this research is to provide a new class of RL-algorithms to perform faster than their conventional techniques and meanwhile keep them independent from any training data or model of environment. Hence, in the opposition-based RL approach the new form of knowledge which is associated to the actions are employed. The new approach can be further developed and combined with other techniques (mentioned earlier) which is a subject for future studies. In the next section the concept of opposition and its connection with learning techniques will be described.

2.2 Opposition-Based Computing

2.2.1 Introduction

Different forms of opposition are apparent in nature and have been studied in some of the fields such as philosophy, psychology, linguistics, logic and physics. Generally, the word opposite "is used to describe things of the same kind which are completely different in a particular way."[12] Two directions of west and east are opposite of each other. Two different results of winning and losing a game are opposite of each other. In a social context, the opposition means "strong, angry or violent disagreement and disapproval" [12]. The opposition influences the environment by using sudden and radical changes. Even though the opposition is used in the fields of engineering, mathematics, and computer science, it is not usually studied in a direct way. Systematical examination and understanding of opposition could yield a better convergence rate, new search or optimization algorithms, and a new approach to reasoning and decision making. In this research the concept of opposition is used to expedite existing RL algorithms. Opposition-based learning (OBL) has been recently introduced to accelerate learning and search in model-free contexts [53, 54, 55, 73, 74, 90, 91, 92, 95].

Consider the function $y = f(s_1, s_2, \ldots, s_n)$. The task could be to approximate f, solve $f(\cdot) = 0$, or develop a policy π that describes the relationship between inputs and outputs according to $y = f(\cdot)$. In machine intelligence, the function f itself is generally unknown. Different methodologies are employed for approximation (e.g. neural nets), optimization (e.g. evolutionary algorithms) and search for an optimal policy (e.g. reinforcement learning). In all cases, random initialization can be applied at the beginning of learning or search process when prior knowledge is not available. Random initialization of weights of neural nets, chromosomes in genetic algorithms, and state-action values in reinforcement learning

are the examples of these cases. A random guess such as $S = (s_1, s_2, \cdots, s_n)$ may be close to or far from the solution vector like $S_q = (s_{q1}, s_{q2}, \cdots, s_{qn})$. If the initial random guess is far from the existing solution[2] then the learning or search time considerably increases. It should be emphasized that the absence of prior knowledge for diverse machine intelligence problems reduces the chance of the best initial guess and makes the machine intelligence techniques less efficient and time consuming.

2.2.2 Basic Definitions

The definition of opposition may be diverse for different problems of engineering and computer science and there is a need to establish at least to some degree mathematical definitions based on the main idea of opposition which is as follows: If we initialize an algorithm with a random guess $S = (s_1, s_2, \cdots, s_n)$, or if we are operating on a candidate solution $S = (s_1, s_2, \cdots, s_n)$, which has been initially a random number, it could be advantageous to simultaneously search in the opposite direction yielding to calculate the *opposite guess* $\breve{S} = (\breve{s}_1, \breve{s}_2, \cdots, \breve{s}_n)$ [90, 91, 92, 95, 93]. In this section the abstract definitions of opposition are introduced [93].

> **Definition of Type-I Opposition Mapping:** Any one-to-one mapping function, $\phi : C \rightarrow C$, which defines an oppositional relationship between two unique elements $C1$, $C2$ of concept class C is a type-I opposition mapping. Furthermore, the relationship is symmetric in that if $\phi(C_1) = C_2$, then $\phi(C_2) = C_1$.

The opposite nature between C1 and C2 is defined by this mapping. The opposite nature between C1,C2 could be understood by considering the problem and/or goal of using opposition (i.e. minimizing an evaluation function).

[2] In the worst-case situation the initial random guess may be located at the *opposite position*

> **Definition of Type-I Opposition:** Let $c \in C$ be a concept in n-dimensional space and let $\phi : C \rightarrow C$ be an opposition mapping function. Then, the type-I opposite concept is determined by $\breve{c} = \phi(c)$.

The "explicit reliance on $\phi(C)$" can be omitted for convenience [93]. Therefore, in type-I opposition if $\breve{x}_i = \phi(x_i) = a_i + b_i - x_i$, it can be written as $\breve{x}_i = a_i + b_i - x_i$, with $x_i \in [a_i, b_i]$. The type-I opposite can be denoted as $\breve{x}_i = -x_i$ or $\breve{x}_i = 1 - x_i$ depending on the range of the universe discourse.

> **Definition of Type-II Opposition Mapping:** Any one-to-many function $\Psi : f(C) \rightarrow f(C)$ which defines an oppositional relationship between the evaluation f of a concept $c \in C$ to a set S of all other evaluations of concepts also in C such that $c \in \Psi(s) \forall s \in S$.

"Given some concept c and its associated evaluation $f(c)$, the type-II opposition mapping will return a set of opposite evaluations" [93].

> **Definition of Type-II Opposition:** Let concept c be in set C and let $\Psi : f(C) \rightarrow f(C)$ be a type-II opposition mapping where f is a performance function (such as error, cost, fitness, reward, etc.) Then, the set of type-II opposites of c are completely defined by Ψ. Type-II opposition can also be considered as non-linear opposition.

"Type-II opposition mappings can be approximated online as the learning/search is in progress" [93]. In this situation the opposition can be acquired via online mining.

> **Definition of Opposition Mining:** Let $c^* \in C$ be a target concept and Ψ a type-II opposition mapping. Then, opposition mining refers to the online discovery of $\breve{\Psi}$, which represents an approximation to Ψ.

Definition of Degree of Opposition: Let ϕ be a type-I opposition mapping and $c_1, c_2 \in C$ arbitrary concepts such that $c_1 \neq \phi(c_2)$. Then, the relationship between c_1 and c_2 is not opposite, but can be described as partial opposition, determined by a function $\tau : c_1, c_2 \rightarrow [0, 1]$. As $\tau(c_1, c_2)$ approaches 1 the two concepts are closer to being opposite of each other.

Tizhoosh [92] uses the opposition-based mining to define a degree of opposition to show how far two actions are opposite of each other. His degree of opposition is defined as

$$\tau(a_1|_{s_t}, a_2|_{s_j}) = \eta \times \left[1 - \exp(-\frac{|Q(s_i, a_1) - Q(s_j, a_2)|}{\max\limits_k (Q(s_i, a_k), Q(s_j, a_k))})\right], \tag{2.7}$$

where the Q is a matrix of accumulated discounted rewards, and η is the state similarity which is calculated based on state clustering or a simple measure presented:

$$\eta(s_i, s_j) = 1 - \frac{\sum\limits_k |Q(s_i, a_k) - Q(s_j, a_k)|}{\sum\limits_k \max(Q(s_i, a_k), Q(s_j, a_k))}. \tag{2.8}$$

The online opposition mining based on this measure has not been fully investigated for reinforcement learning algorithm [92].

Opposition-Based Computing: We speak of opposition-based computing, when a computational method or technique implicitly or explicitly employs oppositional relations and attributes either at its foundation (purely oppositional algorithms) or to improve existing behavior of some parent algorithm (opposition based extensions of existing algorithms).

Opposition can be embedded into existing algorithms implicitly or explicitly in order to improve the performance of the algorithms.

> **Implicit OBC Algorithms:** Any algorithm that incorporates oppositional concepts without explicitly using type-I or type-II opposites is an implicit OBC algorithm, or short an I-OBC algorithm.

Hence, a large variety of I-OBC techniques are available. There are algorithms that are using type-I or type-II opposites which are defined as follows.

> **Explicit OBC Algorithms:** Any algorithm dealing with an unknown function $Y = f(X)$ and a known evaluation function $g(X)$ (with higher values being more desirable) extended with OBC to calculate type-I or type-II opposite $\breve{X}$ of numbers, guesses or estimates X for considering $\max(g(X), g(\breve{X}))$ in its decision-making is an explicit OBC algorithm, or short an E-OBC algorithm.

Three classes of E-OBC algorithms are distinguished in [92], Initializing E-OBC Algorithms, Somatic E-OBC Algorithms, and Initializing and Somatic E-OBC Algorithms.

> **Initializing E-OBC Algorithms:** The concept of opposition is only used during the initialization. The effect is a better start (initial estimates closer to solution vicinity). Further, since it is done before the actual learning/searching begins, this creates no additional overhead.

> **Somatic E-OBC Algorithms:** This approach is based on modification of the body of an existing algorithm. For instance, changing the weights in a neural network during the learning based on integration of opposite weights changes the way the algorithm works in every step. The effect of OBC will be much more visible since opposition is considered in every iteration/episode/generation.

> **Initializing and Somatic E-OBC Algorithms:** This class uses OBC both during initialization and actual learning/search by combining the two previous approaches.

In the next section some applications of opposition-based computing are discussed.

2.2.3 Applications of Opposition-based Computing

The concept of opposition can be implemented in the framework of many learning and optimization techniques to improve the performance or outcome of the methods. The advantage of using this technique has been investigated for reinforcement learning [73, 74, 76, 77, 90, 91, 92], evolutionary algorithms [53, 54, 55], neural network [77, 95], grid-based target estimation [78], and swarm intelligence [43].

The main focus of this research is to implement the concept of opposition for the reinforcement learning to accelerate the learning performance. How can the idea of opposition and RL be combined? As mentioned earlier, the knowledge about actions can be implemented to operate with actions and their associated opposite actions. One of the threads in the history of RL is the study of animal learning. In this research this thread is extended to the concept of opposition for actions. To clarify the opposite action for RL, an example of animal learning is provided to present the idea of learning from reward and punishment[3]:

> *Example* - Assuming an animal trainer has the intention to train his dog to fetch a ball. The dog has two choices, which can be considered as actions. These options are going toward the ball or staying there near the trainer and not moving. If the dog goes toward the goal, it receives its favorite treat as a reward which is provided by the trainer for reinforcing good actions. If the dog stays, then it receives nothing which can be considered as a punishment.
>
> An intelligent agent acts the same way by using reward and punishment. Based

[3]It must be mentioned that in psychology "punishment acts as the opposite of reinforcement" and it should not be confused with the proposed technique [103]

> on this theory, in order to teach the dog a good action, the dog must try all the actions and receives rewards or punishments to learn which action yields to more rewards. What happens if the dog receives reward when bringing a ball knowing that taking opposite action (staying) means receiving punishment? It means that the dog, exactly after receiving a reward for fetching the ball, can realize the strong possibility of receiving punishment for opposite action without taking the opposite action. If the trainer could tell the dog that receiving a reward for bringing a ball means that taking opposite action (staying there) brings punishment, and if the dog could understand this conversation, the learning process would be faster.

The condition for convergence of Q-learning implies that each state-action pair must be visited infinite times which is expensive in terms of running time. The main objective of this research is to propose new techniques for accelerating the learning process for off-policy, step by step, incremental and model-free reinforcement learning with discrete state and action space. Different algorithms are introduced which will be discussed in the next chapters.

There are some primary investigations conducted in this work to define the opposition for the components of reinforcement learning such as environment, actions, and the reinforcement feedback. This led to a new algorithm which is not a reinforcement learning method but it has the same RL components such as environment, actions, and the feedback. This algorithm is oppositional target domain estimation using grid-based simulation (OTE) which will be explained in the next chapter.

Chapter 3

Oppositional Target Domain Estimation Using Grid-Based Simulation

3.1 Introduction

The purpose of this research is employing the concept of opposition to expedite the learning process for some of the reinforcement learning techniques. Concept of opposition has been reported to have positive effects on the performance of Q-learning algorithm in terms of running time [90, 91, 92]. However, the technique was limited to deterministic environments. This reduces the applicability of the technique. Moreover, in some cases the opposition has negative impact on the learning process because of the issue of reward/punishment confusion (for some states) which will be discussed in the next chapter. Hence, the goal of this research is to use opposition for model-free and non-deterministic environments.

As mentioned in the previous chapter, there are many ways to implement opposition

in the existing algorithms since this concept has a broad range of applicabilities. This research is started by implementing the opposition in the components of reinforcement learning algorithms such as states, actions, and reinforcement signal. The grid environment is implemented here to test the benefit of the opposition. This yields to development of Oppositional Target Domain Estimation (OTE). OTE is not constructed based on learning, but it rather uses the same RL components for search in the grid environment. It searches for a reduced space which includes the target. The opposition concept makes the algorithm to perform faster comparing to the random search for the target.

The OTE algorithm can be used for many applications such as optimization or rescue operations. OTE is the first attempt to establish the concept of opposition in the framework of reinforcement learning and is based on type-II opposition. The algorithm can be considered in the category of 'Initializing E-OBC Algorithms' because the concept of opposition is used during the initialization and then a condition is verified to find a reduced space. The further attempt for using opposition in reinforcement learning generates a class of opposition-based $Q(\lambda)$ algorithms which will be introduced in the next chapters.

Since the model of the environment should be presented as a grid for the OTE algorithm, it could be a beneficial technique for search and navigation in the urban area. Hence, in the next section a brief literature review of search and navigation is discussed.

3.2 Search and navigation

Search and navigation are challenging research areas and have many applications in robotics, human-robot interaction, urban search and rescue (USAR), and optimization problems of finding global minima or maxima.

Search and rescue operations require navigation in a vast area. The robots must be

able to sense the environment by receiving feedback information about victims or source of contamination in the area. In other words, the robots should sense and monitor the states describing the environment. The robots search the environment to reach a target. The target could be a victim in the rubble pile, toxic substances, nuclear wastes and other source of contamination, or missing objects and material. A target may denote various meaning for other applications such as a minimum or maximum in optimization domain. However the focus of this chapter is more on the navigation in deterministic environments.

Hudock et al. [25] emphasize the design issues of robotic platform for search and rescue in urban environments. They describe their primary purpose as developing a robot which is mobile, light, and can be controlled easily. They discuss several issues regarding the design of search and rescue robots. For instance, they mention the ability of robot to move in the enclosed debris, flexibility of its movement around obstacles and stands, and transportability. These issues yield to new generation of robots with more movement flexibility to access the search states of the environment. Kadous et al. [28] indicate that generating "dense, textured 3D maps" and merging and presenting them should be considered as design issues regarding the robotic search and rescue.

Kantor et al. [32] investigate the search and rescue operation by proposing distributed search and using teams of robots and sensors. Their ad hoc network of distributed mobile sensors is a solution to the emergency response problem based on the cooperative localization technique. The sensors are responsible for collecting temperature data and running a distributed algorithm to collect the temperature gradient for mobile navigation. They assume that dissimilar groups of humans, robots, static and mobile sensors are involved in search and rescue operation in the building. They propose using of "gradients of temperature" and "concentration of toxins", and searching for "immobile humans".

Thrun et al. [86] also present a localization technique for estimating a robot's location

and orientation. Their technique is based on the Monte Carlo approach. They present "Mixture-MCL" which is "a version of particle filters that combines a regular sampler with its dual" and can be suitable to estimate posteriors for Bayesian in the localization problem.

Birk et al. [8] introduce their intelligent mobile robot, IUB Rugbot, for search and rescue missions. They use the integration of intelligent software and hardware techniques for search and rescue operations. They also take into account several other approaches such as adjustable autonomy strategy, wireless networking and exploration, and mapping of the area with robots. Their Rugbot benefits from onboard PC and a variety of sensors and cameras. Their software has several abilities of mapping a detected human, controlling, and tele-operating.

Castillo et al. [9] propose vision-based template matching for detecting human victim in search and rescue by integrating human silhouettes, corners and skin presence. They present an automatic way of producing templates from set of photos.

Advanced developments in the search and rescue operations provide new capabilities in this field. Mobility, robustness, perception, recognition, decision making, navigation, localization, tele-operation, large scale coordination, resource allocation, autonomous control, sophisticated sensory perception, and mapping are some of the technological advances for the robotic search and rescue mission [17, 36, 39, 65, 99, 100].

The design of the proposed algorithm is motivated by the robot capabilities and requirements for the search and rescue operations. Robots either have capabilities to sense the environment by using sensors, or they can receive the signal from sensors which are distributed in the environment. The main task of a search and rescue operation is to "locate as many victims as possible in the entire area" [65].

In this chapter the environment is modeled by a discrete grid and a software agent has

the ability to sense each cell of the grid. We assume that the map of urban environment is provided and represented by a grid. Hence, the proposed technique is suitable for real urban search and rescue tasks.

The *oppositional target domain estimation*, OTE, is proposed to reduce the size of search and navigation area and accelerate the speed of search for a target. We integrate the concept of opposition [54, 55, 53, 74, 76, 90, 91, 92, 95] within a new scheme to establish a state reduction algorithm for *oppositional target domain estimation*. The goal of the algorithm is to search for some states/actions that satisfy certain conditions in order to reduce the environment to a smaller area containing the target. The mathematical proof and the experimental results show that the proposed algorithm is a promising solution to reduce the navigation environment to a smaller area.

The main idea of OTE algorithm is reducing the size of the environment and increasing the efficiency and applicability of the search agent. The general characteristics of the OTE algorithm are:

- OTE reduces the size of the environment
- OTE estimates the target domain inside the environment
- OTE performs the crucial task of estimating target domain for search and navigation tasks
- The environment is modeled as a grid
- The reduction is based on the concept of opposition
- OTE theorem defines the conditions for the agent to find a sub-space without losing the target

The proposed algorithm is tested for three grid sizes and the target is randomly chosen in different areas of the grid. The aim of the experiment is to investigate the effect of the target position on the performance of the algorithm. Three quantities measure the performance, robustness, and efficiency of the algorithm. The results are also compared with the results of the randomized target domain estimation, RTE, which is a Monte Carlo inspired approach. Monte Carlo techniques have been applied to mobile robot localization as mentioned earlier [86]. The goal of this comparison is to verify the advantage of using opposition concept over the random selection of states.

3.3 Oppositional Target Domain Estimation

One of the characteristics of the reinforcement learning problem is that the model of the environment or samples of learned policy of action-state pairs for supervised learning or rules for knowledge-based schemes, as well as probability distribution over states may not be provided to the agent. However, reinforcement learning agent can learn by trial and error without using a priori knowledge. Therefore reinforcement learning is mostly suitable for the unknown environments. In contrast, for the grid-based applications, the navigation environment is modeled by a grid. The proposed technique is based on state of the art robot characteristics presented earlier. The environment is modeled by a grid and a software agent is designed capable of navigating the grid cells. The map of the urban environment is provided and represented by a grid. Hence, the proposed technique is suitable for real urban search and rescue tasks. Other techniques for the search and navigation may benefit from the proposed technique if the model of the environment could have the same characteristics of the proposed model.

Estimating a target in the search and navigation applications is a crucial task. A target

can be a source of contaminated and hazardous material or a human trapped inside the collapsed buildings in the emergency situations caused by an earthquake or a tornado. Fast estimation of the target domain prevents human mortality and environmental damage. In this chapter an algorithm is proposed to estimate the target domain based on the concept of opposition. As it was mentioned earlier in Section 3.1, software and hardware techniques advance the real-world search and rescue mission. There are new robots with variety of shapes and sizes equipped with cameras, sensors and wireless devices. These facilities make them flexible enough to navigate through the debris and collapsed areas, receive feedback, and perform in hazardous environments.

The *oppositional target domain estimation* (OTE) algorithm [78] is presented as a new strategy to reduce the state of the environment to the area which includes the target. The proposed numerical simulation presents a software agent that uses OTE algorithm for finding a target domain, and saves time by reducing the size of the search space.

The components of the proposed search algorithm are defined as follows:

- *States:* parameters/features describing the environment
- *Actions:* motion commands to navigate through the environment
- *Evaluative feedback:* signals indicating the quality of actions taken

In the following subsections the components of the technique and their major role in the search mission will be described.

3.3.1 States of the Environment

The proposed technique uses the model of the environment (e.g. a map of the area) and represents it as a grid. Hence, each cell of the grid represents a state of the environment.

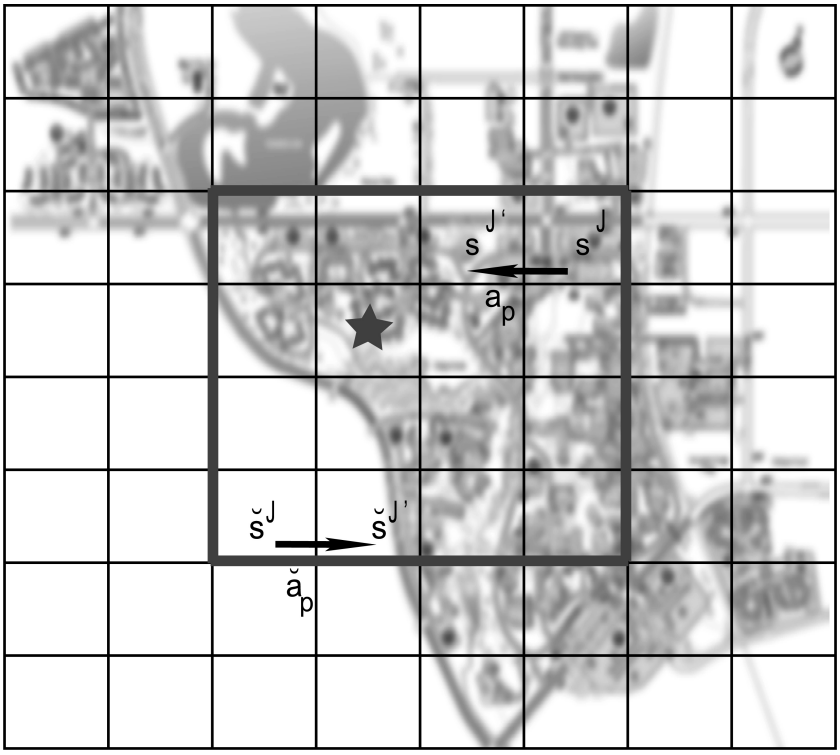

Figure 3.1: State space with the target marked by a star

Figure 3.1 is a map[1] representing an urban environment. A star represents a target inside the environment.

Each state S has its associated opposite state $\breve{S}$. The opposite state is determined based on the definition of opposite number which was introduced in Chapter 2. Here is the definition of opposite state:

> **Definition of Opposite State** Assume that the environment is an n dimensional state space, which is a subset of $\mathbb{Z}$. Each state S is considered as a point with coordinates $s_1, s_2, ..., s_n \in \mathbb{Z}$ and $b_i \leq s_i \leq c_i \ \forall i \in 1, ..., n$. The opposite

[1]Map of University of Waterloo which has been taken from Google Map. The map then has been blurred to make the grid more visible.

state $\breve{S}$ is defined by its coordinates $\breve{s}_1, \breve{s}_2, ..., \breve{s}_n$ where

$$\breve{s}_i = b_i + c_i - s_i. \tag{3.1}$$

Figure 3.1 also illustrates the state S^J with its associated opposite state $\breve{S}^J$.

3.3.2 Actions

An action causes the software agent to move from one grid cell to another by the amount of Δ (a single unit and integer number). In other words, in the n dimensional state space, an action has the ability to change (increase/decrease) the state coordinates by the amount of Δ. Assume that an action a is applied to state S and transfers it to the new state S'. Then there is at least one coordinate like $s_i \in S$ which is transfered to s'_i as a result of taking this action. Therefore, the modified state component can be given by the following relation:

$$s'_i = s_i \pm \Delta, \tag{3.2}$$

where Δ is a single unit and positive integer number. Hence, we can define the opposite action:

> **Definition (Opposite Action)** – If an action a changes one or more coordinates of a given state in a certain direction by the amount of Δ (a single unit and positive integer number), then the opposite action $\breve{a}$ changes the coordinates to the opposite direction by the same amount.

For example, if an action a is applied to state S and transfers it to the new state S' (by changing $s_i \in S$ to $s'_i = s_i \pm \Delta$), then the opposite action $\breve{a}$ can be applied to S' to

transfer the s'_i to s_i. By applying $\breve{a}$ in state S', there will be a transition to a state S such that $s_i = s'_i \mp \Delta$.

3.3.3 Evaluative Feedback

As mentioned earlier in Section 3.1, in search and rescue operations sensors receive information from the environment to use it as a navigation guide to localize the target [32]. New generation of robots are equipped with wireless and intelligent software technology [8]. Many advanced capabilities are also considered for designing search and rescue robots such as ability to move in the enclosed debris and around obstacles as well as transportability of the robots [25]. Such characteristics make the robots capable of sensing the environment and gathering data during the navigation.

In the design of our software agent we are motivated by the robot capability of receiving feedback information from the environment. We define the evaluative feedback for the agent based on varying intensity of the feedback received from environment. The type of the feedback can be different. In real-world applications the type of the sensors defines the nature of the feedback.

It is assumed that the intensity of a feedback signal will increase when the agent takes a step toward the target. On the contrary, if the agent moves away from the target the intensity of the signal will decrease. In other words it is assumed that the intensity of the signal is a function of the distance from the target. Hence, the definition of the evaluative feedback is based on this assumption. We consider two types of evaluative feedback, reward and punishment. The evaluative feedback is a result of taking an action by the agent. If the agent is in a shorter distance to the target (after taking an action) then the evaluative feedback is a reward r otherwise it is a punishment p.

3.3.4 *OTE* Theorem

The oppositional target domain estimation (*OTE*) uses a discrete grid to model the environment. Each grid cell represents a state of the environment. *OTE* addresses the navigation problem and is a suitable technique for searching in a large area. It reduces the size of environment and hence speeds up the search process.

In the *OTE* technique the search function looks for a sub-space (smaller than the original space) that includes the target. Consequently, the number of states will be reduced to the cells in this sub-space. It should be mentioned that the agent has the ability to go from one state in all directions, "left", "up-left", "down-left", "right", "up-right", "down-right", "up", and "down".

The *OTE* theorem is based on the definitions introduced in Sections 3.3.1, 3.3.2, and 3.3.3. The target is a point (terminal state) in a state space where the agent aims to reach. For instance the target can be a human trapped in the hazardous building or a source of contaminated material. The main idea of employing *OTE* for state-space reduction is presented as follows:

If taking opposite actions in opposite states leads to the same evaluative feedbacks, and if at least for one action we receive reward, then the target is located between those states and the rest of the search space can be disregarded. The following theorem establishes this idea in a formal way [78].

> *OTE* **Theorem** – Assuming there is an n dimensional state space with the state coordinates $s_i \in [b_i, c_i]$ containing a target. Further consider a state S with coordinates $s_1, s_2, ..., s_n$. For this state, the opposite state $\breve{S}$ is calculated using the intervals of the state space $[b_i, c_i]$ (see Equation 3.1). Consider that all directions are represented in the action set and Δ is a single unit and positive integer number for each action ($\Delta = 1$). If the results of all admissible

actions a_m applied to the state S are the same as the results of their corresponding opposite actions $\breve{a}_m$ applied to corresponding opposite state $\breve{S}$, and for at least one action and its corresponding opposite action the evaluative feedback is reward r, then the target with the optimal (terminal) state S^T is located between S and $\breve{S}$. For the terminal state S^T (target position) with coordinates $s_1^T, s_2^T, \cdots, s_n^T$, we receive $\min(s_i, \breve{s}_i) \leq s_i^T \leq \max(s_i, \breve{s}_i)\ \forall i \in \{1, ..., n\}$. This sub-space, spanned between S and $\breve{S}$, is called the *reduced space*.

Figure 3.1 shows an example of the application of the OTE theorem for a two-dimensional grid-world. In this figure the reduced space is highlighted by a bounding box.

This thesis will introduce an algorithm and deliver experimental results to establish empirical evidence for the OTE Theorem. However, a proof prior to the experimental verification is provided as well [78] .

Proof – Assume that the j-th state $S^j = (s_1^j, s_2^j, \cdots, s_n^j)$, its opposite state $\breve{S}^j = (\breve{s}_1^j, \breve{s}_2^j, ..., \breve{s}_n^j)$, the terminal state (target position) $S^T = (s_1^T, s_2^T, \cdots, s_n^T)$, and a evaluative feedback $R(S^j, a_m)$ which is a reward r or a punishment $p = -r$ are given. Now, assume that $R(S^j, a_m) = R(\breve{S}^j, \breve{a}_m)\ \forall\ m,\ \exists\ k$ so that $R(S^j, a_k) = R(\breve{S}^j, \breve{a}_k) = r$, and $\exists\ l$ so that $\min(s_l^J, \breve{s}_l^J) \nleq s_l^T \nleq \max(s_l^J, \breve{s}_l^J)$. This means that OTE conditions are satisfied but the target is outside the reduced space. Then there is at least one dimension like l and a corresponding coordinate of the target like s_l^T in such a way that this coordinate is larger or smaller than the corresponding coordinates of state s_l^J, and opposite state $\breve{s}_l^J$. Then for the target coordinate s_l^T one of the following relations must hold:

Case 1: $s_l^T < \breve{s}_l^J$ and $s_l^T < s_l^J$;

Case 2: $s_l^T > \breve{s}_l^J$ and $s_l^T > s_l^J$.

If case 1 occurs, then there is at least one action like a_p leading to the state transition $s_l^J - \Delta = (s_l^J)'$ and causing a reduction of the distance from the target. Therefore $R(s_l^J, a_p) = r$. The opposite action $\breve{a}_p$ must be applied to opposite state, therefore $\breve{s}_l^J + \Delta = (\breve{s}_l^J)'$ leading to increase in distance from the target and therefore $R(\breve{s}_l^J, \breve{a}_p) = -r$. This, however, contradicts the assumption $R(S^j, a_m) = R(\breve{S}^j, \breve{a}_m) \quad \forall m$.

If case 2 occurs, then there is at least one action like a_p leading to the state transition $s_l^J + \Delta = (s_l^J)'$ and causing an increase in distance from the target. Therefore $R(s_l^J, a_p) = -r$. The opposite action $\breve{a}_p$ must be applied to opposite state, therefore $\breve{s}_l^J - \Delta = (\breve{s}_l^J)'$ leading to decrease in distance from the target and therefore $R(\breve{s}_l^J, \breve{a}_p) = r$. This, however, contradicts the assumption $R(S^j, a_m) = R(\breve{S}^j, \breve{a}_m) \,\forall\, m$ as well. ■

Recall that the intensity of a feedback signal has a direct relation to the distance from the target. Figure 3.2 visualizes the proof by contradiction.

Before introducing the algorithm of oppositional target domain estimation (OTE), we summarize the constraints of the OTE Theorem as *equality* and *r-equality* conditions:

- The *equality constraint* constitutes that the result of taking all admissible actions (actions presenting one step movements in all directions) a_m in the state S^J is equal to the result of taking opposite action $\breve{a}_m$ in opposite state $\breve{S}^J$: $R(S^j, a_m) = R(\breve{S}^j, \breve{a}_m) \,\forall\, m$.

- The *r-equality constraint* requires that for at least one action a_k the result is reward: $\exists\, k$ so that $R(S^j, a_k) = R(\breve{S}^j, \breve{a}_k) = r$.

The OTE technique is presented in Algorithm 5 [78]. All directions are represented in the action set of the algorithm and Δ is a single unit and positive integer number. In

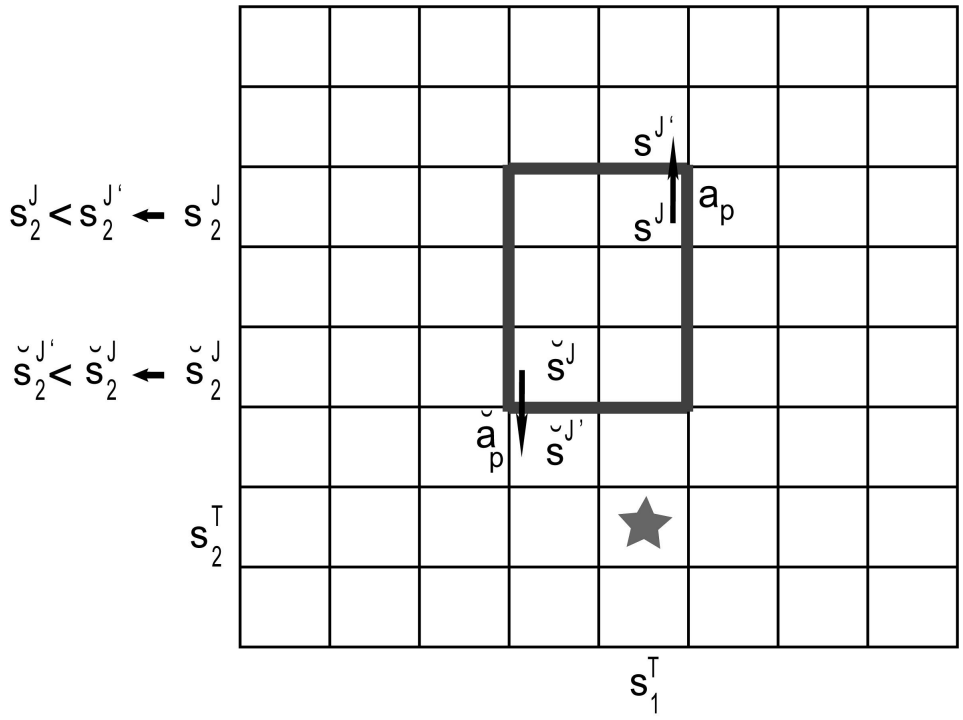

Figure 3.2: State space: If the target is located outside the reduced space (cells between S^J and $\breve{S}^J$, highlighted by the bold bounding box), then the action and opposite action yield different results which contradicts the OTE assumption. In this figure the result of a_p (represented by the upward arrow) is punishment whereas the result of $\breve{a}_p$ (represented by the downward arrow) is reward. The subscripts represent coordinates and the superscripts represent elements of each coordinate

Algorithm 5 Algorithm of oppositional target domain estimation (OTE)

Initialize the boundaries of the state space
Define the set of states and actions
Define reward and opposite reward (punishment)
while *equality* and *r-equality* conditions are not satisfied or there is an obstruction in the state or in the opposite state **do**
  Select an arbitrary state S in the state space
  Calculate the associated $\breve{S}$ using the boundaries of the state space
  for all actions a and their associated opposite actions $\breve{a}$ **do**
    Apply a in state S, observe the result
    Apply $\breve{a}$ in state $\breve{S}$, observe the result
    Compare results of a & $\breve{a}$ for equality & r-equality constraints
    If the state satisfies equality & r-equality for all a then exit
  end for
end while
Return S & $\breve{S}$ that satisfies equality & r-equality constraints for all a
Choose the sub-space between S & $\breve{S}$ as the reduced space
Exit algorithm

the next section the experimental results will be presented and discussed for different grid sizes.

3.4 Experimental Results

In this experiment series a grid is selected to simulate the environment. The experimental results are collected for several grid sizes, 200×200, 500×500, and 1000×1000. The grid is the navigation environment and each grid cell represents a state of the environment. A sample grid with the actions is presented in Figure 3.3 (left). As it is presented in this figure the agent can move in eight possible directions marked by arrows. These directions represent four actions and four corresponding opposite actions (if $a_l = Up$, then $\breve{a}_l = Down$). The actions/opposite actions are left/right, up/down, up-right/down-left, and up-

left/down-right. The goal of the agent is to reach the target marked by a star in the Figure 3.3 (left). The grid is divided into different areas as it is presented in the Figure 3.3 (right) and the target is randomly selected in each area. A constant integer is used as evaluative feedback. Hence, the value of reward has been set to 10 and the value of the punishment has been set to -10. It is intended to design and run experiments to investigate the ability of the OTE algorithm for reducing the search space and measure the sensitivity of the performance to the position of the target.

Intuitively, for the cases that the target is in the vicinity of the grid center, the reduction rate should be higher. This is due to the higher possibility that the condition of the theorem will be met with a small number of random guesses. As it is mentioned, in order to investigate this hypothesis the grids have been divided into several areas and the target has been randomly placed in these areas. In other words, the states have been selected randomly but the target has been chosen randomly in different areas to study the effect of the target location on the performance of the algorithm. The random selection of a state is not required by the algorithm but it is necessary to reduce the influence of any bias associated with state selection for measuring the performance of the algorithm. Figure 3.3 (right) illustrates a sample of the grid with 10 different areas A_i. The target is randomly selected in each area.

For each grid area, the algorithm is run several times and the target is randomly chosen in that area. The average and variance of three measures are calculated after several runs for each area and for each grid size. These measures are presented as $\bar{R}_r$ (average reduction rate), N_f (number of failures), and $\bar{N}_g$ (number of required guesses).

R_r quantifies the amount of state-space reduction for grid environment after applying the algorithm. The average reduction rate $\bar{R}_r$ for several runs quantifies the performance of the algorithm for each grid area.

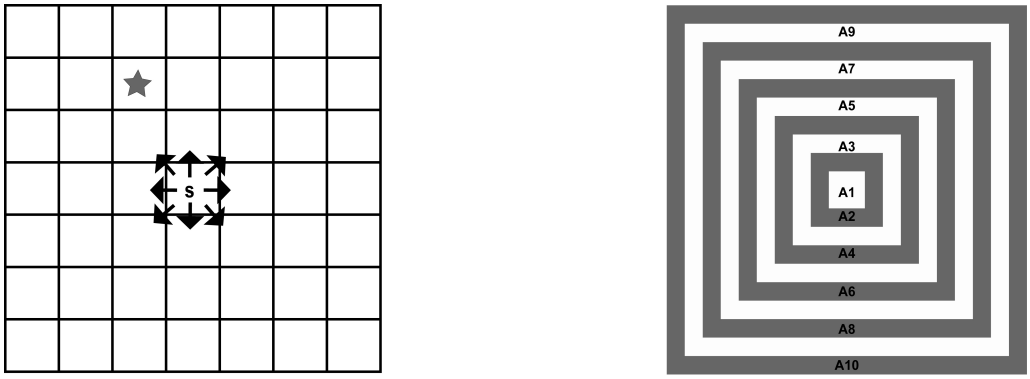

Figure 3.3: left: Sample grid including eight possible actions represented by arrows. S is a possible state and the star is the target; right: Sample areas in the grid for selecting the target in order to investigate the impact of position of the target on the performance of the algorithm

N_f is the number of failures for each area when the algorithm fails to find the state satisfying the conditions of the OTE theorem in a pre-determined number of steps. In each case when number of guesses (number of random states) has exceeded a threshold θ during the search process, one failure will be counted for the algorithm. The definition of threshold θ is arbitrary but the number of states (grid size) and number of actions have been taken into account since in each state the actions should be tested for investigating the satisfaction of the conditions of the theorem. The threshold θ is calculated as

$$\theta = (0.01) \times X_{max} \times Y_{max} \times |A|, \tag{3.3}$$

where $X_{max} \times Y_{max}$ is the grid size and $|A|$ is the number of actions. For individual grid

sizes we receive

$$\begin{aligned} \theta_{200\times200} &= (0.01)\times200\times200\times8 &&= 3200, \\ \theta_{500\times500} &= (0.01)\times500\times500\times8 &&= 20000, \\ \theta_{1000\times1000} &= (0.01)\times1000\times1000\times8 &&= 80000. \end{aligned} \tag{3.4}$$

The number of failures N_f will provide information on the robustness of the algorithm for each grid area.

N_g (number of guesses) is the number of random states that the agent selects during the search process. As mentioned before, agent selects a random state and checks the condition of the OTE theorem for each run. If conditions are not satisfied or there is an obstacle in a selected state or in the corresponding opposite state, then the agent randomly selects another state (another random guess). $\bar{N}_g$ quantifies the efficiency of the algorithm for each grid area.

Table 3.1 summarizes the results of the proposed technique for $200\times200, 500\times500$ and 1000×1000 grid-worlds. Each grid is divided into several areas A_i for random target selection.

The results of the OTE are compared with the results of the randomized target domain estimation, RTE. RTE is a Monte Carlo inspired technique that randomly generates states in the grid environment. In each run two random numbers are uniformly generated. Then the agent reduces the grid area to the area between the two random states as it is performed by the OTE method. Then the agent simply checks whether the reduced area includes the target or not. The goal is to compare the results of the OTE technique with the results of RTE and verify the benefit of the OTE over the totally randomized search for the reduced space. In the RTE approach, N'_f is the number of failures for each area when the algorithm fails to find the target in the reduced space. It is measured by considering the number

Table 3.1: Results of the *OTE* technique for different grid sizes. The target was chosen randomly in each area A_i. Measurements: average reduction size $\bar{R}_r$ (and standard deviation), number of failures N_f, and average number of guesses $\bar{N}_g$ to generate the reduced space

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}
200×200 grid-world with two areas A_1 and A_2 (200 trials in each area)										
$\bar{R}_r$	68% (18%)	64% (21%)								
N_f	0	27								
$\bar{N}_g$	6	229								
500×500 grid-world with five areas $A_1 - A_5$ (500 trials in each area)										
$\bar{R}_r$	71% (20%)	69% (16%)	63% (20%)	60% (20%)	54% (26%)					
N_f	0	0	0	0	56					
$\bar{N}_g$	2	4	13	47	1342					
1000×1000 grid-world with ten areas $A_1 - A_{10}$ (1000 trials in each area)										
$\bar{R}_r$	74% (20%)	73% (18%)	68% (18%)	66% (17%)	64% (19%)	63% (19%)	59% (20%)	57% (22%)	55% (24%)	51% (27%)
N_f	0	0	0	0	0	0	0	0	0	117
$\bar{N}_g$	1	2	3	5	9	17	29	63	202	4540

of random guesses exceeding a threshold θ during the search process. In such cases, one failure will be counted for the algorithm. The threshold θ is the same as the threshold that has been used for calculating number of failures for the OTE algorithm. The results of the randomized approach, RTE are presented in Table 3.2.

The results of all simulations can be summarized as follows:

- An average reduction rate $\bar{R}_r$ of above 50% has been reached in all cases. Although the reduction rate for OTE is less than the RTE, the cost of the reduction rate is significantly higher for RTE (average number of guesses required to meet the target in each area). The maximum difference of the reduction rate between OTE and RTE is 38% for the Area 10 in the 1000×1000 grid. The minimum difference is 11% for the Area 1 in the 1000×1000 grid.

- For the largest state space with 8×10^6 state-action pairs (8 actions $\times 1000 \times 1000$), a maximum of $74\% \pm 20\%$ and a minimum of $51\% \pm 27\%$ reductions were achieved by OTE. Hence, the performance of algorithm, promised by the OTE theorem, was experimentally evident for a relatively large state space.

- The low number of failures N_f for the OTE indicates that the algorithm is quite robust ($N_f = 0$ for almost all cases). Only if the target is located in the last area (grid border) the algorithm sometimes fails to find the reduced target domain within the pre-determined maximum number of iterations θ. In other areas which are not located on the border, the number of failures are zero. This is not the case for RTE except for the Area 1 in the 500×500 grid. Apparently, with increase in the number of states the robustness of OTE increases. Increasing the manually set threshold θ yields to decreasing the failure rate toward zero. This, however, would suppress the algorithm efficiency which is expressed in terms of number of required guesses N_g.

Table 3.2: Results of the *RTE* technique for different grid sizes. Two states are chosen randomly. The target was chosen randomly in each area A_i. Measurements: average reduction size $\bar{R}_r$ (and standard deviation), number of failures N'_f, and average number of guesses $\bar{N}_g$ to generate the reduced space

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}
200×200 grid-world with two areas A_1 and A_2 (200 trials in each area)										
$\bar{R}_r$	87% (13%)	88% (13%)								
N'_f	3	3								
$\bar{N}_g$	2584	2876								
500×500 grid-world with five areas $A_1 - A_5$ (500 trials in each area)										
$\bar{R}_r$	85% (15%)	86% (15%)	87% (14%)	87% (14%)	88% (13%)					
N'_f	0	3	3	5	2					
$\bar{N}_g$	14601	15293	16500	18022	19279					
1000×1000 grid-world with ten areas $A_1 - A_{10}$ (1000 trials in each area)										
$\bar{R}_r$	85% (14%)	86% (14%)	86% (14%)	86% (14%)	87% (14%)	86% (14%)	88% (14%)	87% (14%)	88% (13%)	89% (13%)
N'_f	5	2	1	2	2	2	4	5	1	3
$\bar{N}_g$	60464	59080	61503	61404	65893	67737	71086	72926	76078	78558

- In order to decrease the number of failures (N_f) in the vicinity of the grid border the RTE algorithm can be applied in that area instead of OTE method. The OTE algorithm can be used for other areas which are not in the vicinity of the border. As mentioned before, the OTE method uses the model of the environment and represents it as a grid. Hence, the agent can easily determine the states which are located in the vicinity of the grid border (based on the model of the environment) and switch from OTE algorithm to RTE in that area. Based on the experimental results presented in the Tables 3.1 and 3.2 by using this mixed strategy, the number of failures drops from $N_f = 27$ (OTE result) to $N'_f = 3$ (RTE result) in the area 2 for the 200×200 grid. The same improvement will be gained for the number of failures in the the area 5 of the 500×500 grid. The number of failures drops from $N_f = 56$ (OTE result) to $N'_f = 2$ (RTE result). Also, for the area 10 of the 1000×1000 grid the number of failures changes from $N_f = 117$ (OTE result) to $N'_f = 3$ (RTE result).

  However, this strategy of switching over to RTE increases the number of required guesses as presented in the Tables 3.1 and 3.2 in the vicinity of the border. The other issue which should be addressed in the future work is defining the size of the area located in the vicinity of the border. In the experimental results, area 2 for the 200×200 grid, area 5 for 500×500 grid, and area 10 for 1000×1000 grid are considered as the border areas.

- A low average number of necessary guesses $\bar{N}_g$ for finding the reduced space indicates the efficiency of the OTE algorithm. A couple of hundred guesses and the associated operations takes fractions of a second on ordinary computers. Only if the target is located close to the grid border, $\bar{N}_g$ increases. The number of guesses is significantly high for RTE which makes it very inefficient and time consuming in contrast to OTE.

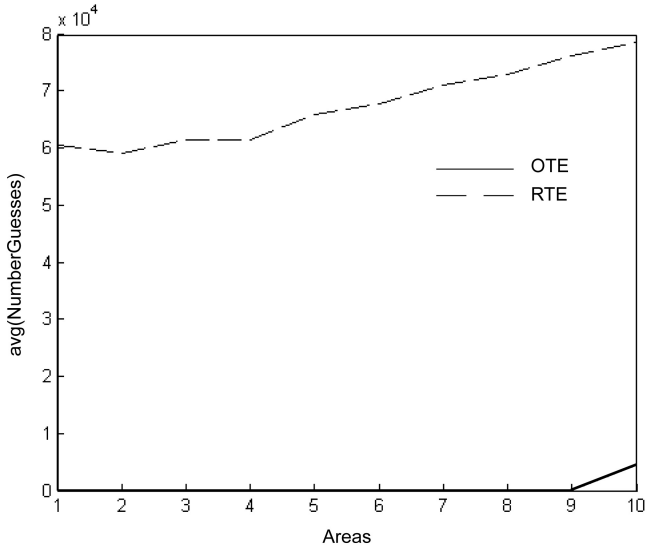

Figure 3.4: Average number of guesses $\bar{N}_g$ to generate the reduced space for OTE and RTE techniques in a 1000×1000 grid

Average number of guesses $\bar{N}_g$ and average reduction sizes $\bar{R}_r$ are also illustrated in the Figures 3.4 and 3.5 for OTE and RTE techniques and for the 1000×1000 grid (using data in Tables 3.1 and 3.2). Overall, it can be stated that OTE offers a considerable state-space reduction. The state-space reduction for grid-based cases can be performed quite efficiently. OTE is robust enough to deliver a solution unless the target is located in the vicinity of the grid border. In such cases, only an increase in computational expense may lead to reduction. The other option is to switch from OTE to RTE in the border areas. The mathematical proof for the benefits of opposition-based techniques comparing to the random approach is presented by Rahnamayan et al. [56].

The state reduction can be conducted during offline training for some of the applications such as optimization. Therefore, additional steps to ascertain the creation of a smaller search space in an offline phase may be well-suited in some applications.

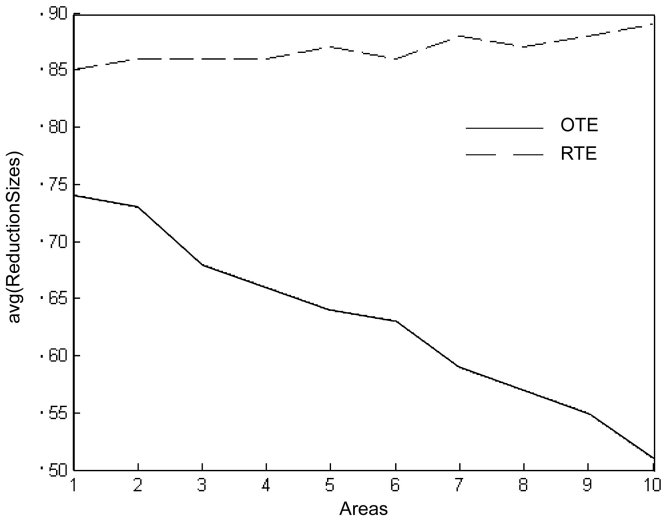

Figure 3.5: Average reduction sizes $\bar{R}_r$ for OTE and RTE techniques in a 1000×1000 grid

3.5 Summary and Conclusions

This chapter addressed the problem of estimating target domain in search and navigation in known environments. In this research oppositional target domain estimation (OTE) algorithm is introduced to tackle the problem of navigation in large environments. OTE helps the search operation by reducing the environment to a smaller area which includes the target. In this technique, the map of the environment is modeled by a grid. Agent can sense each state (grid cell) and has the ability to move to any neighboring cell. The OTE algorithm benefits from the concept of opposition. The OTE theorem established the conditions to find a sub-space without losing the target.

The OTE algorithm was tested for three grid sizes. In order to investigate the effects of the target position, the grids were segmented into different areas and tested the algorithm

for randomly selected targets in each part. Three quantities were measured to evaluate the performance, robustness, and efficiency of the algorithm.

The results of the proposed technique were also compared with the results of the randomized target domain estimation, RTE. The comparison investigated the advantage of using the opposition concept and the *equality* and *r-equality* conditions associated with it (formulated in the OTE *Theorem*) over the randomized target domain estimation inspired by the Monte Carlo technique. The experimental results are promising and suggest several advantages of the OTE algorithm which can be summarized as follows:

1. OTE is a straightforward algorithm for state-space reduction in search and navigation problems.

2. OTE can achieve high reduction rates.

3. It is widely robust and only takes more computational time if the target is close to the grid border.

4. OTE is generally very fast.

5. The agent can switch from the OTE technique to the RTE method in the areas located in the vicinity of the border to reduce the number of failures in these areas.

In summary, the proposed technique has promising performance by providing an estimation of a target domain, reducing the search environment, and accelerating the navigation. However, the model of the environment should be available for the OTE algorithm which limits the applicability of this technique. Therefore, in the next chapters the concept of opposition is extended to well-established reinforcement learning techniques and benefits of this concept are investigated for dynamic environments where the model of the environment or training samples are not provided for the agent.

Chapter 4

Concept of Opposition for Q-learning and $Q(\lambda)$ Techniques

4.1 Introduction

In many real-world problems the environment is non-deterministic and is changing over time or a complete or partial knowledge about parameters and model of the environment or learning policy is not available. In such applications reinforcement learning is useful technique to address the learning process. Temporal difference learning, TD, is an RL method and constructed based on assigning credit using difference between temporally successive predictions. Hence, no supervisor or training example is required in this type of learning and temporal sequence sensory input can act as a training example. Sutton [82] describes the "milder demand on computational speed", distribution of "arithmetic operations" over time, efficient use of "experiments comparing supervised methods", accuracy in prediction over time, and applicability for dynamic systems, as the advantages of using TD methods. However, the explorations of state space is computationally expensive for some applica-

tions. The main focus of this research is on presenting the concept of opposition in the framework of some RL techniques to expedite the learning process.

This research has investigated the effect of opposition for $Q(\lambda)$ technique because it is based on simple and widely used Q-learning method. Q-learning is one of the TD methods. There are hybrid techniques based on the combination of Q-learning and temporal difference learning such as $QV(\lambda)$-learning [98] and incremental multi-step Q-learning [49]. These technique are the bridge between Q-learning and actor-critic learning and take advantage of the qualities of both techniques to improve the learning process. However, the opposition-based $Q(\lambda)$-techniques presented in this research are based on the update for Q-values for states and opposite actions. Hence, any technique that has updates based on Q-function (which is a function of state and action) can benefit from the concept of opposition. In the hybrid techniques presented by Q-function and V-function, the value function (V) will not be affected.

In this chapter the special kind of knowledge about actions is employed. It should be emphasized that the knowledge about actions does not include the knowledge of state/action pair. Rather, this knowledge is embedded in the definition of actions. For instance, direction of the action "left" is part of the characteristic or definition of this action. Hence, the action "right" is the opposite direction of the action "left". Therefore the actions left and right are opposite of each other. Opposition first has been introduced for Q-learning (one of the TD methods) by Tizhoosh [92]. However, it was limited to the model-based techniques. Moreover, the issue of reward/punishment confusion which negatively affects the performance of opposition-based RL techniques has not been addressed [92]. Hence, in this research the opposition is extended to model-free techniques and the problem of reward/punishment confusion is solved by extending the opposition concept to the $Q(\lambda)$ technique [73, 74, 76]. The problem of Markovian update is discussed in this chapter and

non-Markovian update is presented as a remedy for this problem.

In this chapter the opposition-based RL algorithms are presented by using $Q(\lambda)$ technique. These methods are based on multiple and concurrent Q-value updates. In conventional Q-learning, an agent updates one state-action Q-value per iteration. In opposition-based Q-learning we assume that for each action there is an opposite action. If the agent takes an action and receives a reward (or punishment) for that action, it would receive punishment (reward) for the opposite action. Therefore, instead of taking the opposite action in a given state, the agent will update its associated Q-value. It means that for a given state the Q-values can be updated at the same time for both the action (which is taken) and its corresponding opposite action (which is not taken). This strategy saves time and accelerates the learning process [91, 90, 92]. In this chapter two oppositional RL techniques, opposition-based $Q(\lambda)$ ($OQ(\lambda)$) and opposition-based $Q(\lambda)$ with Non-Markovian Update ($NOQ(\lambda)$) will be introduced [73, 74].

4.1.1 Opposition-based $Q(\lambda)$

The relationship between the idea of opposition and RL have been explored in the framework of the opposition-based $Q(\lambda)$ technique, $OQ(\lambda)$ [73, 74]. If an $OQ(\lambda)$ agent at each time t receives a reward for taking the action a in a given state s, then at that time the agent also may receive punishment for opposite action $\breve{a}$ in the same state s without taking that opposite action. It means that the value function, Q, (e.g. Q-matrix in tabular Q-learning) can be updated for two values $Q(s, a)$ and $Q(s, \breve{a})$ instead of only one value $Q(s, a)$. Therefore, an agent can simultaneously explore actions and opposite actions. Consequently, updating the Q-values for two actions in a given state for each time step can lead to faster convergence since the Q-matrix can be filled in a shorter time [90, 91, 92].

Figure 4.1 demonstrates the difference between Q-matrix updating using reinforcement

learning (left image) and Q-matrix updating using opposition-based reinforcement learning (right image). In the left image the states are s_i and actions are a_j where $1 \leq i \leq 5$ and $1 \leq j \leq 4$. The agent in state s_2 takes action a_2 and receives reward r. Then by using reward r the value $v_1 = Q(s_2, a_2)$ of action a_2 in the state s_2 is calculated using the updating formula of Q-learning algorithm as follows:

$$Q(s_2, a_2) \leftarrow Q(s_2, a_2) + \alpha[r + \gamma Q(s', a') - Q(s_2, a_2)]. \tag{4.1}$$

In the right image of Figure 4.1, there are two actions a_1 and a_2 with their associated opposite actions $\breve{a}_1$ and $\breve{a}_2$. The agent in state s_2 takes action a_2 and receives reward r. By using reward r the value v_1 of action a_2 in the state s_2 is calculated as before. In the opposition-based technique we assume that the agent will receive an opposite reward by taking opposite action. Hence, by assuming that the agent will receive punishment p (opposite reward) by taking the opposite action $\breve{a}_2$ in state s_2, the value $v_2 = Q(s_2, \breve{a}_2)$ in Q-matrix is also updated as follows [90, 91, 92]:

$$Q(s_2, \breve{a_2}) \leftarrow Q(s_2, \breve{a_2}) + \alpha[p + \gamma Q(s', a') - Q(s_2, \breve{a_2})]. \tag{4.2}$$

It means that the value function Q can be updated for two values instead of only one value. Therefore, an agent can simultaneously explore actions and opposite actions. Hence, the additional update should accelerate the learning of the algorithm.

The $OQ(\lambda)$ algorithm is constructed based on opposition traces which represent eligibility traces for opposite actions. Assume that $e(s, a)$ is the eligibility trace for action a in state s, then the opposition trace is $\breve{e} = e(s, \breve{a})$. For updating the Q-matrix in a given state s, agent takes action a and receives reward r. Then by using reward r the Q-matrix

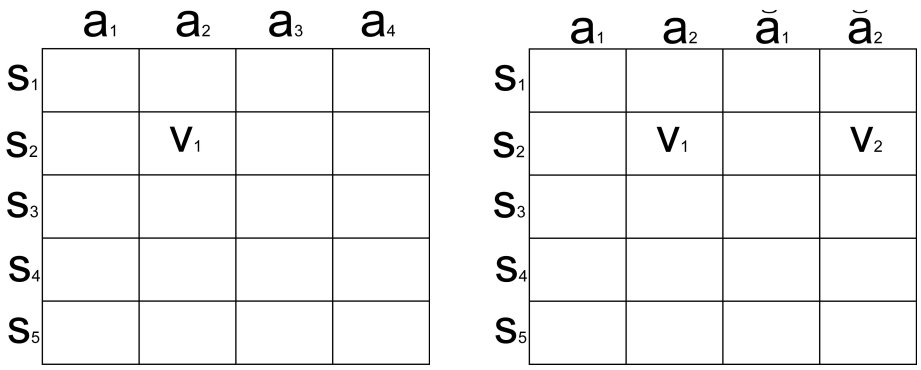

Figure 4.1: Q-matrix updating; Left: Q-matrix updating using reinforcement learning, right: Q-matrix updating using opposition-based reinforcement learning where an additional update (v_2) can be performed for the opposite action $\breve{a}_2$.

will be updated for all states and actions:

$$\delta_1 \longleftarrow r + \gamma Q(s', a^*) - Q(s, a), \tag{4.3}$$

$$Q(s, a) \longleftarrow Q(s, a) + \alpha \delta_1 e(s, a), \tag{4.4}$$

By assuming that the agent will receive punishment p by taking opposite action $\breve{a}$, the Q-matrix will be updated for all states s and opposite actions $\breve{a}$:

$$\delta_2 \longleftarrow \breve{r} + \gamma Q(s'', a^{**}) - Q(s, \breve{a}), \tag{4.5}$$

$$Q(s, \breve{a}) \longleftarrow Q(s, \breve{a}) + \alpha \delta_2 e(s, \breve{a}), \tag{4.6}$$

The $OQ(\lambda)$ technique [73] is presented in the Algorithm 6. $OQ(\lambda)$ differs from $Q(\lambda)$ algorithm in the Q-value updating. In $OQ(\lambda)$ the opposition trace facilitates updating of the Q-values for opposite actions and instead of punishing/rewarding the action and opposite

Algorithm 6 $OQ(\lambda)$ Algorithm. If the agent receives punishment p for taking an action then the opposite action receives reward r

For all s and a initialize $Q(s,a)$ with arbitrary numbers and initialize $e(s,a)=0$
for each episode **do**
  Initialize s and a
  for each step of episode **do**
    Take action a, observe r and next state s'
    Determine opposite action $\breve{a}$ and next state s''
    Calculate opposite reward (punishment) $\breve{r}=p$
    Choose next action a' from s' using policy
    Determine next opposite action $\breve{a}'$ from s''
    $a^* \longleftarrow arg\max_b Q(s',b)$ {if a' ties for max, then $a^* \leftarrow a'$}
    $a^{**} \longleftarrow arg\max_b Q(s'',b)$ {if $\breve{a}'$ ties for max, then $a^{**} \leftarrow \breve{a}'$}
    $\delta_1 \longleftarrow r+\gamma\ Q(s',a^*)-Q(s,a)$
    $\delta_2 \longleftarrow \breve{r}+\gamma\ Q(s'',a^{**})-Q(s,\breve{a})$
    $e(s,a) \longleftarrow e(s,a)+1$
    $e(s,\breve{a}) \longleftarrow e(s,\breve{a})+1$
    for all s, a in action set A **do**
      $Q(s,a) \longleftarrow Q(s,a)+\alpha\delta_1 e(s,a)$
      if $a'=a^*$ **then**
        $e(s,a) \longleftarrow \gamma\lambda e(s,a)$
      else
        $e(s,a) \longleftarrow 0$
      end if
    end for
    $s \longleftarrow s'$; $a \longleftarrow a'$
    for all s, $\breve{a}$ in the opposite action set $\breve{A}$ where $\breve{A}$ is not $\subset A$ **do**
      $Q(s,\breve{a}) \longleftarrow Q(s,\breve{a})+\alpha\delta_2 e(s,\breve{a})$
      if $\breve{a}'=a^{**}$ **then**
        $e(s,\breve{a}) \longleftarrow \gamma\lambda e(s,\breve{a})$
      else
        $e(s,\breve{a}) \longleftarrow 0$
      end if
    end for
  end for
end for

action[1] we punish/reward the eligible trace and opposite trace. The major difference between $Q(\lambda)$ and $OQ(\lambda)$ is in Q-value updating. The opposition trace is a facilitator for updating the Q-values for opposite actions to benefit from the idea of concurrent updating of the Q-matrix.

Reward/Punishment Confusion

One issue that must be addressed is that in some situations two opposite actions in a given state may yield the same result instead of opposite results. The action and opposite action may both lead to reward, or both lead to punishment. The example in Figure 4.2 illustrates this situation using the grid-world problem [73]. The goal is presented by a star and the present state is s. For both action a_1, and its opposite $\breve{a}_1$ the result is punishment because they both increase the distance of the agent from the goal. Hence, both of them should be punished. Rewarding one of them will falsify the value function and affect the convergence.

Opposition traces are possible solutions for this problem since the Q-matrix updating is not limited to one action and one opposite action at a given state, but also depends on updating more Q-values by using eligibility and opposition traces. In the methods based on eligibility traces, the effects of all eligible actions are considered for the update, not just one action. Therefore, all actions in the trace and all opposite actions in the opposition trace will affect the learning process and help to reduce the effect of reward/punishment confusion. It can also be stated that for grid-based problems with one target the influence of confusing cases becomes completely negligible with increase in dimensionality. However, the confusing cases can cause algorithm oscillations around the corresponding state if mechanisms such as opposition traces are not employed. This has been reported for

[1]It is assumed that when the agent receives reward (r)/punishment ($\breve{r}$) for taking an action, it will receive punishment ($\breve{r}$)/reward (r) for taking the opposite action.

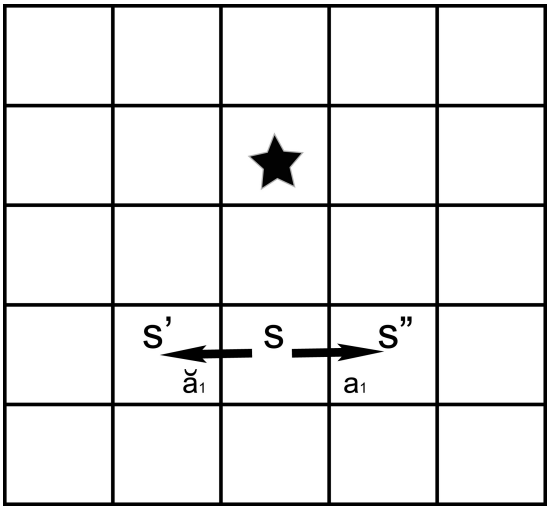

Figure 4.2: Example to show that two opposite actions in state s can yield the same result, here punishment. The target is presented by a star and the next state is s' or s'' [73]. If the agent takes an action and moves toward the target it receives a reward, otherwise it receives a punishment. In this example the agent is located at state s and takes the action a_1 or the opposite action $\breve{a}_1$. Hence, it receives punishment for both cases.

standard Q-learning [92]. The $OQ(\lambda)$ algorithm utilizes a Markovian update of opposition traces.

4.1.2 Opposition-Based Q(λ) with Non-Markovian Update

It has been assumed that the opposite state can be presented to the agent in the $OQ(\lambda)$ technique (see s'' in the Equation 4.5). This may limit the usability of the technique to deterministic environments because the next state of the environment (for the opposite action) should be provided, bearing in mind that the agent will not actually take the opposite action. In order to relax this assumption the Non-Markovian Opposition-Based $Q(\lambda)$ ($NOQ(\lambda)$) is introduced in this section.

The new method is a hybrid of Markovian update for eligibility traces and non-Markovian-

based update for opposition traces. The $NOQ(\lambda)$ method specifically focuses on investigating the possible non-Markovian updating of opposition traces where the next state for the opposite action may not be available. This extends the usability of $OQ(\lambda)$ to a broader range of applications where the model of environment is not provided for the agent.

The issue of the Markovian-based updating of the opposite trace in the $OQ(\lambda)$ algorithm is addressed here. As it is presented in Algorithm 6, line 6, the agent should determine the next state s'' after defining the opposite action $\breve{a}$. In this algorithm the agent does not actually take $\breve{a}$, but only determines the opposite one. For instance, if the action is going to the left in the environment then the opposite action is determined (not taken) as going to the right. In a deterministic environment the agent can figure out the next state by using the model of the environment. In the case of the grid-world problem for instance, the agent assumes that if it takes the action "right" and goes to the right cell (in the grid) then the next state s'' for the opposite action $\breve{a}$ ("left") is the left cell in the grid with respect to initial state. In this case the Q values can be updated for the opposition trace as follows:

$$a^{**} \longleftarrow arg\max_{b} Q(s'', b), \tag{4.7}$$

$$\delta_2 \longleftarrow \breve{r} + \gamma Q(s'', a^{**}) - Q(s, \breve{a}), \tag{4.8}$$

$$Q(s, \breve{a}) \longleftarrow Q(s, \breve{a}) + \alpha\delta_2 e(s, \breve{a}), \tag{4.9}$$

where α is step-size parameter and γ is a discount-rate parameter. Equations 4.7, 4.8, and 4.9 present the Markovian-based updating by considering the next state s'' [74]. In $NOQ(\lambda)$ we address this problem by introducing the non-Markovian updating for opposition traces.

The $OQ(\lambda)$ method updates opposite traces without taking opposite actions. For this reason the opposition update (the Markovian update) depends on the next state of the environment that should be known to the agent. This limits the applicability of the $OQ(\lambda)$ to

deterministic environments. We relax the constraint of Markovian updating by introducing a new update for opposition traces. Equation 4.10 presents the new update formula for opposition traces where $\breve{r}$ is the opposite reward, and $e(s, \breve{a})$ is the opposite trace:

$$Q(s, \breve{a}) \longleftarrow Q(s, \breve{a}) + w \cdot \breve{r} \cdot e(s, \breve{a}). \tag{4.10}$$

w is a parameter introduced to impose a weight between 0 and 1 on the opposition update. Hence, we assume that at the beginning of learning the weight of the update is low and increases gradually as the agent explores the actions and the opposite actions. The new update for $NOQ(\lambda)$ algorithm depends on the parameter w, opposite reinforcement signal $\breve{r}$, and the parameters such as α, γ, and λ which are defined like the parameters in the $Q(\lambda)$ method. Therefore, the Markov property is not violated in this update because it is a non-Markovian update and the next state for the opposite action has not been considered in the update. The new update represents online integration of domain knowledge embedded in the definition of actions; the original condition of the $Q(\lambda)$ algorithms for convergence are not changed. Hence, the convergence of the $NOQ(\lambda)$ can be assumed since the Markovian assumptions of underlying $Q(\lambda)$ have not been altered.

The details regarding the definition of w will be discussed in the next chapter. In the case of simple grid-world problem which is discussed in this chapter the weight w is set to 1. As it is presented in Equation 4.10, $Q(s, \breve{a})$ does not depend on the next state, in contrast to the $OQ(\lambda)$ technique which depends on s'' (see Equations 4.9 and 4.8). The $NOQ(\lambda)$ technique is presented in the Algorithm 7.

Figure 4.3 is the extension of the grid-world example of eligibility trace. In this extension the grid-world is presented with two traces. The black arrows represent increases of the action values. On the other hand, the gray arrows represent the negative increases (decreases) in the action values for the opposition trace. The trace ends at a location of

Algorithm 7 *NOQ*(λ) Algorithm. If the agent receives punishment p for taking an action then the opposite action receives reward r

For all s and a initialize $Q(s,a)$ with arbitrary numbers and initialize $e(s,a) = 0$
for each episode **do**
  Initialize s and a
  for each step of episode **do**
    Take action a, observe r and next state s'
    Determine opposite action $\breve{a}$
    Choose next action a' from s' using policy
    Determine next opposite action $\breve{a}'$
    $a^* \longleftarrow arg\max_b Q(s',b)$ {if a' ties for max, then $a^* \leftarrow a'$}
    $\delta_1 \longleftarrow r + \gamma\, Q(s',a^*) - Q(s,a)$
    $e(s,a) \longleftarrow e(s,a) + 1$
    $e(s,\breve{a}) \longleftarrow e(s,\breve{a}) + 1$
    for all s, a in the action set A **do**
      $Q(s,a) \longleftarrow Q(s,a) + \alpha\delta_1 e(s,a)$
      if $a' = a^*$ **then**
        $e(s,a) \longleftarrow \gamma\lambda e(s,a)$
      else
        $e(s,a) \longleftarrow 0$
      end if
    end for
    for all s, $\breve{a}$ in the opposite action set $\breve{A}$ where $\breve{A}$ is not $\subset A$ **do**
      $Q(s,\breve{a}) \longleftarrow Q(s,\breve{a}) + w \cdot \breve{r} \cdot e(s,\breve{a})$
      if $a' = a^*$ **then**
        $e(s,\breve{a}) \longleftarrow \gamma\lambda e(s,\breve{a})$
      else
        $e(s,\breve{a}) \longleftarrow 0$
      end if
    end for
    $s \longleftarrow s'$; $a \longleftarrow a'$
  end for
end for

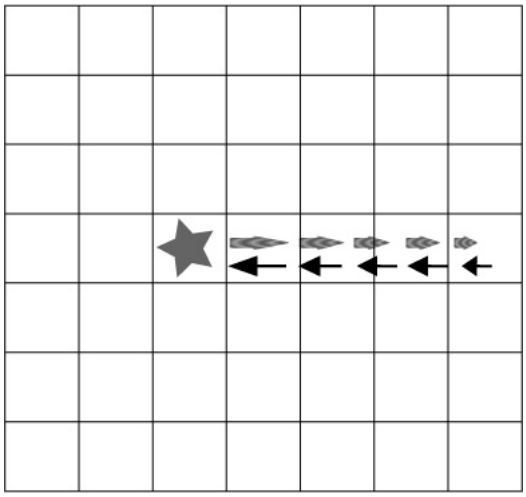

Figure 4.3: The extension of the grid-world with eligibility trace [83]. In this extension the grid-world is presented with two traces, eligibility and opposition trace.

high reward value marked by a star.

4.2 Experimental Results

The grid-world [74] problem with three sizes (20×20, 50×50, and 100×100) is chosen as a test case. The grid represents the learning environment and each cell of the grid represents a state of the environment. A sample grid-world is presented in Figure 4.4. The agent can move in 8 possible directions indicated by arrows in the figure. The goal of the agent is to reach the defined target in the grid which is marked by a star.

Four actions with their corresponding four opposite actions (if $a = Up$ then $\breve{a} = Down$) are defined. By taking an action an agent has the ability to move to one of the neighboring states. The actions/opposite actions are left/right, up/down, up-right/down-left, and up-left/down-right. The initial state is selected randomly for all experiments. If the size of a grid is (X_{max}, Y_{max}), then the coordinates of the target are fixed at $(\frac{X_{max}}{2}, \frac{Y_{max}}{3})$. The

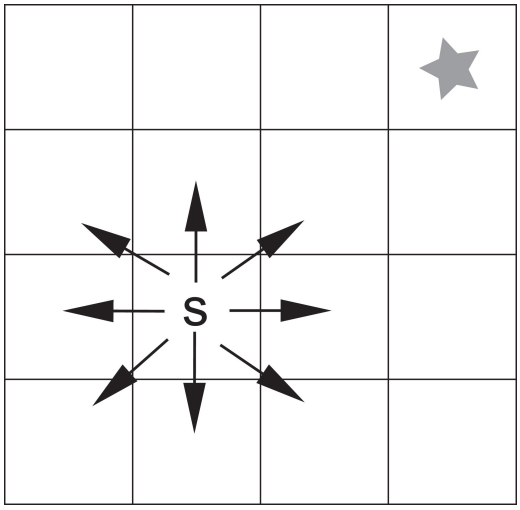

Figure 4.4: Sample grid-world [74]. There are eight possible actions presented by arrows. S is a state and the star is the target.

Table 4.1: The initial parameters for all experiments

n_E	I_{max}	α	γ	λ
100	1000	0.3	0.2	0.5

value of immediate reward is 10, and punishment is -10. After the agent takes an action, if the distance of the agent from the goal is decreased, then agent will receive a reward. If the distance is increased or not changed, the agent receives punishment. The Boltzmann policy [31] is applied for all implementations (see the Equation 2.1 in Chpater 2). The $Q(\lambda)$, $OQ(\lambda)$, and $NOQ(\lambda)$ algorithms are implemented. The initial parameters for the algorithms are presented in Table 4.1.

The following measurements are considered for comparing the results of $Q(\lambda)$, $OQ(\lambda)$, and $NOQ(\lambda)$ algorithms:

- overall average iterations $\overline{\overline{I}}$: average of iterations over 100 runs

Table 4.2: The results for the five measures of $\overline{\overline{I}}$, $\overline{E}$, $\overline{T}$, χ, and ζ for algorithms $Q(\lambda)$, $OQ(\lambda)$, and $NOQ(\lambda)$. The results are based on 100 runs for each algorithm

	$Q(\lambda)$			$OQ(\lambda)$			$NOQ(\lambda)$		
$X \times Y$	20×20	50×50	100×100	20×20	50×50	100×100	20×20	50×50	100×100
$\overline{\overline{I}}$	255 ± 151	528 ± 264	431 ± 238	20 ± 8	57 ± 26	103 ± 50	**19 ± 7**	**48 ± 23**	**100 ± 46**
$\overline{T}$	3 ± 2	17 ± 6.6	102 ± 25	0.3 ± 0.1	5 ± 1	80 ± 7	**0.2 ± 0.1**	**5 ± 0.7**	**58 ± 6**
χ	0	11	9	0	0	0	0	0	0
ζ	93.3%			**100%**			**100%**		

- average time $\overline{T}$: average of running time (seconds) over 100 runs
- number of failures χ: number of failures in 100 runs

The agent performs the next episode if it reaches the target represented by the star. Learning stops when the accumulated reward of the last 15 iterations has a standard deviation below 0.5. The results are presented in Table 4.2 and plotted in Figures 4.5 and 4.6 for visual comparisons. The results presented for $Q(\lambda)$ technique have higher standard deviations compared to its opposition-based version. In addition, it seems that the benefit of opposition-based extension increases as the problem dimension increases.

Figure 4.5 presents changes in the overall average number of iterations for $Q(\lambda)$, $OQ(\lambda)$, and $NOQ(\lambda)$ algorithms for the three grids. It can be observed that the total number of iterations for convergence of $Q(\lambda)$ is far higher than $OQ(\lambda)$ and $NOQ(\lambda)$ algorithms. The $NOQ(\lambda)$ takes slightly less iterations than $OQ(\lambda)$. The reason for this may be related to the independency of $NOQ(\lambda)$ from next state for the opposite action.

Figure 4.6 presents the average time for $Q(\lambda)$, $OQ(\lambda)$, and $NOQ(\lambda)$ algorithms for the three grids. Even though the number of iterations for $OQ(\lambda)$ and $NOQ(\lambda)$ are almost the same, the average computation time of $NOQ(\lambda)$ is much lower than the average time

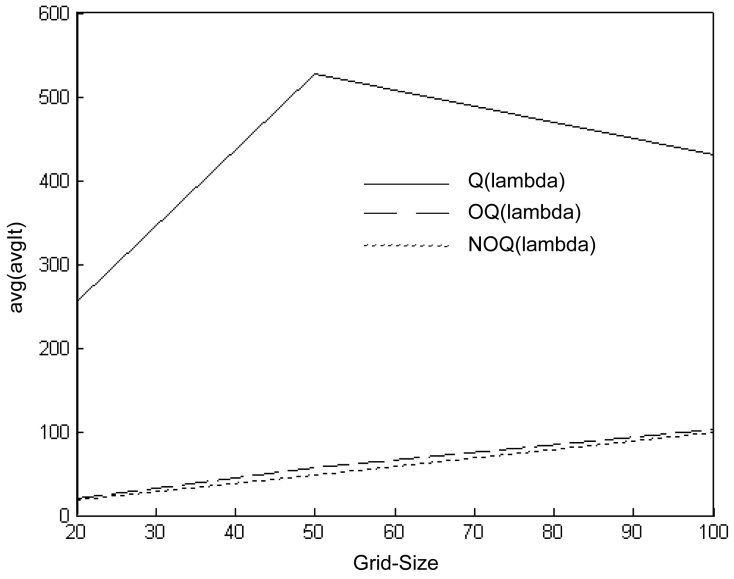

Figure 4.5: The average of average iterations $\overline{\overline{I}}$ for $Q(\lambda)$, $OQ(\lambda)$, and $NOQ(\lambda)$

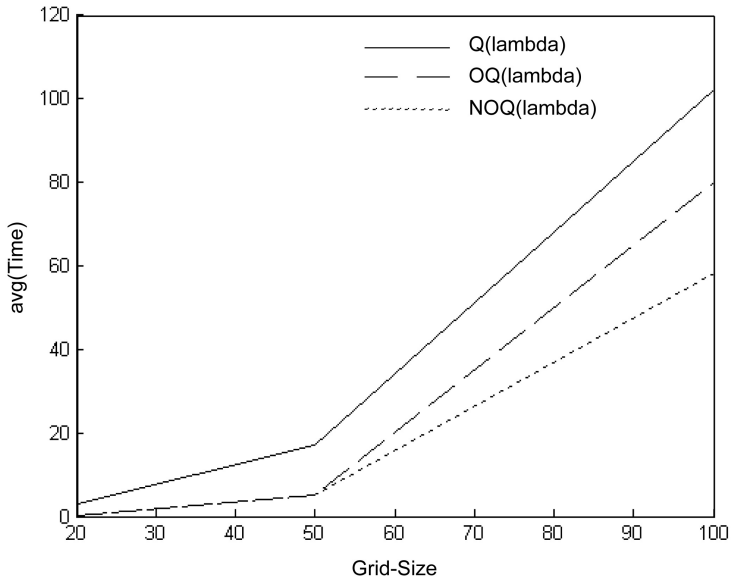

Figure 4.6: The changes of average time T for $Q(\lambda)$, $OQ(\lambda)$, and $NOQ(\lambda)$

of the $OQ(\lambda)$ for 100×100 grid. The reason is that $NOQ(\lambda)$ algorithm is more efficient than $OQ(\lambda)$ due to decrease in the computational overhead associated with updating the opposition traces.

In order to compare $Q(\lambda)$, $OQ(\lambda)$, and $NOQ(\lambda)$ we also need to consider that $Q(\lambda)$ failed[2] to reach the goal or target 20 times. To reflect this failure in the performance measure, the success rate $\zeta_{overall}$ for the algorithms is calculated:

$$\zeta_{overall} = (1 - \frac{\sum_{i=1}^{k} \chi_i}{\sum_{k} H}) \times 100, \tag{4.11}$$

where k is the number of grids tested (in this case $k = 3$), χ is the number of failures, and H is the number of times the code is run for each grid. Considering the convergence conditions, for the $Q(\lambda)$ algorithm the overall success rate is $\zeta_{overall} = 93.3\%$ because the agent failed to reach the goal 20 times. For the proposed algorithm $NOQ(\lambda)$, and the oppsotion-based algorithm $OQ(\lambda)$, the overall success rate is $\zeta_{overall} = 100\%$; They always successfully find the target and reach the goal.

4.3 Summary and Conclusions

Reinforcement learning (RL) can be considered as a goal-directed intelligent technique for solving problems in uncertain and dynamic environments by using reward and punishment. The RL agent learns from receiving reinforcement feedback as a reward or punishment from its environment. One of the concepts in reinforcement learning is the eligibility trace which is a bridge between TD techniques and Monte Carlo methods. The idea is that only eligible states or actions will be assigned a credit or blamed for the error.

[2]If the fixed numbers of iterations and episodes are not enough for the algorithm to reach the target (in one run) then we consider this as one failure.

The goal of this chapter was to introduce new techniques to expedite some of the RL methods for off-policy, step-by-step, incremental and model-free reinforcement learning with discrete state and action space. To solve this problem the concept of opposition-based reinforcement learning has been introduced. The general concept is that in Q-value updating, the agent updates the value of an action in a given state. If the agent knows the value of the opposite state or the opposite action, then instead of one value, the agent can update multiple Q-values at the same time without taking the associated opposite action or going to the opposite state. This should accelerate the learning process in general, and exploration in particular.

A variety of algorithms can be generated based on the concept of opposition to improve the learning and design faster RL techniques. Opposition is applied to create several algorithms using Q-learning. The $OQ(\lambda)$ has been introduced to accelerate $Q(\lambda)$ algorithm with discrete state and action space. The $NOQ(\lambda)$ method is an extension of $OQ(\lambda)$ to a broader range of non-deterministic environments. The update of the opposition trace in $OQ(\lambda)$ depends on the next state of the opposite action (which cannot be taken). This limits the usability of this technique to the deterministic environments because the next state should be detected by or known to the agent. $NOQ(\lambda)$ was presented to update the opposition traces independent of knowing the next state for the opposite action. The primary results show that $NOQ(\lambda)$ can be employed in non-deterministic environments and performs even faster (see Figure 4.6) than $OQ(\lambda)$.

In this chapter the effects of the opposition on the performance of $Q(\lambda)$ algorithm with respect to the number of iterations and running time have been studied for the grid-world environment. The next chapter will focus on the investigation of the $NOQ(\lambda)$ technique for three examples of dynamic environments.

Chapter 5

Oppositional Agents in Dynamic Environments

5.1 Introduction

Opposite actions have been introduced as a counter-concept to actions in the reinforcement learning model [73, 74, 77, 91, 90, 92]. This duality in concept has been implemented in the framework of $Q(\lambda)$ and contributes to the acceleration of the learning with regard to the number of iterations and running time. We have been able to develop the opposition-based Q(λ) by first proposing the concept of opposition trace in [73] and then solving several issues regarding the modeling of the problem by generalizing the algorithm [74]. In the previous technique, $NOQ(\lambda)$, the possible non-Markovian update of opposition traces was investigated where the next state for opposite action may not be available. However, the $NOQ(\lambda)$ was tested for a deterministic grid-world problem and it has not been implemented for non-deterministic environments [74].

In the previous chapter the opposition weight w was introduced. However, the effects

of the opposition weight was not investigated since the value of the opposition weight was set to one.

In the Section 5.2 the trade-off between exploration and exploitation will be studied by presenting the opposition weight as a function. The value of the opposition weight w increases as learning progresses and the number of steps increases. Hence, the positive effects of the opposition for the update of Q-values ($Q(s, \breve{a})$ where $\breve{a}$ is opposite action) increases.

In the Section 5.3 the $NOQ(\lambda)$ will be applied to the grid-based navigation. Three different weight functions are examined for different grid sizes with obstacle and walls (as parts of the environment) and the effects of weights will be discussed in this section.

In the Section 5.4 the $NOQ(\lambda)$ will be applied to the dynamic environment of elevator control. A simple case of the elevator control problem is selected for the experiments. The elevator control problem has several design challenges. Despite the continuous state and time, the elevator simulation is based on discrete events which imposes a challenge for simulation [13]. Another design challenge of the elevator control domain is the states of the environment which are not fully observable. Moreover the states are non-stationary because the arrival rate of passengers is not constant [13].

The $NOQ(\lambda)$ for elevator control problem with one elevator car will be implemented. The experiment will be kept simple by considering only one elevator car in order to demonstrate the positive effects of the opposition in the $NOQ(\lambda)$ algorithm and compare it with $Q(\lambda)$ technique without any training data.

The more complex version of the elevator control was presented by Crites et al. [13]. They demonstrated the application of the reinforcement learning for the elevator control. The elevator system simulated by Crites et al. was for a 10-storey building with 4 elevator cars. They have used feedforward neural networks to store Q-values because of the large

number of states and implemented a team of RL-agents in which every single agent controls one elevator car. The global RL signal was affected by the action of the agents, random arrivals of passengers, and incomplete state observation.

In this chapter, the intention is not to propose a new solution for elevator control, but employ this application, with a manageable state-space size to verify the improvement via $NOQ(\lambda)$ in comparison to $Q(\lambda)$.

Reinforcement learning methods have been successfully applied to difficult problem domains such as robotics [15, 34, 35], operation research [52], games [5, 67, 102], economics and marketing [1, 45], control [4, 47], human computer interaction [27, 80], image thresholding [75, 70, 69], simulation [81], and E-learning [72, 88]. In Section 5.5 the $NOQ(\lambda)$ will be employed to find the optimal threshold for an image. Image thresholding is a common segmentation technique which has many applications in computer vision. The states of the environment in the images can be dynamic because the intensity of gray levels can change due to changes in the illumination or noise in the image. The environment can also be non-deterministic because the next state of the environment may not be determined by the current state and action. "If the next sate of the environment is completely determined by the current state and the actions selected by the agents", then the environment is deterministic [66]. For instance, if the states of the environment are presented based on the gray levels in the image and a current action changes the current state by the amount of Δ then the next state which is a gray value may not even be available in the image. Hence, developing RL-based image thresholding is a challenging task. In the Section 5.5 the details of $NOQ(\lambda)$ algorithm for image thresholding will be discussed.

In all the applications in this chapter the performance of the $NOQ(\lambda)$ algorithm (with regard to the number of iterations and the running time) will be investigated and compared to the performance of the $Q(\lambda)$ method. It is demonstrated that the $NOQ(\lambda)$ algorithm

can successfully perform in non-deterministic and dynamic environments.

5.2 $NOQ(\lambda)$ in Dynamic Environments

Most of the early solutions to the approximate dynamic programming (ADP) involved using value iteration and exact model of the system. In contrast, reinforcement learning techniques have the advantage of solving ADP problems with or without using a model of the system [13]. Function approximation, good state definition, knowledge about parameters and model of the environment, learning model of the environment, or any knowledge about policy increases the range of applications for reinforcement learning.

The $NOQ(\lambda)$ employs the notion of opposition as a kind of knowledge of action representation for $Q(\lambda)$ technique. Actions are core components of reinforcement learning framework. The effect of some actions could oppose other actions. Hence, we consider them as opposite of each other. For instance we can consider "up/down", "left/right", and "plus/minus" as opposite pairs of actions.

The $NOQ(\lambda)$ takes into consideration such a dichotomy between actions in the RL framework. In the Q-learning technique there are two updates for $Q(s, a)$ and $Q(s, \breve{a})$ where $Q(s, a)$ is the expected infinite discounted return by taking action a in state s, and $Q(s, \breve{a})$ is the expected infinite discounted return for opposite action $\breve{a}$ in state s. The update for $Q(s, \breve{a})$ is applied without actually taking $\breve{a}$. It is based on the idea of considering opposite reinforcement signal for opposite action which was presented in the previous chapter.

The $NOQ(\lambda)$ also employs the idea of opposition trace and non-Markovian update. This chapter takes into account the challenging trade-off between exploration and exploitation.

The $NOQ(\lambda)$ technique was demonstrated in the Algorithm 7 (see Chapter 4 Section 4.1.2). However, the weight w provides a trade-off between exploration and exploitation for non-Markovian update. The second update in the algorithm which is also presented in Equation 5.1 takes advantage of the opposite reinforcement signal $\breve{r}$ and w to update the $Q(s, \breve{a})$:

$$Q(s, \breve{a}) \longleftarrow Q(s, \breve{a}) + w \cdot \breve{r} \cdot e(s, \breve{a}). \tag{5.1}$$

As was presented in the Algorithm 7, in addition to the update for $Q(s, a)$ we have another update for opposite actions in a given state s, $Q(s, \breve{a})$. Because such updates are based on step by step learning from reinforcement signal, we introduce the term w in the update formula (Equation 5.1).

w is a parameter introduced to impose a weight between 0 and 1 on the opposition update and can simply be set to a constant number. In [74] we set the weight w to 1 to simplify the update. In order to present w in a way to contribute to the exploration and exploitation process it must be increased as the iterations increase. The reason for this is that the agent can trust the state-action values more as it gradually moves toward the optimal values by visiting every state and taking every action several times. Therefore the weight w should be an increasing function in the second update (Equation 5.1). A simple way of defining the weight w is presenting it by a linear or quadratic function presented in the Equations 5.2 and 5.3 respectively.

$$w(\kappa) = \frac{\kappa}{\theta}, \tag{5.2}$$

κ is a variable for counting steps (iterations) of the learning process and θ is the total number of steps for all the episodes (constant number).

$$w(\kappa) = \frac{\kappa^2}{\theta^2}, \tag{5.3}$$

The equations are normalized in the implementation process to set the weight between 0 and 1 in order to prevent any bias toward opposition-based update for Q-values. We should bear in mind that increasing the order of the Equation 5.3 and increasing the value θ, decreases the values of w. Consequently, the values of w tend to approach zero and the effects of opposition-based update will diminish.

Since the new update presented in the Equation 5.1 depends on the parameter w and opposite reinforcement signal $\breve{r}$ (in contrast to the first update which depends on difference between Q-values in the present state and the next state), the convergence of the $NOQ(\lambda)$ can be satisfied by assuming that the parameters of the $NOQ(\lambda)$ technique such as α, γ, and λ are the same as $Q(\lambda)$. The Markov property is not violated in the second update because it is a non-Markovian update and the next state for the opposite action has not been considered in the update.

In this chapter the $NOQ(\lambda)$ algorithm is tested for three applications, grid navigation, elevator control, and image thresholding which will be presented in the following sub-sections.

5.3 $NOQ(\lambda)$ for Grid-Based Navigation

Barto et al. [79] use a two room grid environment to apply temporal abstraction technique in reinforcement learning. However, their approach is based on identifying subgoal states. Their "Two-Room" grid-world has two rooms of equal size and there is a wall with hole in the wall for passing agent. They have presented the environment as a 21×10 grid [79].

In this work the Two-Room grid-world has been slightly changed to produce more

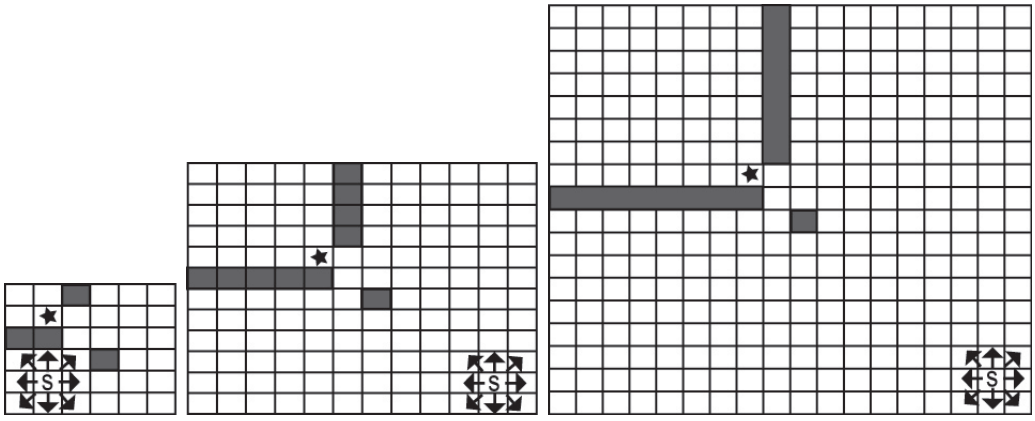

Figure 5.1: Two-Room grid-world environment for grid sizes 6×6 (left), 12×12 (middle), and 18×18 (right). S represents a random state and the star represents the target. The walls and obstacles are presented by gray cells.

challenge to the navigation of the agent. The size of the rooms are not equal, and there are two walls and one obstacle in the grid. The algorithm is tested for three grid sizes 6×6, 12×12, and 18×18 to present the results for grids with different number of states. Figure 5.1 presents the Two-Room grid-world environment for grid sizes 6×6, 12×12, and 18×18. If the size of the grid is presented by ω_1 (presenting the number of rows) and ω_2 (presenting the number of columns) then it is assumed that the target is located in the area with coordinate $(\frac{\omega_1}{2} - 1, \frac{\omega_2}{2} - 1)$ in all grids. The walls and obstacles are presented by gray cells in the Figure 5.1.

The grid environment presented in this application is deterministic and not dynamic. The algorithm will be tested for different weight functions, constant, linear, and quadratic. S represents a random state and the star represents the target. The agent begins its navigation from a random state s and in each time step it receives a scalar reinforcement signal after taking action. It uses a scalar for opposite reinforcement signal to update the Q-matrix for opposite action in a given state.

The reinforcement signals are presented by different scalar values $r_1 = 20$, $\breve{r}_1 = -5$,

Table 5.1: The Initial Parameters for all experiments

max. episodes	max. iterations	α	γ	λ
1000	100	0.3	0.2	0.5

$r_2 = 10$, and $\breve{r}_2 = -3$. If the agent hits the walls, obstacle, or the grid borders then it receives $\breve{r}_1$. In this scenario, the agent uses r_1 as a reinforcement signal for updating the opposite action. If the distance of the agent from the target decreases after taking an action then it receives r_2 as a reinforcement signal (reward). In this case the opposite reinforcement signal is $\breve{r}_2$ for updating the opposite action.

The walls and obstacles are presented by gray cells. The ϵ-greedy policy is applied for action selection by the agent. The parameters used in the algorithm are presented in the Table 5.1. Each cell of the grid presents one state of the environment. Hence, there are 36 states for 6×6 Grid, 144 states for 12×12 Grid, and 324 states for 18×18 grid. There are 8 actions to move to 8 possible directions, up, down, right, left, up-right, up-left, down-right, down-left. The opposite actions are defined based on the directions. Hence, (up,down), (right,left), (up-right,down-left), (up-left,down-right) are pairs of actions and opposite actions. The results will be discussed in the following subsection. The n_E presents the maximum number of episodes and I_{max} is the maximum number of iterations initialized at the beginning of the algorithm.

5.3.1 Experimental Results

The $NOQ(\lambda)$ algorithm introduced in Chapter 4 (see Algorithm 7 in Section 4.1.2) is used for the navigaion task. Three different weights are used for updating Q-matrix for opposite action in a given state. The linear (Equation 5.2) and Quadratic (Equation 5.3) functions

Table 5.2: The results for three measures of $\overline{\overline{I}}$, $\overline{E}$, and $\overline{T}$ for the $NOQ(\lambda)$-based navigation with three weights. The results are based on 15 runs. The ϵ-greedy policy is used for action selection.

	$NOQ(\lambda)$								
	W_1			W_2			W_3		
$X \times Y$	6×6	12×12	18×18	6×6	12×12	18×18	6×6	12×12	18×18
$\overline{\overline{I}}$	26 ± 16	16 ± 5	15 ± 3	23 ± 12	23 ± 8	19 ± 5	29 ± 18	26 ± 6	21 ± 2
$\overline{E}$	15 ± 8	130 ± 37	494 ± 75	21 ± 13	137 ± 65	615 ± 222	15 ± 12	163 ± 45	564 ± 142
$\overline{T}$	$\mathbf{0.4 \pm 0.06}$	$\mathbf{1 \pm 0.3}$	$\mathbf{8 \pm 1.5}$	0.4 ± 0.1	1.9 ± 0.8	15 ± 6	0.4 ± 0.1	3 ± 1.8	17 ± 7

as well as a constant value equal to the learning rate α are used to examine the effects of different weights on the learning process. The results are also compared with the results of $Q(\lambda)$ technique. The results for $NOQ(\lambda)$-based navigation is presented in the Table 5.2 where $\overline{\overline{I}}$ is overall average iterations, $\overline{E}$ is average number of episodes, $\overline{T}$ is average running time for the algorithm, W_1 presents a linear weight, W_2 presents a quadratic weight, and W_3 presents a constant weight which is equal to the learning rate α.

The results of the $Q(\lambda)$-based navigation are shown in the Table 5.3. In all the cases the agent successfully reaches the target for both $NOQ(\lambda)$ and $Q(\lambda)$ algorithms and all the grid sizes.

The results of $NOQ(\lambda)$ and $Q(\lambda)$ algorithms are compared for different weights, W_1, W_2, and W_3 and presented in the Figures 5.2, Figures 5.3, and Figures 5.4 respectively based on the measures presented in the Tables 5.2 and 5.3.

The number of iterations and episodes together affect the running time of the algorithm. The average running time is 0.4 seconds for $NOQ(\lambda)$ with W_1, W_2, and W_3 weights for 6×6 grid. The average time is 0.33 seconds for $Q(\lambda)$ technique for the same grid size. However, as the grid size increases, opposition-technique exhibits a more positive impact.

Table 5.3: The results for three measures $\overline{\overline{I}}$, $\overline{E}$, and $\overline{T}$ for $Q(\lambda)$. The results are based on 15 runs. The ϵ-greedy policy is used for action selection.

	$Q(\lambda)$		
$X \times Y$	6×6	12×12	18×18
$\overline{\overline{I}}$	15 ± 9	29 ± 15	29 ± 17
$\overline{E}$	28 ± 16	183 ± 71	504 ± 147
$\overline{T}$	0.33 ± 0.07	3 ± 1.3	22 ± 14

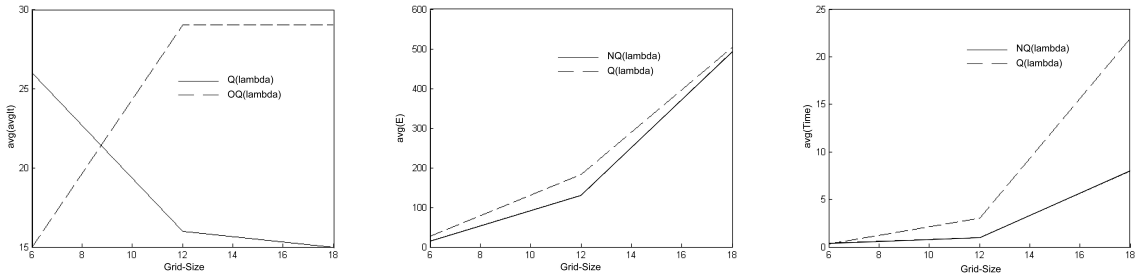

Figure 5.2: Left: Overall average iterations for $NOQ(\lambda)$ (with W_1) and $Q(\lambda)$ algorithms; Middle: Average episodes for $NOQ(\lambda)$ (with W_1) and $Q(\lambda)$ algorithms; Right: Average time for $NOQ(\lambda)$ (with W_1) and $Q(\lambda)$ algorithms

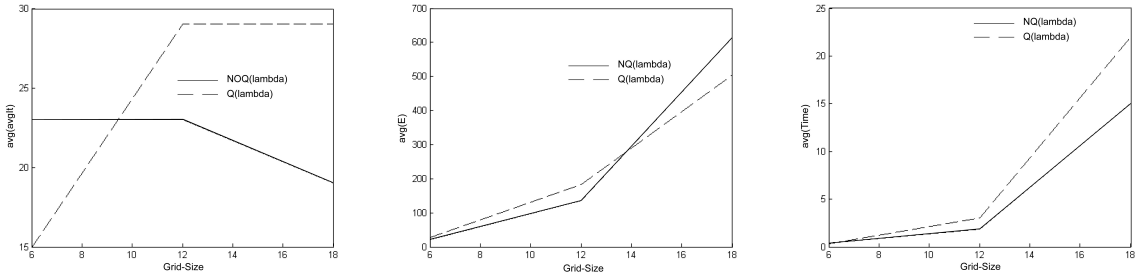

Figure 5.3: Left: Overall average iterations for $NOQ(\lambda)$ (with W_2) and $Q(\lambda)$ algorithms; Middle: Average episodes for $NOQ(\lambda)$ (with W_2) and $Q(\lambda)$ algorithms; Right: Average time for $NOQ(\lambda)$ (with W_2) and $Q(\lambda)$ algorithms

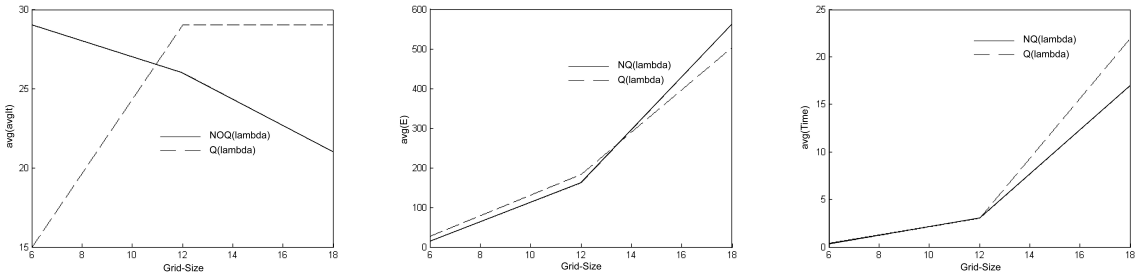

Figure 5.4: Left: Overall average iterations for $NOQ(\lambda)$ (with W_3) and $Q(\lambda)$ algorithms; Middle: Average episodes for $NOQ(\lambda)$ (with W_3) and $Q(\lambda)$ algorithms; Right: Average time for $NOQ(\lambda)$ (with W_3) and $Q(\lambda)$ algorithms

As presented in the results, the average time for $NOQ(\lambda)$ with W_1 and W_2 is less than the average time in the $Q(\lambda)$ technique for 12×12 and 18×18 grids. The average time in the $NOQ(\lambda)$ with W_3 is equal to the average time in the $Q(\lambda)$ technique for 12×12 grid. However, the average time for $NOQ(\lambda)$ with W_3 is less than the average time in the $Q(\lambda)$ method for larger grid size, 18×18.

5.3.2 Conclusions

The $NOQ(\lambda)$ algorithm was applied to the navigation task in the two room grid environment with walls and obstacle. The effects of three weights have been examined in this technique. The aim of using $NOQ(\lambda)$ is to provide a faster approach in terms of running time. The results demonstrated that the linear weight (W_1) saves more time comparing to other weight functions. Hence, in the following sections we employ the linear weight to demonstrate and discuss the implementation and efficiency (in terms of running time) of $NOQ(\lambda)$ algorithm for other challenging applications in dynamic and non-deterministic environments.

5.4 $NOQ(\lambda)$ for Simple Elevator Control

The elevator simulation of the proposed technique is for one elevator car [76]. The number of floors is defined by a variable that is initialized at the beginning of each run. Hence, the program could perform for different number of floors. Five variables contribute to the definition of states which are $ElLoc$, UH, UE, DH, and DE. These variables are described as follows:

- $ELLoc$ is a variable presenting the floor number in which the elevator car is located and can vary from one to the maximum number of floors.
- UH is a variable presenting the number of requests in the hallway floors located in the floors above the elevator car.
- DH is the number of hallway requests in the floors located in the same or lower levels of the elevator car.
- UE is a variable presenting the number of passengers' requests for the floors above the current position of the elevator car.
- DE is a variable corresponding to the number of passengers' requests for the floors in the same level of the elevator car or the lower floors.

The elevator control simulation was tested for a 5-storey building, hence, the number of states generated to represent the RL environment is 524. There are three actions, 'up', 'down', and 'stop' for changing the location of the elevator.

In each episode the elevator receives random requests from the passengers in the elevator car or people in the hallway. The elevator location is initialized randomly for each

episode. By using this information the state of the environment is determined based on the requirements of the state definition explained earlier.

Two matrices are defined for elevator car and the hallway to keep track of the requests. Whenever the elevator satisfies a request the value of the matrix corresponding to the hallway or elevator car will be set to zero. In contrast, whenever the elevator receives a request for a specific floor the value for the matrices in that floor will be set to one.

The priorities for the elevator to satisfy the requests are presented in terms of reward and punishment definitions. I use the term "current floor" for the floor that the elevator has stopped, "upper floors" for the floors above the elevator car level, and "lower floors" for the floors below the elevator car level. These priorities are described as follows:

- The first priority of the elevator is to satisfy the requests of the passengers in the elevator car for the current floor (if any passenger decides to leave the elevator in the current floor). The second priority is to consider the requests for upper floors (in the elevator car) if the number of these requests is more than the requests for lower floors.

- If the number of the requests for the lower floors is larger than the requests for upper floors in the elevator car then the first priority for the elevator is to pick up passengers in the hallway or let the passengers in the elevator leave the car (if they requested to leave) in the current floor. The second priority for the elevator is to satisfy the requests of the passengers for lower floors because the number of requests for the lower floors is more than the requests for upper floors.

- If the number of requests in the elevator car for the lower floors is equal to the upper floors then the number of requests in the hallway should be considered for the elevator priorities. If this number for the upper floors is greater than the lower floors

then the priority for the elevator is to pick up people in the upper floors, otherwise it should first pick up people in the hallway (in the current floor) or let the people leave the elevator if they requested. Then it should move to the lower floors to pick up passengers.

- If the number of hallway requests for the upper floors is equal to the requests for lower floors then the first priority is to satisfy the requests of the people for the current floor and the second priority is satisfying the requests of the people in the upper floors.

- If all requests are either for lower floors or for the upper floors (not for both upper and lower floors together) then the priority for the elevator is to satisfy those requests.

- If the elevator is in the top floor and takes upward action or if the elevator is in the first floor and it takes downward action then the position of the elevator will not be changed but elevator receives punishment for taking wrong action.

These priorities contribute to the definition of the reward and punishment for the elevator. Whenever the action taken by the agent does not satisfy the priorities then the agent receives punishment otherwise it receives reward. The reward and punishment are defined by a set of if/then rules. The reinforcement signal has two values, 10 and -1. If the agent receives reinforcement signal $r = 10$ then the opposite reinforcement signal ($\breve{r}$) is -1. If the agent receives the reinforcement signal $r = -1$ then the opposite reinforcement signal ($\breve{r}$) is 10.

The Boltzmann policy is applied for action selection. Boltzmann is a common action policy method based on using Boltzmann distribution which presents a probability $P(a)$ of taking action a in a given state s. The following equation is defined to be applied in the definition of Boltzmann equation:

$$\varpi = \left(\theta - \left(\frac{0.001 - \theta^2}{1 - \theta}\right)\right) \times \kappa + \left(\frac{0.001 - \theta^2}{1 - \theta}\right). \tag{5.4}$$

Here κ is a variable for counting steps (iterations) of the learning process and θ is the total number of steps across all episodes. Now the probability of taking each action is estimated by Boltzmann policy as follows:

$$P(a) = \frac{e^{\frac{Q(s,a)}{\varpi}}}{\sum e^{\frac{Q(s,:)}{\varpi}}}. \tag{5.5}$$

Hence, the action is determined as follows:

$$A_c = \{a | P(s,a) = \max_j P(s,a_j)\}, \tag{5.6}$$

where s is a given state, and $1 \leq j \leq N_a$ (N_a is the number of actions) and a is an action. If there are several actions available in A_c ($|A_c| > 1$), the agent will randomly choose one of them otherwise $a = A_c$.

In each iteration if the elevator satisfies all requests in the hallway and the elevator car, then the agent goes to the next episode to take new random requests.

Because the number of requests for the elevator car and the hallway is changing and it is selected randomly, the sates are not stationary. The states of the elevator environment are generalized by considering the number of requests for upper or lower floors comparing the current elevator position or the current floor. As stated by Crites et al. [13] "the desired destination of the passengers" (part of the state) in the hallway is not provided which is also the case in our simulation, hence, this part of state is not fully observable in our simulation.

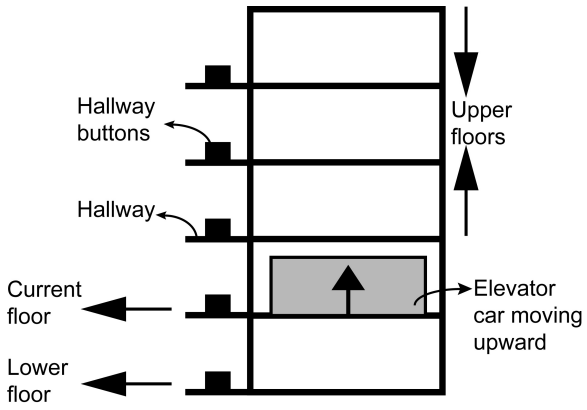

Figure 5.5: Elevator simulation for five-floor building; The black box presents the hallway buttons and the gray box presents the elevator car moving upward

As was presented earlier, the $NOQ(\lambda)$ algorithm is implemented for the elevator control. The action 'up' is considered the opposite of action 'down'. There is not any opposite action for action 'stop' hence in the update relation for the opposite actions (presented in the Equation 5.1) the update is applied just for one opposite action in the opposite action set $\breve{A}$.

The states of the elevator control are not fully observable and non-stationary which makes the elevator simulation challenging. The reinforcement learning applied to elevator control problem operates in a dynamic environment. The purpose of this research is not only to show that $NOQ(\lambda)$ method can successfully perform in this kind of environments but that it is also a faster alternative to $Q(\lambda)$ method. The experimental results presented in the next section confirm the superiority of $NOQ(\lambda)$ for the applications that at least one opposite action is available to the agent. Figure 5.5 illustrates the elevator simulation for a five-storey building.

5.4.1 Experimental Results of Elevator Control Application

The results of the elevator simulation are presented for a 5-storey building with one elevator car. The results of $NOQ(\lambda)$ are also compared with the results of $Q(\lambda)$. The initial parameters for the algorithms are $n_E = 500$, $I_{max} = 50$, $\alpha = 0.3$, $\gamma = 0.2$. These parameters represent maximum number of episodes, maximum number of iterations, learning step, and discount factor, respectively. The parameter λ is set to 0.5 for both $NOQ(\lambda)$ and $Q(\lambda)$ algorithms.

As was described earlier, in each iteration in the $NOQ(\lambda)$ and $Q(\lambda)$ algorithms if the agent satisfies all requests then the agent goes to the next episode. The average number of iterations for all episodes is measured for each run as well as running time of the algorithms. The simulation has been run 10 times to acquire average values. The results are presented in Table 5.4.

The average running time, $\overline{T}$, required for the proposed technique is $\mathbf{0.0455 \pm 0.0041}$ seconds and the overall average iterations, $\overline{\overline{I}}$ for this technique is $\mathbf{16 \pm 1}$. The average running time, $\overline{T}$, required for the $Q(\lambda)$ technique is 0.0774 ± 0.0119 seconds and the overall average iterations, $\overline{\overline{I}}$ for the $Q(\lambda)$ method is 29 ± 5.

The average running time required for $Q(\lambda)$ is 1.7 times higher than average running time for the $NOQ(\lambda)$ technique. The overall average iterations for the $Q(\lambda)$ is 1.9 times higher than the overall average iterations for the proposed technique. Hence, $Q(\lambda)$ requires almost double time and iterations to run (for the same number of episodes) compared with the proposed algorithm.

Figure 5.6 shows the average rewards versus the number of steps (iterations) for the $NOQ(\lambda)$ and $Q(\lambda)$ algorithms. Figure 5.6 also shows the convergence of the average rewards for both algorithms.

Table 5.4: The results of $NOQ(\lambda)$ and $Q(\lambda)$ techniques based on the running time (second) and the average iterations in 500 episodes; Numbers are averaged over 10 trials.

	$NOQ(\lambda)$									
T	0.0481	0.0459	0.0462	0.0424	0.0376	0.0502	0.0464	0.0510	0.0465	0.0412
$\overline{I}$	17	16	16	14	13	18	16	16	16	15
	$Q(\lambda)$									
T	0.0887	0.0591	0.0734	0.0840	0.0887	0.0824	0.0565	0.0714	0.0833	0.0868
$\overline{I}$	33	21	28	30	35	33	21	28	30	35

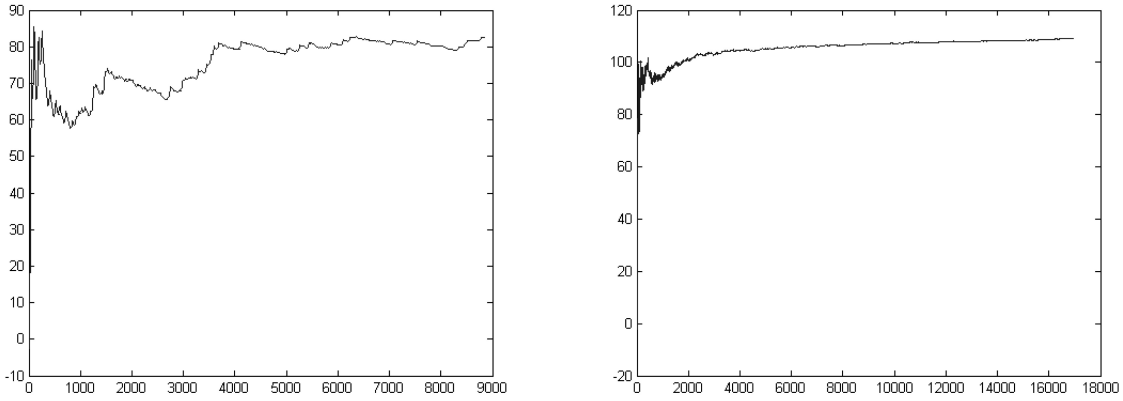

Figure 5.6: Left: Plot of average reward for $NOQ(\lambda)$; Right: Plot of average reward for $Q(\lambda)$ technique

5.4.2 Conclusions

In this section the $NOQ(\lambda)$ algorithm for a dynamic and non-deterministic environment is presented. I have considered the simple case of an elevator control problem for the experiments. The states of the environment in the elevator control simulation impose challenges for the reinforcement learning model. The states are non-stationary and not fully observable. The purpose of this work was not to propose a new solution for elevator control, but to employ this application, with a manageable state-space size and verify the improvement of $Q(\lambda)$ via $NOQ(\lambda)$.

The $NOQ(\lambda)$ technique is modified by using the increasing function $w(\kappa)$ employed for updating the Q-values of opposite action in a given state s, $Q(s, \breve{a})$ (Equation 5.1). This facilitates the tradeoff between exploration and exploitation in the opposition-based technique. As the learning steps increase, $w(\kappa)$ increases.

The results of the proposed technique are compared with the results of $Q(\lambda)$. The $NOQ(\lambda)$ can perform faster than $Q(\lambda)$ algorithm in terms of running time and average

number of iterations required for learning. Hence, the proposed algorithm can be used as an alternative to $Q(\lambda)$ whenever the opposite actions are available for the agent.

5.5 $NOQ(\lambda)$ for Image Thresholding

5.5.1 Introduction

Image thresholding is one of the common segmentation techniques in image processing. In numerous applications the foreground and background of the image must be separated to obtain a binary image containing the objects of interest. The challenge is how to find the optimal threshold under non-optimal circumstances. Finding an optimal threshold in order to segment digital images is a difficult task in image processing. A large number of image thresholding methods already exist in the literature, each providing good results for some specific image classes [70, 69, 71, 75, 64].

In previous research [69], the subjective and objective versions of RL techniques were applied and investigated for global image thresholding where the definition of states depended on the ratio of black pixels to the total number of pixels and number of objects in the image [69]. In order to define the states in a straightforward way we introduced "a reinforcement agent for threshold fusion" [75]. The idea behind this threshold fusion technique was the combination or fusion of different thresholding methods. In the cases that apparently no individual technique can be applied to all types of images, the fusion of some thresholding techniques could have promising results. The performance of a combination of several methods can yield better results provided a proper fusion of weighted thresholds specially for the cases that individual techniques deliver unsatisfactory results. We applied a reinforcement learning (RL) method with a fuzzy reward function to find the optimal weight for each method. This hybrid technique can detect the best combination

of thresholding techniques at hand.

However, if all thresholding methods fail to provide the appropriate threshold value due to the non-uniform illumination or noise in the image, then the fusion will not be able to deliver satisfactory results. Another issue that should be addressed is definition of states. In the previous work we defined the states based on the weighted average of the threshold values that each thresholding technique in the fusion delivers to the agent. However, a more effective way is to have a technique that gives the agent the opportunity to explore each gray level of the image as a state of the environment. Hence, the states are not limited to any specific method. Therefore, the design components of the RL thresholding agent should be revised. In the previous technique [75] the agent changes the weights of each thresholding technique to find the optimal fusion but the new approach will not be limited to any specific thresholding technique. Hence, the agent has a chance to explore every single gray level of the image presented as a state of the environment.

5.5.2 Background Literature

Thresholding is one of the segmentation techniques in image processing which separates the foreground and background of the image [10, 19, 64, 94, 62, 38]. In some applications, the gray-level images will be thresholded to acquire binary images. In these cases, the image contains a background and one or more objects. The generation of binary images is mainly for feature extraction and object measurement and recognition. Image thresholding can be regarded as the simplest form of segmentation, or in more general terms, as a two-class clustering procedure. In order to separate the object gray levels, g_O, from the background gray levels, g_B, a threshold, T, must be determined. The thresholding can be carried out by the following decision (L is the total number of gray levels in the image):

$$g_i(T) = \begin{cases} g_O = 0 & \text{for} \quad 0 \leq g_i \leq T, \\ g_B = 1 & \text{for} \quad T < g_i \leq L-1. \end{cases} \tag{5.7}$$

Generally, finding an appropriate threshold depends on the shape of the histogram, the image content and the application requirements [24, 101]. Illumination during image acquisition can have a major influence on thresholding. Poor and non-uniform illumination complicates the detection of an appropriate threshold.

Sankur and Sezgin [64] provide an extensive survey of over sixty thresholding techniques, in which the method according to Kittler [33] has found to be the best algorithm.

There are numerous threshold selection methods introduced in literature [64, 60, 71]. Thresholding methods could be divided into the several categories based on information they are using [64, 60, 71, 94, 62, 38]:

- Clustering and classification-based methods

  One of the most common thresholding techniques in clustering category [64] is Otsu's method [48, 71]. This method minimizes the weighted sum of within-class variances of foreground and background pixels to create the optimal threhsold [48]. Generally, clustering techniques are unsupervised techniques (without using prior knowledge) that generate separate regions in the image such that a pixel in a given region dose not belong to other regions [60]. On the other hand, classification techniques generate the classes in such a way that objects in the same class have high degree of similarity and objects belong to different classes have lower degree of similarity [60]. Usually, the classification techniques require training data with known labels [60, 71].

- Histogram shape-based methods

Histogram-based techniques are very practical for some applications in image segmentation. In multi-thresholding based on histogram the values in the range of histogram should be found to divide the image in to different regions. It should be noted that "the histogram is based on some global characteristics of the image, such as brightness distribution" [60].

- Entropy-based methods

  Sahoo, Wilkins, and Yeager presented a threshold selection method using entropy [63]. They used two probability distribution related to object and background. These probability distributions are derived from the original gray-level distribution of the image. They define the probability distributions of the object and background classes by using probability distribution of gray levels, P_0, P_1, P_2,..., P_{255}.

- Local, global and hybrid methods

  Local techniques are based on operation on the local parts of the image. The global image information will not be applied in these techniques. In contrast, the global methods are based on information in the whole image not the local characteristics. The features of the objects in the image are a useful source of information in these techniques [60]. Hybrid techniques are based on combination of local and global characteristics.

- Spatial and feature-based methods

  There are basic elements that can be used to represent the image such as features and spatial characteristics of the image. Any segmentation technique that employs certain features in the image and extracts or detects the homogeneous regions in feature space is a feature-based technique. Spatial-based techniques are constructed based on the

spatial characteristics of pixels in the image. In the spatial-based techniques the points of an object are considered spatially close which helps to better segment the image in many cases.

- Intelligent methods

  Machine learning techniques can be used for image segmentation as well. For instance, artificial neural networks [21] and reinforcement learning are two intelligent leaning techniques that have been employed to segment images. They both can be applied in different ways for segmentation tasks [60]. However, neural techniques need sample data for training but reinforcement learning can perform the required task without any training samples [70, 69, 71, 75].

- Attribute-similarity methods

  The similarity between the binarized image and original image could lead to better object detection. The edges, shapes, connectivity, or compactness could be the attribute of interest [64]. For many problems in image processing the subjective knowledge can be used to solve the problem. Fuzzy methods have the ability to represent subjective information. Fuzzy thresholding methods define a threshold as a fuzzy number [89].

There are more thresholding techniques which are discussed in [60, 71, 64]. The emphasize of this thesis is not on the thresholding methods rather it is about the applicability of opposition-based reinforcement learning for the real-world applications such as image thresholding. Hence, the details regarding thresholding techniques are not discussed in this thesis.

Ultrasound (US) imaging is one of the common modalities in medical imaging for soft tissues or the fluid filled parts in the body because it is based on sound waves [85]. Ul-

trasound imaging is not a harmful technique due to its radiation-free nature. Ultrasound imaging is a technique with the ability of helping to diagnose "breast lesions" and distinguish "cysts from solid breast tumors". US also has the ability to assist the doctors to "differentiate benign from malignant solid masses" [11]. Ultrasound images are noisy and difficult to segment. Image thresholding is a segmentation technique and has a crucial role in many applications. Image thresholding increases the contrast of the image specially foreground and background of the image and makes the image more understandable. The detection of the object of interest will be easier after thresholding because of increasing the contrast of foreground and background.

Non-uniform illumination, noise, lack of clear contrast in the objects and other factors contribute to segmentation errors such as under-segmentation and over-segmentation. If there are regions in the image that appear as one region after segmentation there is under-segmentation. In this situation the "full segmentation has not been achieved" [60]. If a region in the image splits into several parts after segmentation then we have over-segmentation. Solving over- and under-segmentation is reported to be a difficult task and can be solved based on the specification of the application [60].

Sahba et al. [61] employ the opposition-based Q-learning in image segmentation to threshold Transrectal Ultrasound (TRUS) images (for prostate cancer detection). Reinforcement learning is reported as an alternative approach for learning-based segmentation techniques in the cases that there is not sufficient number of training data available [75, 61]. As mentioned earlier in Chapter 4, opposition-based Q-learning may suffer from reward/punishment confusion. The opposition-based $Q(\lambda)$ was introduced as a cure for this problem because of using opposition trace and making updates for Q-matrix for all the opposite actions in all the sates [73]. Hence, in this chapter we have applied $NOQ(\lambda)$ technique [74, 76] for the thresholding of breast ultrasound images.

5.5.3 $NOQ(\lambda)$ Image Thresholding

The definition of the reinforcement learning components are different in this work comparing the components of RL-based thresholding presented in [75] and [61]. In the work presented in [75], a reinforcement agent is designed to learn the optimal weights for thresholds delivered by different algorithms and segment the image globally by applying the fused threshold. However, if all the techniques fail to find a good threshold for the image due to the noise or the poor resolution, the fusion may not yield to satisfactory results which is the case for many ultrasound images. In the work presented by Sahba et al. [61] the states of the environment are represented by some parameters quantifying the quality of the image. They have used features such as area, compactness, number of objects, etc. Sahba et al. emphasize that the choice of these parameters and features depends on the application. Here we use a more universal approach to define the states of the environment. The sates in this work are based on gray levels in the image. The actions suggested by Sahba et al. [61] change the threshold value and the size of structuring elements for each sub-image. In this research the actions change the thresholds and not the structuring elements because the features are not presented as states and consequently the agent does not need to change the size of structuring elements. Sahba et al. calculate reward and punishment by comparing the result with its associated gold standard image (which is thresholded by an expert). In this work the immediate reward and punishment are calculated based on gold standard image and other measurements which will be described later.

A reinforcement agent is designed to learn the optimal threshold for the image. The proposed approach is a weakly supervised technique for image thresholding. The structure of the method is presented in Figure 5.7.

The first step of the algorithm is to read the image and its associated gold standard image. The gold standard image is the image which has been manually thresholded by the

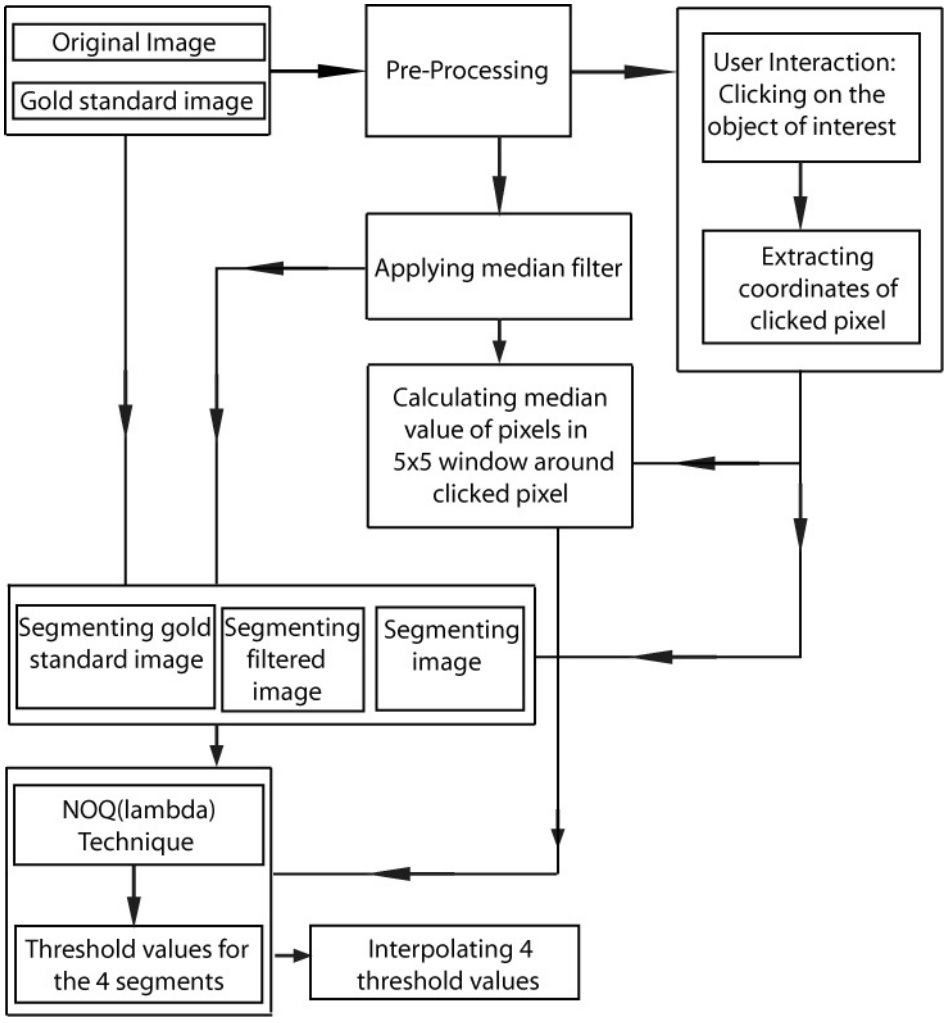

Figure 5.7: Overall structure of the $NOQ(\lambda)$-based image thresholding

expert and used to define the immediate reinforcement signal for the agent [60, 61]. One of the advantages of using reinforcement learning comparing to supervised learning methods is that this technique requires only a few or no training examples [60, 61, 75].

The next step in the algorithm is applying pre-processing techniques. In the proposed technique the λ-enhancement technique is applied [87]. This method aims to provide optimal contrast enhancement for digital images. It is based on a new class of fuzzy membership functions to detect the suitable modification of gray levels. The technique is constructed on involutive membership functions as follows [87]:

$$\mu^*_\lambda(g) = \frac{\mu(g).(1+\lambda)}{1+\lambda\mu(g)} \qquad g = 0, L, L-1. \tag{5.8}$$

$\mu(g)$ is initial membership value and L is number of gray levels. Tizhoosh et al. [87] have generated images with "different level of brightness" using monotone membership functions and varying λ. Extreme values for λ are gained when $\lambda \to -1$ (generating black image) or $\lambda \to \infty$ (generating white image). In the extreme cases the ambiguity of gray levels reduces. Tizhoosh et al. [87] state that "images in the middle of the sequence are more suitable for human perception". I should emphasize that pre-processing (e.g. enhancement techniques) and post-processing methods are not investigated in this research. The focus of this research is to introduce the opposition concept for reinforcement learning and investigate the implementation and advantages of opposition for selected problems that can be solved using $Q(\lambda)$ technique.

RL agents can be employed to design a personalized system to adapt to human intention, intuition, needs and requests [72, 88]. The user can provide his/her requests, responses, and reactions for the computer by interacting with intelligent agent. This yields the most efficient system that can perform challenging tasks, save the user's time, and prevent user tiredness and confusion. Humanized computational intelligence is a new and

challenging direction in computational intelligence. Mixed-initiative interaction represents a link between AI (artificial intelligence) and human-computer interaction, and refers to a flexible interaction strategy in which each agent (user or computer) contributes what it is best suited for at the most appropriate time [23].

In this technique the mixed initiative interaction is employed to provide a feedback for the agent and increase the chance of detecting the object of interest and learning the optimal threshold by the agent. Otherwise, the poor resolution of ultrasound images will have a negative effect on the learning process of RL agent.

After pre-processing the image will be presented to the user. User clicks on the object of interest. Then, the coordinates of the clicked pixel are extracted. The clicked pixel is used as a pivot for calculating reward and punishment and segmenting image to four sub-images.

After pre-processing a separate copy of the pre-processed image is filtered by median filter to reduce the noise. Then each of the filtered image, gold standard image and the pre-processed image are divided into four sub-images based on the coordinates extracted from the clicked pixel. Each preprocessed sub-image with its corresponding sub-gold image and sub-filtered image will be provided to the $NOQ(\lambda)$ algorithm. The algorithm learns the optimal threshold value for each sub-image separately (Figure 5.8).

The median value in a 5×5 window around the central clicked pixel is also calculated which will be provided for the $NOQ(\lambda)$ technique for calculating reinforcement signal. As mentioned earlier, the $NOQ(\lambda)$ algorithm is applied for learning the optimal threshold for each sub-image. The states are defined based on the gray level values of the image. In other words, each gray level value represents a single state of the environment. Hence, the number of states is varying from one image to the other one and is equal to the number of gray level values in the image.

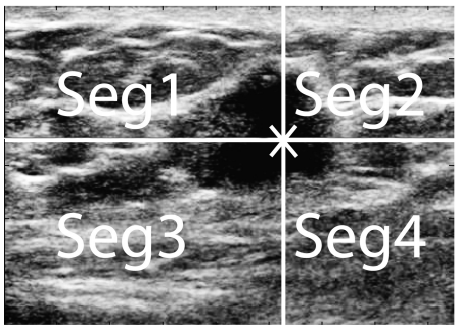

Figure 5.8: Expert user clicks on the object of interest on the image (marked by a cross) which divides the image into four sub-images

The states are stored in a vector with two rows. The first row indicates the state numbers and the second row is the gray level values associated to that state. To calculate the states for each image the pixels are sorted (the repeated gray levels are also eliminated). The MATLAB function, "unique", is used to return the same gray level values in the image but without repetitions. It also sorts the gray level values in the image.

Actions and opposite actions change the state of the environment (gray values). It should be emphasized that the goal of the agent is to find the optimal or near optimal policy to mark the best gray level value as a threshold by considering accumulated rewards generated by reinforcement signal in each step of the episode. In the implementation of the proposed technique[1], 53 actions and opposite actions are responsible for changing the threshold values. They are defined arbitrarily as $g+0, g+1, g-1, g+10, g-10, g+20, g-20, ..., g+250, g-250$ assuming that g is a gray level value in a given state. It is assumed that if the agent takes the action a_i where i is an even number then a_{i+1} is its associated opposite action. If the agent selects the action a_i where i is an odd number, then a_{i-1} is its associated opposite action. Hence, the set of even numbers is presenting the actions and

[1]The implementation process is described here because the emphasis of this work is introducing the concept of opposition for the selected RL-based algorithms and discussing how the opposition can be employed in the components of RL-techniques

the set of odd numbers contains their associated opposite actions and vice versa. There is not any opposite action for a_1 (a_1 is $g + 0$) in this scheme.

The reinforcement signal is a scalar number quantifying reward or punishment. If the consequence of an action is a gray level which is not available in the state vector then the agent should stay in the current state (and not go to the next state) but receive punishment (= -10) for that action and reward (= 10) for the opposite action. It should be emphasized that agent would not know the next state of the opposite action in the $NOQ(\lambda)$ technique. If the consequence of an action is a gray level which is presented in the state vector then the reinforcement signal is calculated based on the thresholded result and is a combination of three measurements.

As mentioned earlier the agent filters a separate copy of the pre-processed image with the median filter and divides it into four sub-images which are given to the $NOQ(\lambda)$ algorithm for calculating reinforcement signal. Then each of these sub-images are normalized to have values in the interval [0,1]. If matrix N presents the normalized smooth image then $M = 1 - N$ is a matrix of membership values for object pixels. These values will be used in calculating the reinforcement signal.

There are three measures applied to calculate reinforcement signal. One of them is a measure reported in the author's previous works [71, 75] where it was assumed that there are no ground-truth images available for the training, hence, a possible way to assess the agent performance during the training is to compare the binary image with membership matrix corresponding the original gray-level image. Therefore, the agent calculates a fuzzy dissimilarity measure F_{dissim} as follows:

$$F_{dissim} = \sum \sum |B - M|, \tag{5.9}$$

where B is the binary image and M is the membership matrix generated earlier. The

values of M which are less than a constant membership value ψ are set to zero. The agent must be configured with the appropriate value of ψ based on image content. The difference between the binary value and the membership values is a convenient way to quantify the discrepancy between black-and-white objects and gray-level image. The agent receives the fuzzy reward and updates the Q-matrix. The fuzzy reward is defined as follows:

$$r_1 = \frac{1}{F_{dissim}+\delta}, \tag{5.10}$$

where δ is positive constant (e.g. $\delta = 0.001$) and F_{dissim} is the fuzzy dissimilarity provided earlier. Therefore, r_1 presents a fuzzy similarity between the binarized image and the original image; the higher the similarity, the more reward the agent will receive. In this case the measure that will be applied for calculating opposite reinforcement signal is $-r_1$.

Another measurement which is applied in evaluating agent's action and calculating reinforcement signal is based on comparing the binary result with its associated gold standard image [61]. The gold standard image is also divided into four sub-images. The binarized segmented image now can be compared with the associated gold standard sub-image. The Hamming distance H [20, 22] between the foreground of the gold standard sub-segment image and the foreground of the thresholded sub-image[2] is calculated.

The Hamming distance H presents the difference between two parts. Hence, $r_2 = 1/H$ represents a measure that is used for calculating reinforcement signal and $-1/H$ is used for calculating opposite reinforcement signal. The value of H can be set arbitrarily. In this work, the constant numbers are used to define H. If $H = 0$, then $r_2 = 200$ and its associated opposite reward $\breve{r}_2$ is -200.

The third measure is based on median value *med* in the 5×5 window around the central

[2]The foreground of thresholded sub-image is the parts witch is related to the object of interest and is determined by using MTLAB function, "bwlabel".

Table 5.5: The Initial Parameters for all experiments

n_E	I_{max}	α	γ	λ	ψ
1000	20	0.3	0.2	0.5	0.99

clicked pixel calculated earlier and can be given as $r_3 = \frac{1}{\|med-T\|}$. Its associated opposite value is presented by the equation $\breve{r}_3 = -r_3$.

The reinforcement signal is a combination of r_1, r_2, and r_3 and is presented by $r = \sigma \times r_1 + r_2 + r_3$ where $0 < \sigma < 0.1$. The effect of r_1 is reduced because it depends on the initial constant membership value ψ. This value can be changed based on the application. Hence, in order to reduce the dependency of the proposed technique from a new variable, the weight of r_1 is less than the other measures. The opposite reinforcement signal is $\breve{r} = -\sigma \times r_1 - r_2 - r_3$ which is equals to $-r$.

The choice of reinforcement signal may vary depending on the application and the availability of image features for implementation. In this work the effort is to reduce the dependency of the technique from the features and keep the implementation general and independent from the application at hand.

The initial parameters for the algorithms are presented in Table 5.5. The weight w is linear which is based on Equation 5.2. The ϵ-greedy policy is applied for action selection by the agent.

At the end of the learning, agent has four learned threshold values associated with four sub-images. We can interpolate these threshold values to generate thresholds for each pixel. Then the thresholded image will be presented to the user based on interpolated values.

The mixed initiative interaction has a positive effect on the performance of the algorithm. However, we need to reduce or simplify the tasks for the user to prevent any

tiredness and confusion for the user. At the beginning of the learning the user is engaged by providing one click on the object of interest on the image. At the end of the learning and before interpolation we can also have user interactions to verify which of the four threshold values are acceptable and which ones are not. Therefore, at the end of the learning, the algorithm can be slightly changed. The modified algorithm is depicted in Figure 5.9. The agent applies the learned threshold values to their associated pre-processed sub-images (without interpolation) and presents the result to the expert. After that, the expert selects the proper thresholds. Then the agent calculates average of the proper thresholds. Subsequently, the improper threshold values will be replaced by the calculated average value of the proper thresholds. At the end, we can again interpolate the four threshold values, threshold the image and present the binary image to the user.

5.5.4 Experimental Results

Reinforcement learning has been successfully employed to threshold images for the applications that the sufficient training samples are not available [60, 70, 69, 71, 75, 61]. The goal of the proposed technique is to apply an opposition-based reinforcement learning technique to perform faster during the learning process. Meanwhile, a new implementation is presented here in terms of implementing the components of $NOQ(\lambda)$ algorithm such as states, actions, and reinforcement signal. It generates a more universal approach which is less dependent on image features comparing with the work reported in [61].

In order to verify the advantage of using opposition for RL-based image thresholding, the results are presented and compared for two RL algorithms $NOQ(\lambda)$ and $Q(\lambda)$. The results show three images, Breast Implant Hematoma[3], Intraductal Carcinoma[4], and Post

[3]Hematoma is "a mass of usually clotted blood that forms in a tissue, organ, or body space as a result of a broken blood vessel" [44].

[4]Carcinoma is "a malignant tumor of epithelial origin" [44].

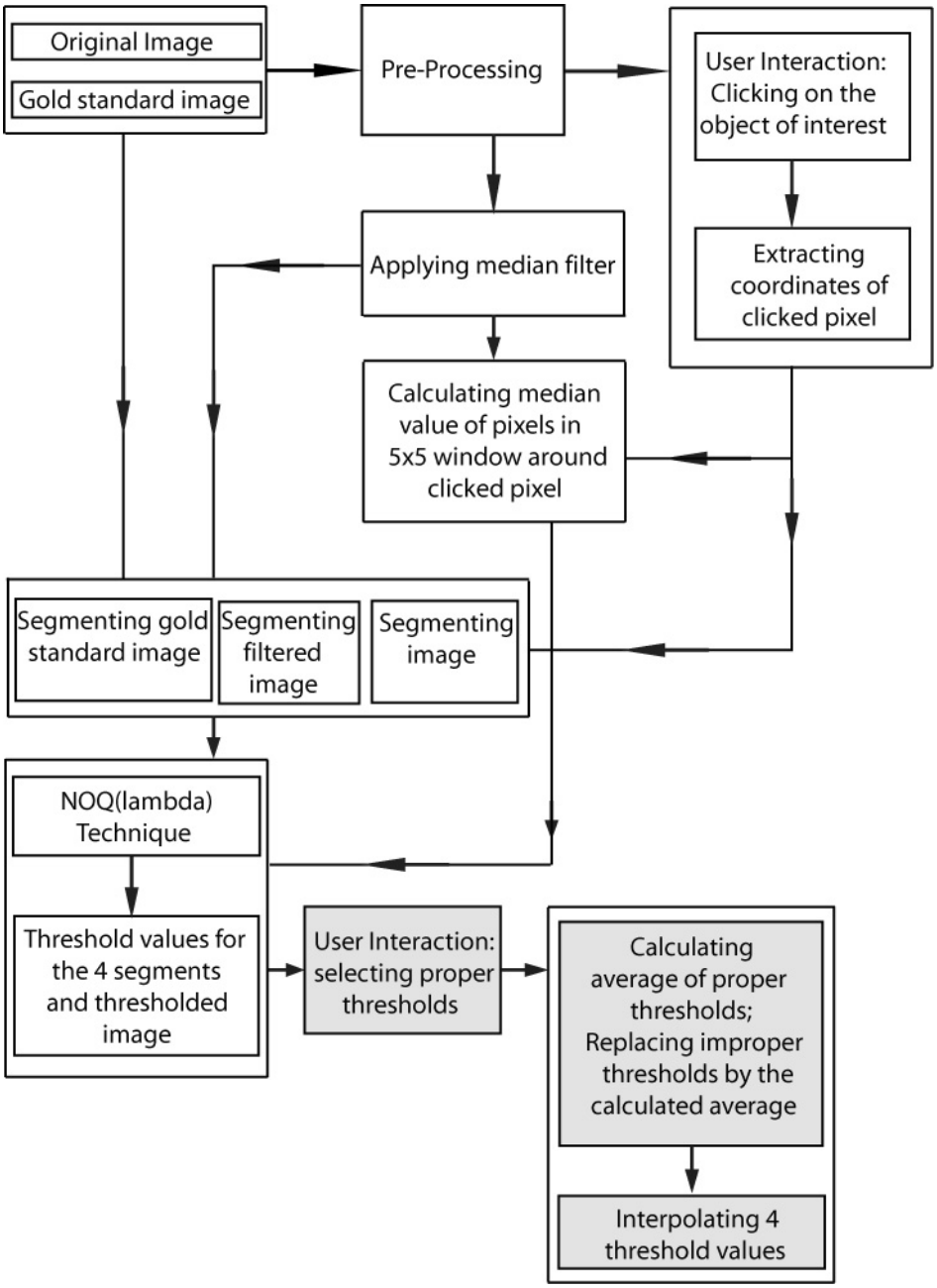

Figure 5.9: Overall structure of the $NOQ(\lambda)$-based image thresholding using expert feedback to improve the results; The gray boxes present the difference of this algorithm with the technique presented in Figure 5.7

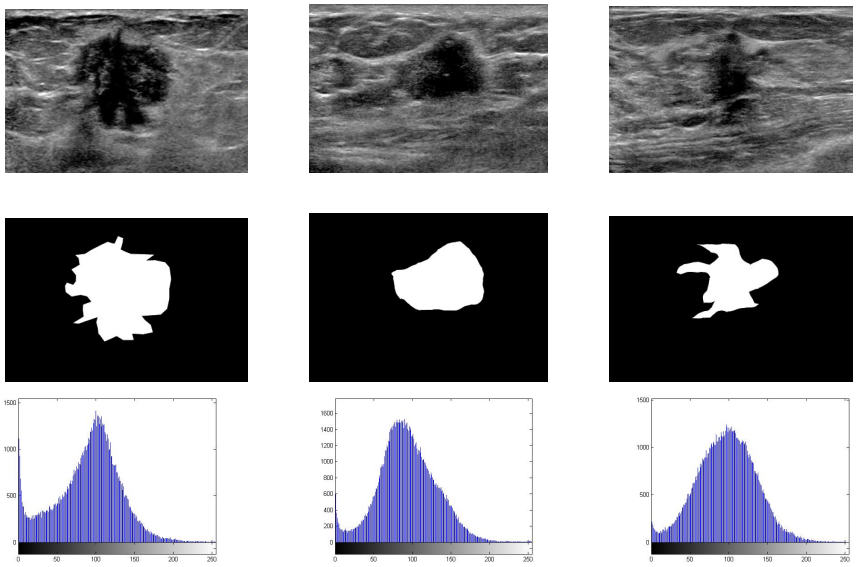

Figure 5.10: First row: Original images (Left - Image1 - Breast Implant Hematoma; Middle - Image2 - Intraductal Carcinoma; Right - Image3 - Post Surgical Breast Scar); Second row: The associated gold standard images of the first row; Third row: Histograms of the images

Surgical Breast Scar. The original images, their associated gold standard images, and the image histograms are presented in the Figure 5.10.

The number of states is 182 for Image1, 177 for Image2, and 180 for Image3 (based on the gray level values in the image). As a quantitative performance measure, the ratio of Hamming distance $H(I_B, I_G)$ [20, 22] between the binary image I_B and the gold standard image I_G to the number of pixels N_{im} for each image is calculated as a dissimilarity measure χ:

$$\chi = \frac{H(I_B, I_G)}{N_{im}}. \tag{5.11}$$

Results of $NOQ(\lambda)$ are presented in the Table 5.6 and compared with the results of $Q(\lambda)$. Each algorithm has been run 10 times. The average running time $\overline{T}$ and the average dissimilarity measure $\overline{\chi}$ is calculated for each technique. The overall average iterations and average episodes required for the convergence of the algorithms are observed for each segment of the image. $\overline{\overline{I}}_i$ and $\overline{e}_i$ represent the overall average iterations and average episodes for each segment.

The results of the proposed thresholding algorithm based on $NOQ(\lambda)$ are presented in Figures 5.11 and 5.13. The results of the proposed thresholding method based on $Q(\lambda)$ are presented in Figures 5.12 and 5.14. For all the figures (5.11, 5.12, 5.13, 5.14) columns from left to right are the results corresponding to Image1, Image2, and Image3 respectively and each row demonstrates the results for one run.

As apparent in the Table 5.6, the average running time of the proposed algorithm (algorithm in Figure 5.7) with the $NOQ(\lambda)$ is less than average running time of the same algorithm with $Q(\lambda)$. In the results of thresholding with $NOQ(\lambda)$, either the overall average iterations or the average episodes for the segments are less than the overall average iterations or the average episodes of the same technique with $Q(\lambda)$ (for all the images). The only exception is segment 3 of the image3. However, the average running time of the $NOQ(\lambda)$-based thresholding is less than $Q(\lambda)$-based technique for all the cases.

The other measure introduced is χ (see Equation 5.11) which is the relative inaccuracy of the technique. As shown in the Table 5.6, there is a small difference between $Q(\lambda)$ and $NOQ(\lambda)$ thresholding techniques. For the Image1 the inaccuracy is 0.23 for $NOQ(\lambda)$ and 0.26 for $Q(\lambda)$, for Image2 this is 0.23 for $NOQ(\lambda)$ and 0.22 for $Q(\lambda)$, and for the last Image it is 0.31 for $NOQ(\lambda)$ and 0.29 for $Q(\lambda)$. The $NOQ(\lambda)$ can perform faster by keeping almost the same level of accuracy as $Q(\lambda)$.

As mentioned before, user can play an important role to improve the results at the end

Table 5.6: Results of the $NOQ(\lambda)$ and $Q(\lambda)$ for thresholding of breast ultrasound images

	$NOQ(\lambda)$			$Q(\lambda)$		
Images:	Image1	Image2	Image3	Image1	Image2	Image3
$\overline{\overline{T}}$	**658±116**	**707±209**	**739±196**	876 ± 210	902 ± 305	779 ± 177
$\overline{\chi}$	**0.23±0.07**	0.23 ± 0.09	0.31 ± 0.10	0.26 ± 0.06	**0.22±0.12**	**0.29±0.09**
$\overline{\overline{I}}_1$	10 ± 0.4	10 ± 0.3	10 ± 0.1	11 ± 0.4	10 ± 0.3	10 ± 2.1
$\overline{\overline{I}}_2$	10 ± 0.2	10 ± 0.3	10 ± 0.2	11 ± 0.4	11 ± 0.3	10 ± 2.2
$\overline{\overline{I}}_3$	10 ± 0.4	10 ± 0.2	10 ± 0.2	11 ± 0.4	11 ± 0.3	10 ± 0.3
$\overline{\overline{I}}_4$	10 ± 0.3	10 ± 0.2	10 ± 0.2	11 ± 0.4	10 ± 0.4	11 ± 0.3
$\overline{e}_1$	421 ± 112	502 ± 231	503 ± 148	627 ± 192	578 ± 195	572 ± 211
$\overline{e}_2$	369 ± 78	386 ± 112	429 ± 119	383 ± 150	415 ± 86	466 ± 145
$\overline{e}_3$	472 ± 99	423 ± 117	511 ± 233	444 ± 205	500 ± 157	424 ± 128
$\overline{e}_4$	401 ± 95	375 ± 88	419 ± 123	427 ± 141	434 ± 117	442 ± 89

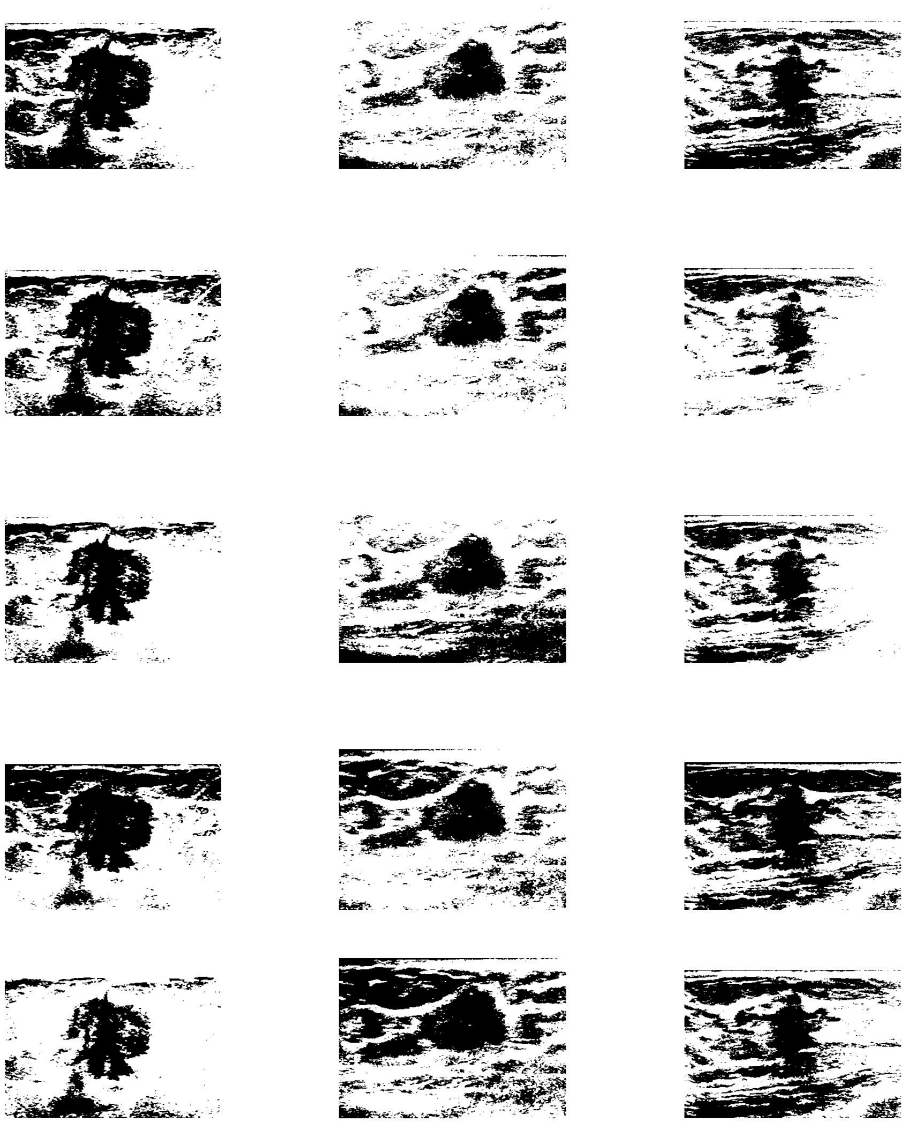

Figure 5.11: From left to right: Results based on $NOQ(\lambda)$ algorithm for Image1, Image2, Image3

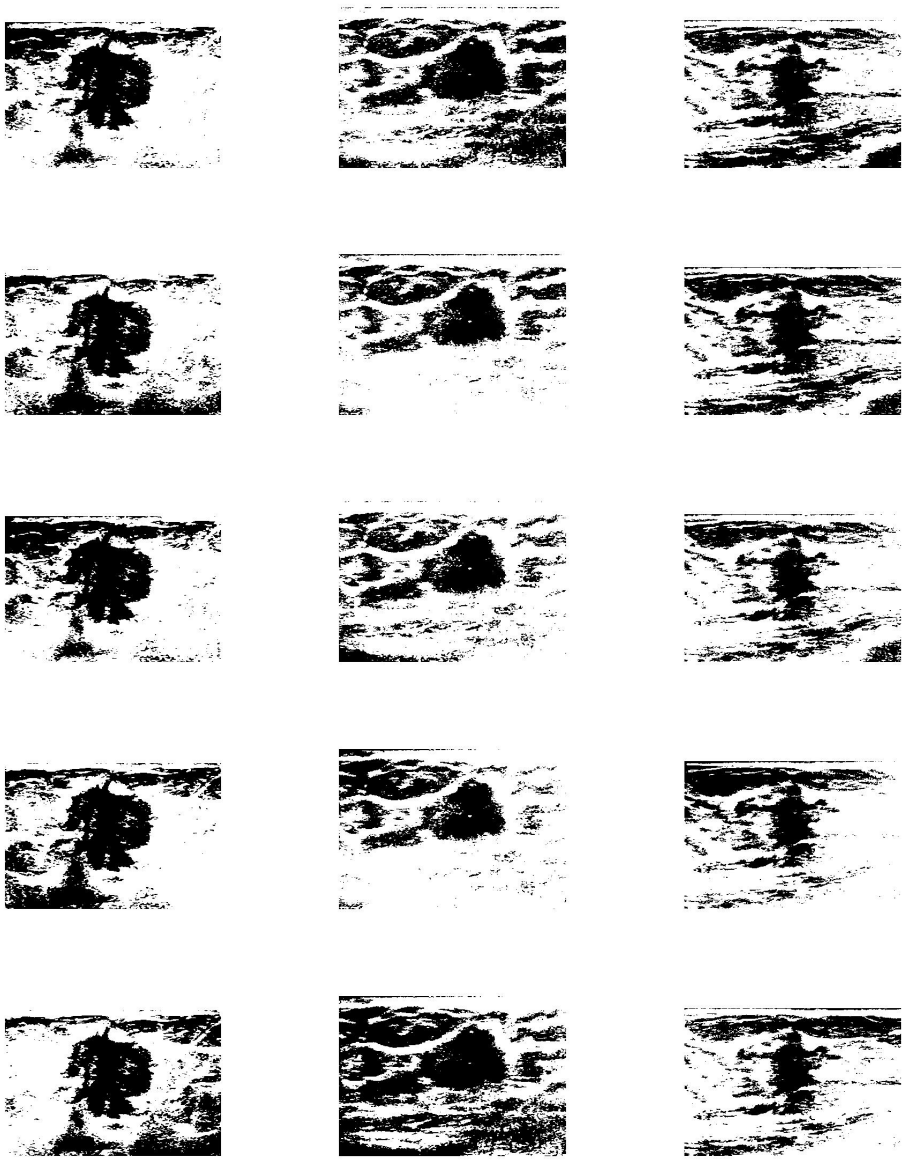

Figure 5.12: From left to right: Results based on $Q(\lambda)$ algorithm for Image1, Image2, Image3

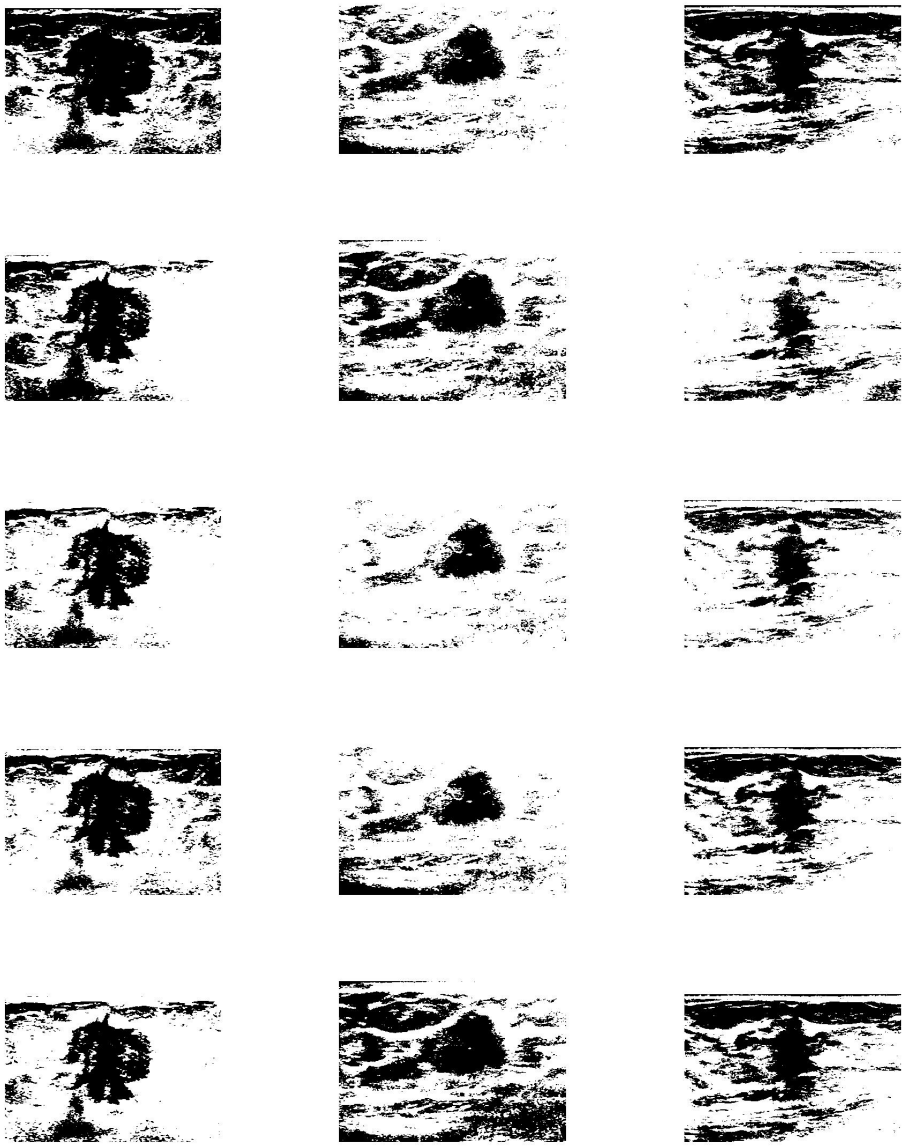

Figure 5.13: From left to right: Results based on $NOQ(\lambda)$ algorithm for Image1, Image2, Image3

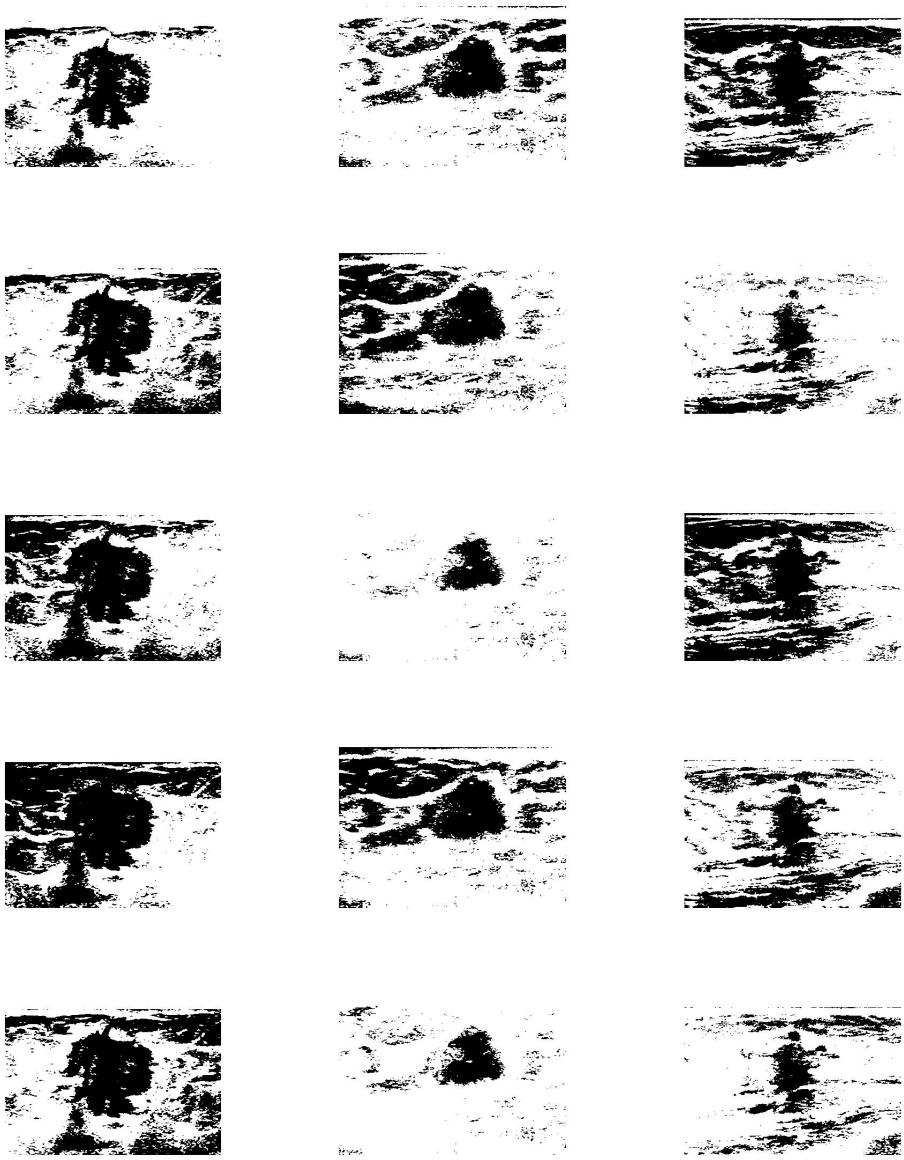

Figure 5.14: From left to right: Results based on $Q(\lambda)$ algorithm for Image1, Image2, Image3

of the learning. For this reason the algorithm presented in the Figure 5.9 is tested for the same images by using $NOQ(\lambda)$. The results are presented in Figures 5.15, 5.16, 5.17, and 5.18. Columns from left to right are the results corresponding to Image1, Image2, and Image3 respectively and each row demonstrates the results for one run. Figures 5.15 and 5.17 illustrate the images after learning without interpolation. Figures 5.16 and 5.18 are the corresponding images after user intervention, respectively.

For the first column (Image1 in Figure 5.15) from top to bottom the segments that are replaced (improper segments) are Seg1-Seg3, Seg1-Seg2-Seg3, Seg1-Seg3, Seg1-Seg2, Seg3.

For the first column (Image1 in Figure 5.17) from top to bottom the segments that are replaced (improper segments) are Seg1-Seg2-Seg4, Seg1-Seg3, Seg1-Seg3, Seg1-Seg2, Seg1-Seg2-Seg3.

For the second column (Image2 in Figure 5.15) from top to bottom the segments that are replaced (improper segments) are Seg2-Seg3, Seg2, Seg3-Seg4, Seg1-Seg4, Seg1-Seg2-Seg3.

For the second column (Image2 in Figure 5.17) from top to bottom the segments that are replaced (improper segments) are Seg1-Seg3, Seg1-Seg3-Seg4, Seg1-Seg3, Seg2-Seg3, Seg1-Seg3-Seg4.

For the third column (Image3 in Figure 5.15) from top to bottom the segments that are replaced (improper segments) are Seg3-Seg4, Seg1, Seg1-Seg3, Seg1-Seg2-Seg3, Seg1-Seg3-Seg4.

For the third column (Image3 in Figure 5.17) from top to bottom the segments that are replaced (improper segments) are Seg1-Seg2-Seg3, Seg2-Seg3-Seg4, Seg1-Seg2, Seg1-Seg2-Seg3, Seg1-Seg2-Seg3.

The elimination of improper thresholds improves the accuracy of the technique by decreasing $\overline{\chi}$ from 0.23 ± 0.07 to 0.13 ± 0.05 for the first image, from 0.23 ± 0.09 to

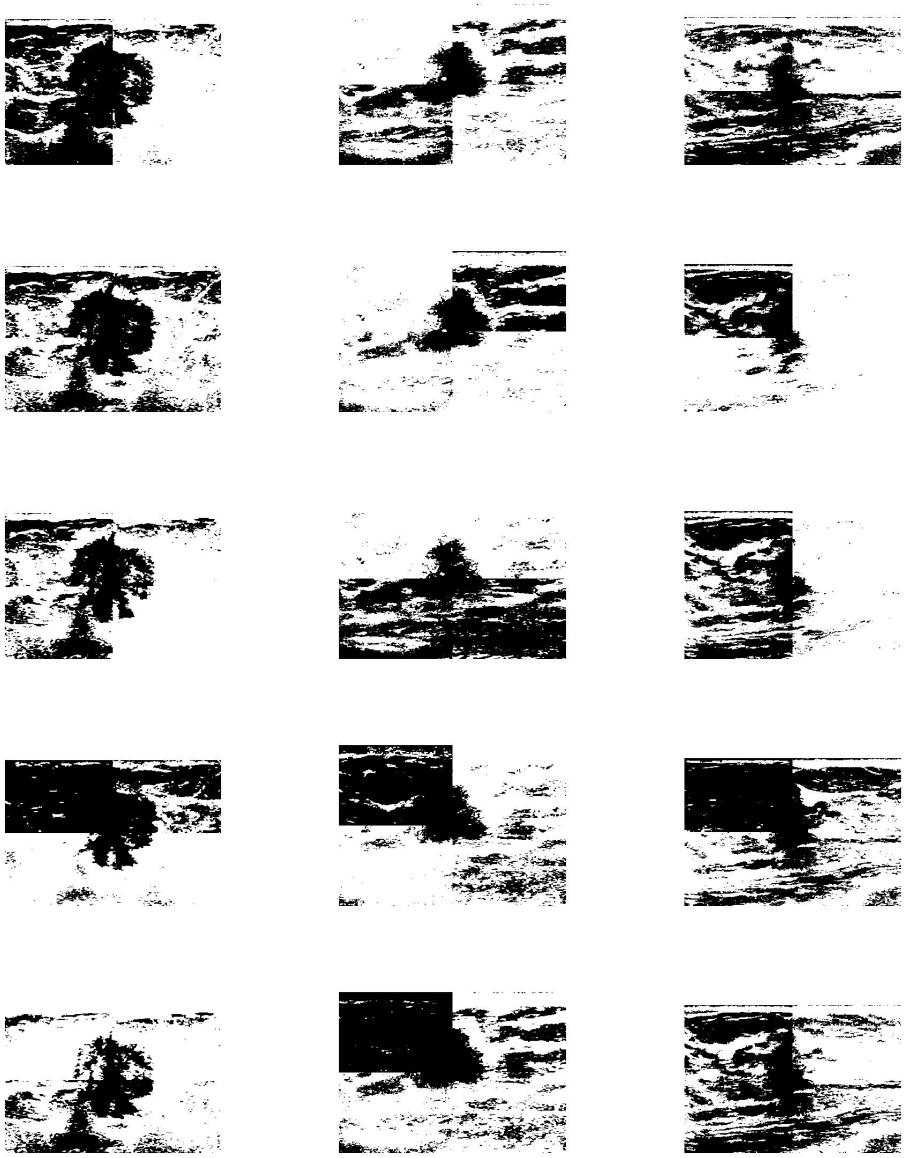

Figure 5.15: Results based on the algorithm presented in the Figure 5.9 using $NOQ(\lambda)$; From left to right (Results for image1, image2, and image3): Results (using $NOQ(\lambda)$) which were demonstrated to the user after learning (without interpolation); Each row is related to one run.

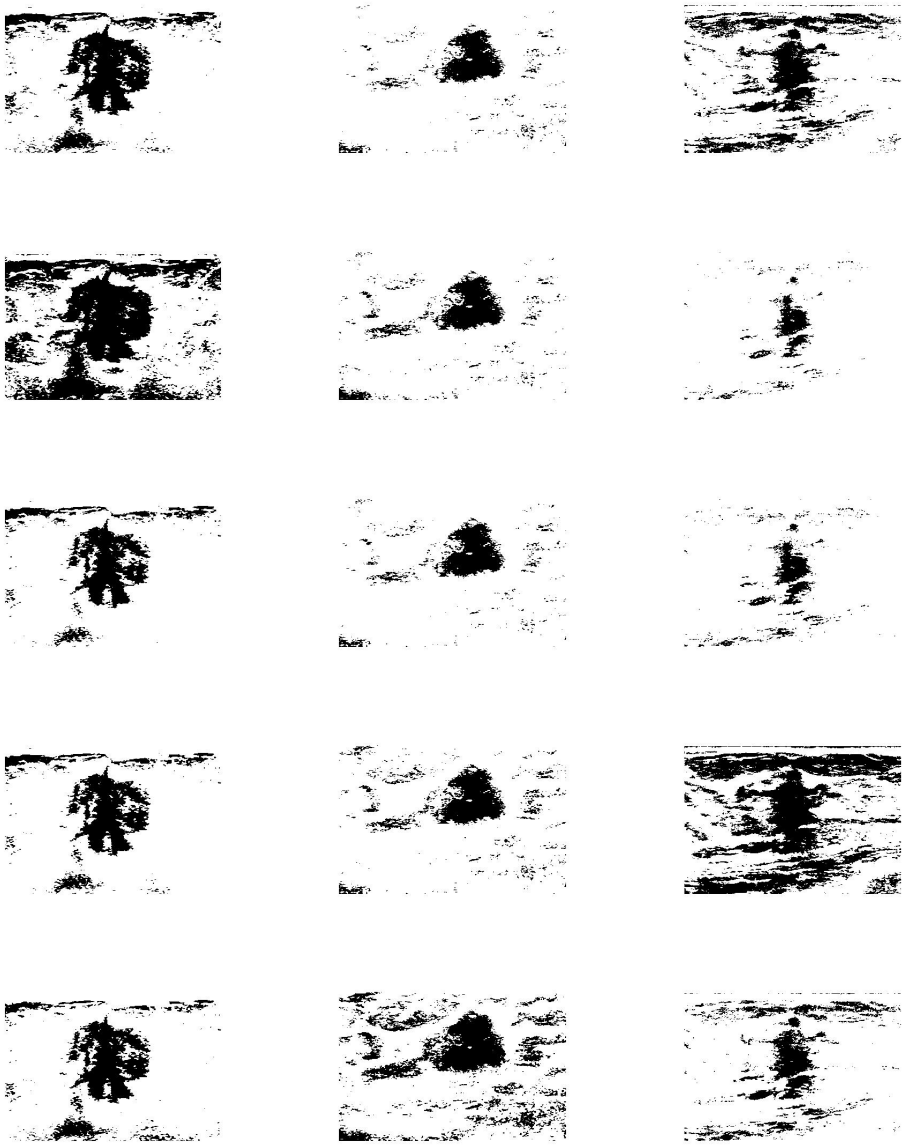

Figure 5.16: Results based on the algorithm presented in the Figure 5.9 using $NOQ(\lambda)$; From left to right (Results for image1, image2, and image3): Results (using $NOQ(\lambda)$) after the proper threshold selection by the user and applying the procedures presented in the gray boxes in the Figure 5.9; Eeach row is related to one run.

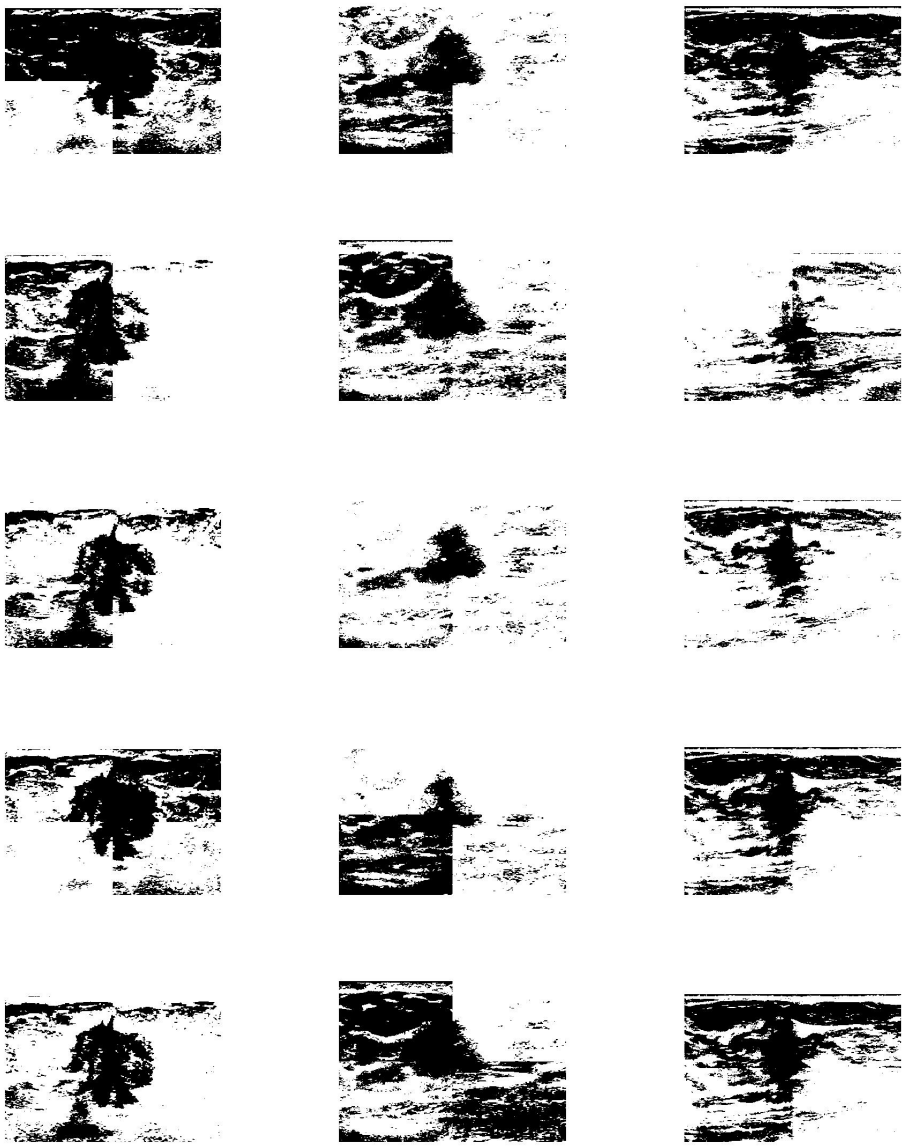

Figure 5.17: Results based on the algorithm presented in the Figure 5.9 using $NOQ(\lambda)$; From left to right (Results for image1, image2, and image3): Results (using $NOQ(\lambda)$) which were demonstrated to the user after learning (without interpolation); Each row is related to one run.

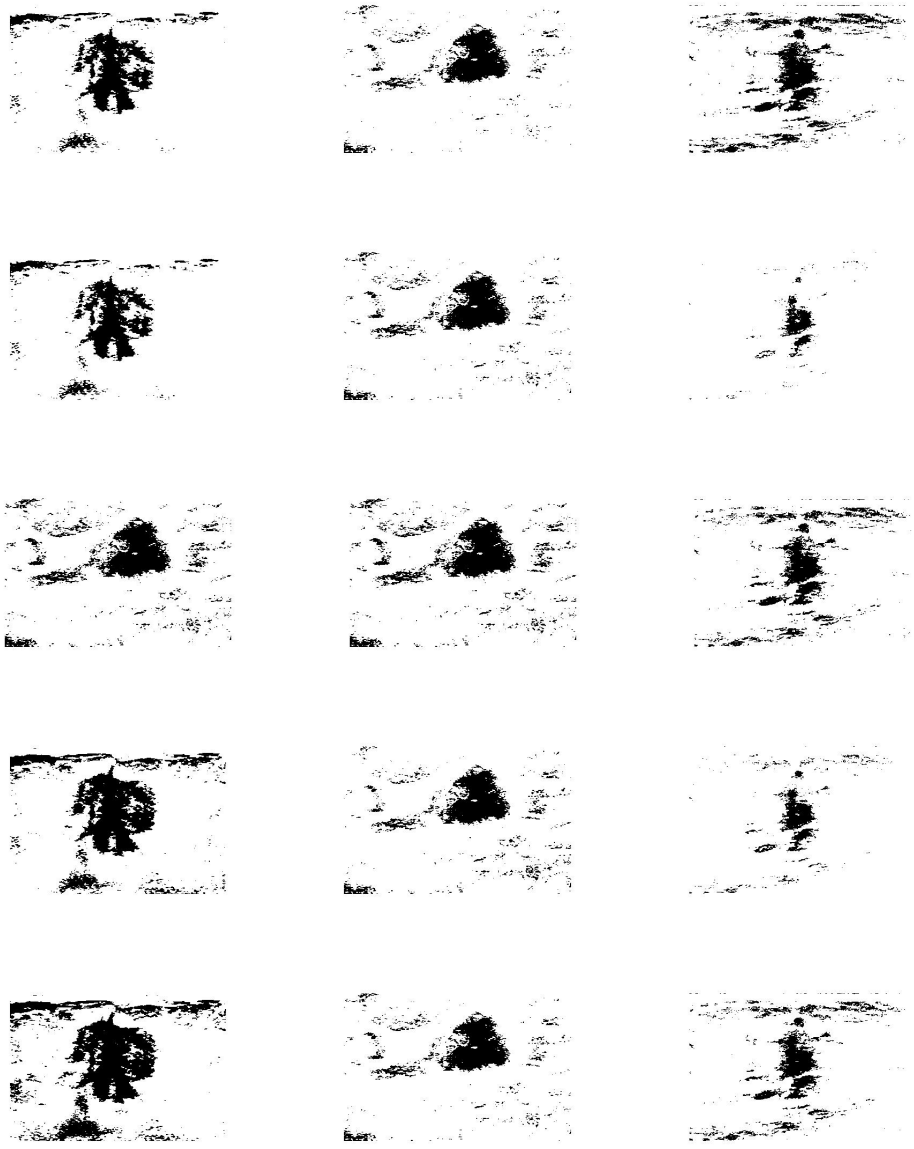

Figure 5.18: Results based on the algorithm presented in the Figure 5.9 using $NOQ(\lambda)$; From left to right (Results for image1, image2, and image3): Results (using $NOQ(\lambda)$) after the proper threshold selection by the user and applying the procedures presented in the gray boxes in the Figure 5.9; Each row is related to one run.

0.09 ± 0.03 for the second image, and from 0.31 ± 0.10 to 0.14 ± 0.09 for the third image.

It should be mentioned that the author selected the improper segments and served as a user (expert). However, for more accurate results a radiologist with expertise in this field should participate in this study which is a subject for future works. Different post-processing approach also must be employed and studied to have better and more accurate results.

5.5.5 Concluding Remarks: $NOQ(\lambda)$ for Image Thresholding

Image thresholding techniques are applied in many applications such as medical imaging to facilitate the detection and diagnostics of the object of interest by increasing the contrast of foreground and background of the images and eliminating the irrelevant details in the image. Training samples are either not available or not sufficient in many learning-based segmentation techniques. In some applications the environment is dynamic and training examples may not be effective. Reinforcement learning is reported to be used as an alternative approach because it can perform with a few training samples [61] or in some applications without any training examples [75].

The goal of this work was to introduce and examine the effects of opposition in RL-based image thresholding techniques. Opposition-based Q-learning has been applied in image thresholding in the paper presented by Sahba et al. [61]. As mentioned before, the opposition in the Q-learning may yield to reward/punishment confusion. Hence, in this work a solution for this problem is introduced which is using $NOQ(\lambda)$ technique. Earlier in this work it was mentioned that the eligibility trace and opposition trace helps to prevent reward/punishment confusion because of multiple update of Q-matrix for all states and actions as well as all states and opposite actions. In this work the $NOQ(\lambda)$ algorithm which is the modified version of opposition-based $Q(\lambda)$ algorithm was implemented and

the results are compared with the results of $Q(\lambda)$ algorithm.

The results show that $NOQ(\lambda)$ algorithm can perform faster with almost the same level of accuracy as $Q(\lambda)$ algorithm. The mixed initiative interaction was implemented to engage the user in the learning process to employ the expert's knowledge in the learning process but meanwhile prevent the user from tiredness by limited user dependency. At the beginning of the learning user provides the feedback by one click on the object of interest. At the end of the learning user can also select the segments with improper threshold to help providing more accurate results.

In this work, all the gray levels available in the image were considered in the definition of states. The dependency of reward and punishment on the image features is reduced in this work comparing to the research reported in literature to provide a more universal approach and a more general platform to compare the effects of $NOQ(\lambda)$ with $Q(\lambda)$ in the proposed image thresholding algorithm. The choice of pre-processing should be further investigated. In future works, the post-processing techniques should be employed to provide more accurate results and the expert user should participate to provide appropriate feedbacks for the agent.

Chapter 6

Summary and Conclusions

This research has explored the potentials of oppositional concepts to expedite the learning of popular reinforcement algorithms. Several types of opposition have been scrutinized in this thesis and the type-II opposition was employed in the framework of the proposed algorithms to initialize, learn/search in a shorter time. Hence, the developed extensions of RL techniques belong to the class of initializing and somatic E-OBC Algorithms.

In this thesis the initial motivation was to apply the idea of opposition to reinforcement learning focusing on its essential components. The concept of opposition was first applied to states, actions and reward/punishment which resulted in development of the OTE (oppositional target domain estimation) technique (Chapter 3). OTE is based on grid-based simulation and can successfully perform state-space reduction for search and navigation problems and achieve high reduction rates. Since the model of the environment must be available, the OTE technique is limited to model-based applications.

In many applications the model of the environment may not be existent. Hence, the challenge is how to use the concept of opposition for such applications. Since the knowledge of the states may not be available for some problems, the concept of opposition was em-

ployed to include actions. The knowledge about actions was used to integrate the opposite actions within the RL paradigm.

The problem of reward-punishment confusion was addressed by introducing the opposition traces and $OQ(\lambda)$ algorithm. Another issue elucidated in this research was the problem of the impractical Markovian update of opposite actions, which was resolved by introducing non-Markovian update for opposite actions representing a-priori knowledge integration during the learning. An opposition weight was implemented in the opposition-based update of Q values to control the trade off between exploration and exploitation of the learning process for $NOQ(\lambda)$ algorithm. Opposition is a novel approach in reinforcement learning and its implementation may be challenging in some cases. Hence, real-world case studies such as navigation, elevator control and image thresholding were selected to address the design issues in Chapter 5. The proposed technique was verified by investigation of opposition for these challenging applications of dynamic and non-deterministic nature. The results for different application areas demonstrated the advantage of using opposition in the framework of reinforcement learning techniques in terms of faster performance comparing the proposed technique with $Q(\lambda)$.

The objective of this research was to expedite some of RL-techniques using the concept of opposition by maintaining the same level of accuracy. It should be mentioned that the opposition-based RL algorithms presented in this research can be implemented whenever the opposite actions are known. In the applications such as backgammon which the opposite actions are not known the opposite actions should be extracted during the learning process. Hence, opposition mining should be further investigated.

6.1 Contributions

The contributions of this thesis can be summarized as follows:

- Opposition-based computing and its potentials for incremental learning has been investigated in details.
- The nature of opposition in reinforcement learning has been examined and the applicability of oppositeness in different components of RL paradigm has been discussed [77].
- The OTE algorithm has been introduced for model-based search and navigation [78].
- The problem of reward-punishment confusion in opposition-based Q-learning has been recognized and the opposition trace as well as opposition-based $Q(\lambda)$ algorithm have been introduced to address this issue [73].
- The opposition-based $Q(\lambda)$ has been developed for applications, for which the model of the environment is not available. Thus, the $NOQ(\lambda)$ algorithm has been introduced for model-free applications. The problem of impractical Markovian update has been solved by introducing non-Markovian update [74].
- Appropriate tradeoff between exploration and exploitation in $OQ(\lambda)$ with non-Markovian update in dynamic environments has been realized by introducing the opposition weight [76].
- The benefits of oppositional concepts in reinforcement learning has been demonstrated by detailed investigations of three major real-world examples, namely navigation, elevator control and medical image processing.

- A mixed initiative interaction [72, 88] has been applied to provide a feedback for the agent and increase the chance of learning the optimal threshold by the agent in the image thresholding application.

6.2 Future Works

The future work should focus on several remaining issues. The first emphasis should be on extending the concept of opposition to other reinforcement learning techniques. This may constitute applying the opposition to other components of reinforcement learning model such as states. Here, *opposition mining* will be a major research field, in which opposites can be extracted during learning and employed as they become available. The applicability of modeling the environment by opposite concepts should be investigated both in theoretical and experimental settings. The effects of opposition in hybrid techniques should be studied in future works as well. The choice of reinforcement signal and opposite reinforcement signal is also a crucial aspect, which should not be neglected.

Bibliography

[1] N. Abe, B. Zadrozny, J. Langford, Cross Channel Optimized Marketing by Reinforcement Learning, Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 767-772, 2004

[2] E. Alpaydin, Introduction to Machine Learning, MIT Press, 2004

[3] A. Ayesh, Emotionally Motivated Reinforcement Learning Based Controller, IEEE SMC, The Hague, The Netherlands, 2004

[4] J. Bagnell J. Schneider, Autonomous Helicopter Control using Reinforcement Learning Policy Search Methods, Proceedings of the International Conference on Robotics and Automation, 2001

[5] J. Baxter, A. Tridgell, L. Weaver, KnightCap: A Chess Program that Learns by Combining TD(λ) with Game-Tree Search, In Machine Learning Proceedings of the Fifteenth International Conference (ICML '98), 24-27 1998

[6] H.R. Berenji,Fuzzy Q-learning: a new approach for fuzzy dynamic programming, IEEE World Congress on Computational Intelligence, Fuzzy Systems, 1, 486-491, 1994

[7] H.R. Berenji, Fuzzy Q-learning for generalization of reinforcement learning, Proceedings of the Fifth IEEE International Conference on Fuzzy Systems, 3, 2208-2214, 1996

[8] A. Birk, K. Pathak, S. Schwertfeger, W. Chonnaparamutt, The IUB Rugbot: An Intelligent, Rugged mobile Robot for Search and Rescue Operations, International Workshop on Safety, Security, and Rescue Robotics (SSRR), IEEE Press, 2006

[9] C. Castillo, C. Chang, A Method to Detect Victims in Search and Rescue Operations using Template Matching, Proceedings of the 2005 IEEE International Workshop on Safety, Security and Rescue Robotics, Kobe, Japan, 2005

[10] K.R. Castleman, Digital Image Processing, Prentice Hall, Upper Saddle River, New Jersey 07458, 1998

[11] Chii-Jen Chen, Ruey-Feng Chang, Woo Kyung Moon, Dar-ren Chen, Hwa-Koon Wu, 2-D Ultrasound Strain Images For Breast Cancer Diagnosis Using Nonrigid Subregion Registration, Ultrasound in Med. & Biol., Elsevier, Vol. 32, No. 6, 837-846, 2006

[12] Collins Cobuild English Dictionary, HarperCollins Publishers, 77-85 Fulham Palace Road, London, England, 2000

[13] R.H. Crites, A.G. Barto, Improving Elevator Performance Using Reinforcement Learning, In D.S. Touretzky, M.C. Mozer, M.E. Hasselmo, (Eds.), Advances in Neural Information Processing Systems 8. MIT Press, Cambridge MA, 1996

[14] K. Driessens, S. Dzeroski, Integrating Guidance into Relational Reinforcement Learning, Machine Learning, 57, 271304, 2004

[15] P. Fidelman, P. Stone, The Chin Pinch: A Case Study in Skill Learning on a Legged Robot. In Gerhard Lakemeyer, Elizabeth Sklar, Domenico Sorenti, and Tomoichi

Takahashi, Editors, RoboCup-2006: Robot Soccer World Cup X, 5971, Springer Verlag, Berlin, 2007

[16] S. Gadanho, Reinforcement Learning in Autonomous Robots: An Empirical Investigation of the Role of Emotions, Edinburgh: PhD Thesis, University of Edinburgh, 1999

[17] M. Gemeinder, M. Gerke, GA-Based Path Planning for Mobile Robot Systems Employing an Active Search Algorithm, Applied Soft Computing, 3, 149-158, 2003

[18] S.K. Goel, Subgoal Discovery for Hierarchical Reinforcement learning Using Learned Policies, Department of Computer Science and Engineering, University of Texas at Arlington, TX, USA, Master of Science in Computer Science and Engineering, 2003

[19] R.C. Gonzalez, R. E. Woods, Digital Image Processing, Second Edition, Prentice Hall, New Jersey 07458, 2002

[20] W.D. Gregg, Analog and Digital Communication, John Wiley and Sons, Inc., USA, 1977

[21] l.O. Hall, A. Bensaid, L.P. clarke, R. Velthuizen, M.S. Silbiger, J.C. Bezdek, A Comparison of Neural Network and Fuzzy Clustering Techniques in Segmenting Magnetic resonance Images of Brain, IEEE Transactions on Neural Network, 3,672-682, 1992

[22] R.W. Hamming, Error Detecting and Error Correcting Codes, Bell System Tech. J., 29, 933-968, 1950

[23] M.A. Hearst, Trends & Controversies, Mixed-Initiative Interaction, IEEE Intelligence Systems, 1999

[24] L.K. Huang, M.J.J. Wang, Image Thresholding by Minimizing the Measures of Fuzziness, Pattern Recognition, 28, 41-51, 1995

[25] B.M. Hudock, B.E. Bishop, F.L. Crabbe, On the Development of a Novel Urban Search and Rescue Robot, in Proceedings of the Thirty-Sixth Southeastern Symposium on System Theory, 451 - 455, 2004

[26] M. Humphrys, Action Selection Methods Using Reinforcement Learning, PhD Theses, University of Cambridge, 1997

[27] C. Isbell, C. Shelton, M. Kearns, S. Singh, P. Stone, Cobot: A Social Reinforcement Learning Agent, In Proceedings of Neural Information Processing Systems 14 (NIPS), 1393-1400, 2002

[28] M.W. Kadous, R. Ka-Man Sheh, C. Sammut, CASTER: A Robot for Urban Search and Rescue, In Proceedings of the 2005 Australasian Conference on Robotics and Automation, 2005

[29] L.P. Kaelbling, Hierarchical Reinforcement Learning: Preliminary Results, in Proceedings of the Tenth International Conference on Machine Learning, 1993

[30] L.P. Kaelbling, M.L. Littman, A. R. Cassandra, Planning and Acting in Partially Observable Stochastic Domains, Artificial Intelligence, 101, 99-134, 1998

[31] L.P. Kaelbling, M. L. Littman, A. W. Moore, Reinforcement Learning: A Survey, Journal of Artificial Intelligence Research 4, 237-285, 1996

[32] G. Kantor, S. Singh R. Peterson, D. Rus A. Das, V. Kumar, G. Pereira, J. Spletzer, Distributed Search and Rescue with Robot and Sensor Teams, In Proceedings, The 4th International Conference on Field and Service Robotics, 2003

[33] J. Kittler, J. Illingworth, Minimum error thresholding, Pattern Recognition, 19, 1, 41-47, 1986

[34] N. Kohl, P. Stone, Policy Gradient Reinforcement Learning for Fast Quadrupedal Locomotion, IEEE International Conference on Robotics and Automation (ICRA 2004), 2619-2624, New Orleans, LA, May 2004

[35] C. Kwok, D. Fox, Active Sensing Using Reinforcement Learning, IROS, 2004

[36] A. Jacoff, E. Messina, B.A. Weiss, S. Tadokoro, Y. Nakagawa, Test Arenas and Performance Metrics for Urban Search and Rescue Robots, Proceedings of the 2003 IEEE/RSJ International conference on Intelligent Robots and Systems, Las Vegas, NV, 2003

[37] T. Jaakkola, S.P. Singh, M.I. Jordan, Reinforcement Learning Algorithm for Partially Observable Markov Decision Problems, In Advances in Neural Information Processing Systems 7, 1994

[38] S.U. Lee, S.Y. Chung, R.H. Park, A Comparative Performance Study of Several Global Thresholding Techniques for Segmentation, Computer Vision, Graphics, and Image Processing 52, 171-190, 1990

[39] M. Lewis, K. Sycara, I. Nourbakhsh, Developing a Testbed for Studying Human-robot Interaction in Urban Search and Rescue, Proceedings of the 10th International Conference on Human Computer Interaction (HCII'03), 270-274, 2003

[40] R. Maclin, J. Shavlik, L. Torrey, T. Walker, E. Wild, Giving Advice about Preferred Actions to Reinforcement Learners Via Knowledge-Based Kernel Regression, American Association for Artificial Intelligence, 2005

[41] S. Mahadevan, L. P. Kaelbling, The NSF Workshop on Reinforcement Learning: Summary and Observations, Appeared in AI Magazine, 1996

[42] A. Mc Govern, R. S. Sutton, Macro-Actions in Reinforcement Learning: An Empirical Analysis, University of Massachusetts, Amherst, Technical Report Number 98-70, 1998

[43] A.R. Malisia, H.R. Tizhoosh, Applying Opposition-Based Ideas to the Ant Colony System, IEEE Swarm Intelligence Symposium (SIS2007), Hawaii, 182 - 189, 2007

[44] Merriam-Webster Online English Dictioinary, www.m-w.com

[45] J. Moody, M. Saffell, Learning to Trade via Direct Reinforcement, IEEE Transactions on Neural Networks, 12, 4, 2001

[46] E.F. Morales, Relational State Abstractions for Reinforcement Learning, Appearing in Proceedings of the ICML'04 workshop on Relational Reinforcement Learning, Banff, Canada, 2004

[47] A.Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, E. Liang. Inverted Autonomous Helicopter Flight via Reinforcement Learning, In International Symposium on Experimental Robotics, 2004

[48] N. Otsu, A Threshold Selection Method From Gray Level Histogram, IEEE Transactions on Systems, Man, and Cybernetics, SMC-9, 62-66, 1979

[49] J. Peng, R. J. Williams, Incremental Multi-Step Q-Learning, Machine Learning, 22, 1996

[50] T.D. Pham, Perception-Based Hidden Markov Models: A Theoretical Framework for Data Mining and Knowledge Discovery, Soft Computing 6, 400-405, Springer-Verlag, 2002

[51] A. Potapov, M.K. Ali, Convergence of Reinforcement Learning Algorithms and Acceleration of Learning, Physical Review, E 67, 026706, 2003

[52] S. Proper, P. Tadepalli, Scaling Average-Reward Reinforcement Learning for Product Delivery, American Association for Artificial Intelligence, 2004

[53] S. Rahnamayan, H.R. Tizhoosh, M.M.A. Salama, Opposition-Based Differential Evolution Algorithms, IEEE Congress on Evolutionary Computation (CEC-2006), Vancouver, 2010-2017, 2006

[54] S. Rahnamayan, H.R. Tizhoosh,M. Salama, Opposition-Based Differential Evolution for Optimization of Noisy Problems, IEEE Congress on Evolutionary Computation (CEC-2006), Vancouver, 1865-1872, 2006

[55] S. Rahnamayan, H.R. Tizhoosh, M.M.A Salama, A Novel Population Initialization Method for Accelerating Evolutionary Algorithms, Elsevier Journal on Computers and Mathematics with Applications,53, 10, 1605-1614, 2007

[56] S. Rahnamayan, H.R. Tizhoosh, M. Salama, Opposition Versus Randomness in Soft Computing Techniques, Applied Soft Computing, 8, 2, 906-918, 2008

[57] C. H. Ribeiro, Embedding a Priori Knowledge in Reinforcement Learning, Journal of Intelligent and Robotic Systems 21, 51-71, 1998

[58] C. Ribeiro, Reinforcement Learning Agent, Artificial Intelligence Review, 17, 223-250, 2002

[59] S.J. Russell, P. Norvig, Artificial intelligence: a modern approach, Pearson Education Inc., New Jersey, 2003

[60] F. Sahba, Reinforced Segmentation of Images Containing One Object of Interest, PhD Thesis, Department of Systems Design Engineering, University of Waterloo, 2007

[61] F. Sahba, H.R. Tizhoosh1, M.M.M.A. Salama, Application of Opposition-Based Reinforcement Learning in Image Segmentation, Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Image and Signal Processing, CIISP 2007

[62] P.K. Sahoo, S. Soltani, A. K. C. Wong, Y. C. Chen, A Survey of Thresholding Technique, Computer Vision, Graphics, and Image Processing 41, 233-260, 1988

[63] P. Sahoo, C. Wilkins, J. Yeager, Threshold Selection Using Renyi's Entropy, Pattern Recognition, 30, 1, 71-84, 1997

[64] B. Sankur, M. Sezgin, Survey over image thresholding techniques and quantitative performance evaluation, Journal of Electronic Imaging, 13, 1, 146 - 165, 2004

[65] J. Scholtz, J. Young, J.L. Drury, H.A. Yanco, Evaluation of Human-Robot Interaction Awareness in Search and Rescue, Proceedings of he 2004 IEEE International Conference on Robotics and Automation, New Orleans, 2004

[66] S.J. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Pearson Education Inc., New Jersey, 2003

[67] J. Schaeffer, M. Hlynka, V. Jussila, Temporal Difference Learning Applied to a High-Performance Game-Playing Program, International Joint Conference on Artificial Intelligence (IJCAI), 529-534, 2001

[68] D. Shapiro, P. Langley, R. Shachter, Using Background Knowledge to Speed Reinforcement Learning in Physical Agents, AGENTS'01, Montral, Quebec, Canada, 2001

[69] M. Shokri, H.R. Tizhoosh, Using Reinforcement Learning for Image Thresholding, Canadian Conference on Electrical and Computer Engineering, 1, 1231-1234, 2003

[70] M. Shokri, H.R. Tizhoosh, $Q(\lambda)$-Based Image Thresholding, Canadian Conference on Computer and Robot Vision, 504-508, 2004

[71] M. Shokri, Reinforced Adaptive Image Thresholding, Thesis, Master of Applied Science in Systems Design Engineering, University of Waterloo, 2004

[72] M. Shokri, H.R. Tizhoosh, M. Kamel, Reinforcement Learning for Personalizing Image Search, In Proceedings of the 3rd Annual Scientific Conference of the LORNET Research Network (I2LOR-06), 7, 2006

[73] M. Shokri, H. R. Tizhoosh, M. Kamel, Opposition-Based $Q(\lambda)$ Algorithm, International Joint Conference on Neural Networks, IJCNN, 646-653, 2006

[74] M. Shokri, H.R. Tizhoosh, M.S. Kamel, Opposition-Based $Q(\lambda)$ with Non-Markovian Update, IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning, Hawaii, USA, 288-295, 2007

[75] M. Shokri, H.R. Tizhoosh, A Reinforcement Agent for Threshold Fusion, Journal of Applied Soft Computing, ASOC-309, Elsevier (Accepted 2006), 8, 1, 174-181, January 2008

[76] M. Shokri, H.R. Tizhoosh, M.S. Kamel, Tradeoff between Exploration and Exploitation of $OQ(lambda)$ with Non-Markovian Update in Dynamic Environments, IEEE International Joint Conference on Neural Networks, 2916-2922, 2008

[77] M. Shokri, H.R. Tizhoosh, M.S. Kamel, The Concept of Opposition and its Use in Q-learning and $Q(\lambda)$ Techniques, H.R. Tizhoosh, M. Ventresca, (Editors), Oppositional

Concepts in Computational Intelligence, To be Published by Springer Physika-Verlag, 2008

[78] M. Shokri, H.R. Tizhoosh, M.S. Kamel, Oppositional Target Domain Estimation Using Grid-Based Simulation, Journal of Applied Soft Computing, Elsevier, In Press, 2008

[79] O. Simsek, A.G. Barto, Using Relative Novelty to Identify Useful Temporal Abstractions in Reinforcement Learning, Proceedings of theTwenty-First International Conference on Machine Learning (ICML), 2004

[80] S. Singh, D. Litman, M. Kearns, M. Walker, Optimizing Dialogue Management with Reinforcement Learning: Experiments with the NJFun System, In Journal of Artificial Intelligence Research (JAIR), 16, 105-133, 2002

[81] P. Stone, R.S. Sutton, Scaling Reinforcement Learning toward RoboCup Soccer, In The Eighteenth International Conference on Machine Learning (ICML 2001), 537-544, Williamstown, MA, USA, June 2001

[82] R.S. Sutton, Learning to Predict by the Methods of Temporal Differences, Machine Learning, 3, 9-44, 1988

[83] R.S. Sutton, A.G. Barto, Reinforcement learning: An Introduction, Cambridge, Mass., MIT Press, 1998

[84] R.S. Sutton, Open theoretical questions in reinforcement learning. In Proceedings of the Fourth European Conference on Computational Learning Theory (Proceedings EuroCOLT'99), pp. 11-17, Fischer, P., Simon, H.U., Eds. Springer-Verlag, 1999

[85] T. Taxt, A. Lundervold , J. Strand, S. Holm, Advances in Medical Imaging, Invited plenary, 14th Int'l Conference on Pattern Recognition (ICPR98), Brisbane, Australia, 1998

[86] S. Thrun, D. Fox, W. Burgard, F. Dellaert, Robust Monte Carlo Localization for Mobile Robots, Artificial Intelligence, 128, 99-141, 2001

[87] H.R. Tizhoosh, G. Krell, B. Michaelis, λ-Enhancement: Contrast Adaptation Based on Optimization of Image Fuzziness, Proceeding of FUZZ-IEEE'98, 1548-1553, Alaska, USA, 1998

[88] H.R. Tizhoosh, M. Shokri, M. Kamel, The Outline of a Reinforcement-Learning Agents for E-Learning Applications, Accepted for Samuel Pierre (ed.), E-Learning Networked Environments and Architectures: A Knowledge Processing Perspective, Springer Book Series, 2005

[89] H.R. Tizhoosh, Image thresholding using type II fuzzy sets, Pattern Recognition, 38, 12, 2363-2372, 2005

[90] H.R. Tizhoosh, Reinforcement Learning Based on Actions and Opposite Actions, ICGST International Conference on Artificial Intelligence and Machine Learning (AIML-05), Cairo, Egypt, 2005

[91] H.R. Tizhoosh, Opposition-Based Learning: A New Scheme for Machine Intelligence, International Conference on Computational Intelligence for Modeling Control and Automation - CIMCA'2005, Vienna, Austria, vol. I, 695-701, 2005

[92] H.R. Tizhoosh, Opposition-Based Reinforcement learning, Journal of Advanced Computational Intelligence and Intelligent Informatics, Vol. 10, No. 4, pp. 578-585, 2006

[93] H.R. Tizhoosh, Mario Ventresca, Shahryar Rahnamayan, Opposition-Based Computing, H.R. Tizhoosh, M. Ventresca, (Editors), Oppositional Concepts in Computational Intelligence, To be Published by Springer Physika-Verlag, 2008

[94] W. Tsai, Moment-preserving thresholding: A new approach, Computer Vision Graphics, and Image Processing 29, 377-393, 1985

[95] M. Ventresca, H.R. Tizhoosh, Improving the Convergence of Backpropagation by Opposite Transfer Functions, International Joint Conference on Neural Networks (IJCNN), Vancouver, 9527-9534, 2006

[96] C.J.C.H. Watkins, Learning from Delayed Rewards, Cambridge, Cambridge University, 1989

[97] C.J.H. Watkins, P. Dayan, Technical Note, Q-Learning, Machine Learning, 8, 279-292, 1992

[98] M. Wiering, $QV(\lambda)$-learning: A New On-policy Reinforcement Learning Algorithm, Proceedings of the 7th European Workshop on Reinforcement Learning, D. Leone (editor), 17-18, 2005

[99] S. Yamada, Recognizing Environments from Action Sequences Using Self-Organizing Maps, Applied Soft Computing, 4, 35-47, 2004

[100] S. Yamada, Evolutionary Behavior Learning for Action-Based Environment Modeling by a Mobile Robot, Applied Soft Computing, 5, 245-257, 2005

[101] H. Yan, Unified Formulation of a Class of Image Thresholding Techniques, Pattern Recognition, 29, 12, 2025-2032, 1996

[102] X. Yan, P. Diaconis, P. Rusmevichientong, B. Van Roy, Solitaire: Man Versus Machine, Advances in Neural Information Processing Systems 17, MIT Press, 2005

[103] P.G. Zimbardo, R. L. Johnson, A. L. Weber, Psychology, Core Concepts, 5/e, Sample Chapter 6, Allyn & Bacon, Boston, MA 02116, www.ablongman.com, 2005

Glossary of Terms

RL Reinforcement learning

TD Temporal difference

Q Q-learning technique

DP Dynamic programming

MC Monte Carlo

MDP Markov decision process

Sarsa State action reward state action

OBL Opposition-based learning

OBC Opposition-based computing

I-OBC Implicit OBC

E-OBC Explicit OBC

OTE Oppositional target domain estimation

RTE Randomized target domain estimation

OBL Opposition-based learning

OQ(λ) Opposition-based $Q(\lambda)$

NOQ(λ) Non-Markovian opposition-based $Q(\lambda)$