

**Technical University of Košice**  
**Faculty of Electrical Engineering and Informatics**

**Learning Using Teleoperation on  
Humanoid Robotic Systems**

**Bachelor's Thesis**

**2015**

**Ján Gamec**

**Technical University of Košice**  
**Faculty of Electrical Engineering and Informatics**

# **Learning Using Teleoperation on Humanoid Robotic Systems**

**Bachelor's Thesis**

Study Programme: Cybernetics

Field of study: 9.2.7 Cybernetics

Department: Department of Cybernetics and Artificial Intelligence (KKUI)

Supervisor: prof. Ing. Peter Sinčák, CSc.

Consultant(s):

**Košice 2015**

**Ján Gamec**

# Errata

Learning Using Teleoperation on Humanoid Robotic Systems

Ján Gamec

Košice 2015

Page	Line	Wrong	Correct

## **Abstract**

This bachelor thesis is focused on problems regarding teleoperation of humanoid robots and knowledge acquired during it. It deals with a utilization of this knowledge studying specific learning techniques. The main focus is given to reinforcement learning and also neural networks. This thesis overviews an existing methods utilized in real-world applications, and also proposes some improvements for one of these methods. In addition, the method is tested on a task with a humanoid robot.

## **Keywords**

Teleoperation, Learning, Humanoid Robotic systems

## **Abstrakt**

Táto bakalárska práca je orientovaná na problematiku týkajúcu sa teleoperácie na humanoidných robotoch a taktiež znalostí nadobudnutých počas nej. Predmetom je aj štúdium metód, ktoré pomáhajú efektívne využiť tieto znalosti. Veľká pozornosť je venovaná tzv. učeniu odmenou a trestom a problematike neurónových sietí. Cieľom práce je prehľad existujúcich metód využívajúcich sa v reálnych aplikáciách ako aj návrh vylepšenia jednej z nich. Táto metóda je následne podrobená testu na úlohe s humanoidným robotom.

## **Klíčové slová**

Teleoperácia, Učenie, Humanoidný robotický systém

# **ZADANIE BAKALÁRSKEJ PRÁCE**

Študijný odbor: **9.2.7 Kybernetika**  
**5.2.14 Automatizácia**  
Študijný program: **Kybernetika**

Názov práce:

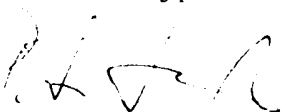
**Učenie stroja teleoperáciou na humanoidných robotických systémoch**  
Learning Using Teleoperation on Humanoid Robotic Systems

Študent: **Ján Gamec**  
Školiteľ: **prof. Ing. Peter Sinčák, CSc.**  
Školiace pracovisko: **Katedra kybernetiky a umelej inteligencie**  
Konzultant práce:  
Pracovisko konzultanta:

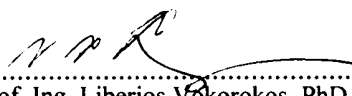
Pokyny na vypracovanie bakalárskej práce:

1. Realizujte prehľad metód pre teleoperáciu robotických systémov v oblasti humanoidnej robotiky
2. Realizujte prehľad metód typu „reinforcement learning“ pre problémy teleoperácie s učením
3. Navrhnete a vyskúšajte vybranú metódu vo vybranom simulátore robotických systémov na testovacej úlohe pre robotickú platformu NAO
4. Preskúmajte implementovanú metódu z pohľadu inkrementálneho učenia a postupného znižovania ingerencie operátora do teleoperácie na vybranom experiment pre platformu NAO
6. Vypracujte užívateľskú a systémovú príručku a dokumentáciu podľa pokynov vedúceho práce

Jazyk, v ktorom sa práca vypracuje: slovenský  
Termín pre odovzdanie práce: 29.05.2015  
Dátum zadania bakalárskej práce: 31.10.2014

  
-----  
prof. Ing. Peter Sinčák, CSc.  
vedúci garantujúceho pracoviska



  
-----  
prof. Ing. Liberios Vokorokos, PhD.  
dekan fakulty

## **Declaration**

I hereby declare that this thesis is my own work and effort. Where other sources of information have been used, they have been acknowledged.

Košice, May 29, 2015

.....

*Signature*

## **Acknowledgement**

I would like to thank sincerely my supervisor, prof. Peter Sinčák for his time and a guidance throughout the entire process of development and research. I would also like to express a special thanks to all the current, and also previous members of the Center for Intelligent Technologies at TU Košice, for their advice and also pleasant working environment. Last but not least, I would like to thank all, who helped me with writing and completing this thesis.

# Contents

<b>1</b>	<b>The problem definition</b>	<b>3</b>
<b>2</b>	<b>Teleoperation Systems</b>	<b>5</b>
2.1	Overview . . . . .	5
2.2	Important notions . . . . .	5
2.3	Teleoperation methods and applications . . . . .	7
2.3.1	Methods . . . . .	7
2.3.2	Applications . . . . .	8
2.4	From teleoperation to autonomy . . . . .	9
2.5	System autonomy . . . . .	10
<b>3</b>	<b>Learning from Teleoperation</b>	<b>12</b>
3.1	Learning from teleoperation using Fuzzy systems . . . . .	13
3.1.1	Expanding hyperboxes in the state space . . . . .	13
3.2	Learning from Teleoperation using Neural Networks . . . . .	15
<b>4</b>	<b>Reinforcement Learning</b>	<b>17</b>
4.1	Introduction . . . . .	17
4.2	Basic terminology in Reinforcement Learning . . . . .	17
4.3	Markov Decision Process . . . . .	19
4.4	General Reinforcement Learning Approaches . . . . .	20
4.4.1	Dynamic Programming . . . . .	20
4.4.2	Monte Carlo Methods . . . . .	23
4.4.3	Temporal Difference methods . . . . .	24
4.4.4	Q-Learning . . . . .	24
<b>5</b>	<b>Reinforcement learning in Teleoperation</b>	<b>27</b>
5.1	General scheme of learning . . . . .	27
5.2	GARIC . . . . .	29



---

5.2.1	Overview . . . . .	29
5.2.2	Action Evaluation Network . . . . .	29
5.2.3	Action Selection Network . . . . .	31
5.2.4	Stochastic Action Modifier . . . . .	32
5.3	Neural Fitted Q Iteration . . . . .	32
5.3.1	Main concept . . . . .	33
5.3.2	Basic NFQ Algorithm . . . . .	34
5.3.3	Utilization of NFQ in Teleoperation . . . . .	35
<b>6</b>	<b>Experiment</b>	<b>36</b>
6.1	Task description . . . . .	36
6.2	Technology used in experiment . . . . .	37
6.2.1	Webots . . . . .	37
6.2.2	Redis . . . . .	38
6.2.3	NAO . . . . .	38
6.3	Reinforcement learning from teleoperation . . . . .	39
6.3.1	Modified NFQ Algorithm . . . . .	39
6.3.2	RPROP Training . . . . .	41
6.4	Experiment setup . . . . .	43
6.5	Experiment evaluation . . . . .	44
<b>7</b>	<b>Conclusion</b>	<b>50</b>
	<b>Bibliography</b>	<b>52</b>
	<b>Appendices</b>	<b>54</b>

## List of Figures

2-1	Visual representation of teleoperation process and terms. . . . .	7
2-2	iRobot's view of autonomy levels. . . . .	11
3-1	Conversion of hyperboxes into corresponding membership functions. The $x_i$ in the figure illustrates a state variables, and $A_{1,2}$ are particular classes(actions). The $\mu_{A_i}(x_i)$ represents a degree of membership of state variable $x_i$ to the corresponding class. . . . .	15
4-1	Reinforcement learning process. . . . .	19
5-1	Utilizing Teleoperation in process of Reinforcement Learning to learn decision strategy according to human. . . . .	28
5-2	Operator in a position of a supervisor, after system has learned the decision model. . . . .	29
5-3	General architecture of GARIC. . . . .	30
5-4	Structure of AEN module in GARIC. . . . .	31
5-5	Internal structure of ASN module of GARIC architecture. . . . .	32
6-1	State information in the experiment . . . . .	45
6-2	Task learning process performance overview. . . . .	46
6-3	MTIA level of an agent during learning process. . . . .	47
6-4	State variables before learning process. . . . .	48
6-5	State variables after learning process. . . . .	49

## List of Tables

6-1 Comparison of RPROP and Backprop learning algorithms. . . . .	43
---	----

## List of Symbols and Abbreviations

$E_\pi$  Expected value returned from cumulative reward

$Q^\pi(s, a)$  State-action value function

$R$  Reward function

$T$  Transition function

$V^\pi$  Value function

$\mu_{A_i}$  Fuzzy membership functions

$\pi$  Decision strategy, probabilistic function

$r_t$  Reward/reinforcement acquired in time t

$s_t$  State in time t

AEN Action Evaluation Network

ASN ACTION Selection Network

DP Dynamic programming

GARIC Generalized Approximate Reasoning based Intelligent Control

GTI Global Task Intelligence

HTI Human Task Intelligence

LfD Learning from Demonstration

MC Monte Carlo methods

MDP Markov Decision Process

MTI Machine Task intelligence

MTIA Machine Task Intelligence Autonomy

NFQ Neural Fitted Q Iteration

RPROP Resilient Propagation

SAM Stochastic Action Modifier

TD Temporal Difference methods

## Introduction

In the past two decades, the world has encountered a rapid increment of occurrences of robotic platforms in various sectors. Robots are used during surgeries or on a production line in a factory. Applications areas have huge differences but still a few thing in common. Since the number of robots is increasing, demand for specialists, who can control such machines is also growing. We have two options, either teach and train more specialized people or, on the other hand, start to develop machines that are more responsive to the human. As far as I am not a qualified teacher and don't know about teaching, This work will be focused on the problems regarding the second case.

Teleoperation has been developing for long decades, since the age of Nicola Tesla. Since then, people studied how to control machines without using physical force. However, this field was available only to the few people dedicating to research. In the middle of the last century, when the field of Cybernetics started to expand, scientists also began to think of a way that could bring machines closer to common people. They studied interactions between humans and machines. Nowadays, we can say that Teleoperation is closely bound to the area called Human-Robot Interaction.

The fundamental goal of this work is to study methods, which can help interaction with machines to common people in a user-friendly way. This means that a human controlling some robot does not have to care about the physical and mathematical background of the processes running in the background. To go even further, we want to study approaches, which can unburden human from infinitely often repeating same tasks. This means that we study system that are capable of learning from experience and apply this knowledge to specific tasks after then.

To make these applications available to ordinary, we need to bring the maximal possible level of abstraction into the learning process and also to the user interface.

For this purposes we've chosen reinforcement learning methods, that does not require a detailed mathematical description of the environment and whole system. Even though it is working in the laboratory conditions, real-world applications require more sophisticated methods. Due to this we have chosen a method called the Neural Fitted Q Iteration (NFQ), and we are proposing its modification so it can be applied to learning directly from even layman's knowledge.

To test this method's performance, we are proposing an experiment, in which humanoid robot is facing the trivial task. The goal is to learn the task as fast as possible and make this robot fully autonomous in performing this task. Controlling of this robot is designed and adjusted, so that anybody with a computer or cell phone can perform this learning process.

# 1 The problem definition

This bachelor thesis is focused on machine learning, especially areas called the reinforcement learning and learning from teleoperation.

The goal of this work is to fulfill following tasks:

1. Review of methods, which are applicable to the teleoperation of a robotic system. The main focus is given to the humanoid robotics area. The work will also explain the basic terminology used in teleoperation, due to its necessity for describing different teleoperation approaches. It also gives a focus to learning from the teleoperation utilizing various artificial intelligence approaches.
2. Review of reinforcement learning methods applicable to the problems regarding learning from teleoperation. The work also reviews and describes general methods and terminology in reinforcement learning. We consider it to be a necessary background for advanced reinforcement learning methods applied in the learning from teleoperation.
3. Proposal, implementation and also a testing of such method. The work proposes a modification for advanced reinforcement learning method that can be utilized in the learning from teleoperation of a humanoid robot. To verify the method, this work also describes an experiment done in a simulated environment utilizing a NAO humanoid robot.
4. Analysis of implemented method from the aspect of incremental learning. The experiment also evaluates how successful is a robot before and after the learning procedure. It also analyzes the progressive decrease of operator's involvement in the task. The goal of the experiment is to teach a robot to perform a task so that a human operator does not have to make any effort to finish it.
5. Generate a user and also a reference guide documenting the created applica-



tion. The file and the code structure might be complex, this work also includes a complete guide and explanation of all parts of the code.

## 2 Teleoperation Systems

### 2.1 Overview

Teleoperation is a scientific field studying approaches to operating objects remotely. It has been becoming more and more popular over last decades and been chosen as the topic for several academic works. In the pursuit of the origin of this word, one has to understand the meaning. The word *teleoperation* consists of two parts: *tele* - that means at distance and *operation* - which can be understood as manipulation or control. Joining this two words together, one gets a powerful method to manipulate and change the environment at a distance, almost without any physical activity or force. The idea was not invented nowadays, but it is dating back to late 1800s when Nikola Tesla as the first man, patented fundamental principles about manipulating objects at a distance using radio communication. However, the idea has grown up more than one century later at world war. The Russian army developed a teletank that could be operated at a distance up to 1500 meters using radio communication. Despite the military applications, that spread widely during the war, first famous non-military application was developed by R.Goertz for the very first nuclear reactor. The mechanical manipulator was able to move radioactive material protecting human from contact with dangerous material. Over the years until now, the field of teleoperation became an important area studied by cybernetics and artificial intelligence.

### 2.2 Important notions

This section overviews and explains most important terms used in teleoperation and especially in telerobotics. These terms have been explained in detail in [1, 2, 3].

**Robot** is a word firstly mentioned in a book of Karel Čapek called R.U.R. History

and the meaning of this word has been subject of many discussions and articles, but the one I like the most is the one from [4]: "The robot is any automatically operated device that replaces human effort." This means that the robotic systems should be able to act physically or mentally as a human being. Thus, robotics is the field studying construction, engineering and programming of such systems.

**Humanoid robot** is a type of robot whose body shape (or part of it) ought to resemble the shape of human body.

**Operator** is a person, who takes responsibility for actions executed by an agent, or in our case, the robot. Generally speaking, the operator is the person who holds the joystick and moves the robot at a distance.

**Telemanipulator** or remote manipulator is a mechanical device providing operator ability to control a remote agent. This term is usually bound to the area of microsurgeries when the operator controls mechanical hands to perform a surgery. In a less strict definition, it can be any hardware (or even software) that the operator controls to move and manipulate the remote agent in an environment.

**Teleoperator** is the mentioned agent or the robot controlled at a distance. Generally speaking, teleoperator is any device manipulated by an operator and providing feedback about own presence and state.

**Teleoperation** is then a process, when the operator controls a telemanipulator to move the real teleoperator in a distant environment, where distance can vary from millimeters to thousands of kilometers. Supervision of state and progress of controlled system is achieved using sensory information provided by teleoperator system.

These terms are visually explained in the figure 2–1 according to the [5].

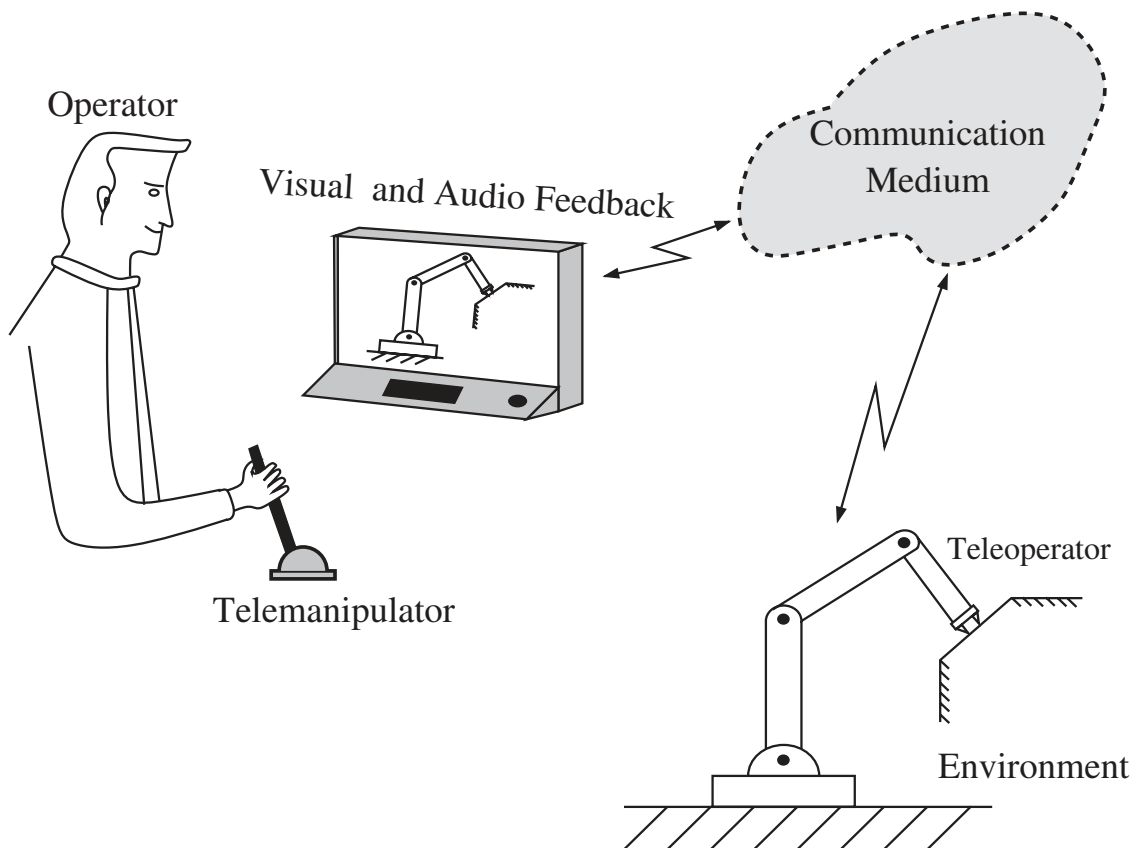


Figure 2–1 Visual representation of teleoperation process and terms.

## 2.3 Teleoperation methods and applications

### 2.3.1 Methods

Most teleoperation methods used in a field of robotics can be characterized by a master-slave relation. This means that the telerobot (slave) fulfills commands given by an operator (master). Such methods can be distinguished by a form of teleoperator used for giving orders to a slave.

- **Simple, Joystick** - approach utilizing a gamepad or joystick is one of the easiest and most used methods for teleoperation nowadays. Such hardware is easily accessible and doesn't require complex programming implementation. Almost every higher programming language has support for such devices. This

approach covers a wide range of suitable application, and, in addition, such mechanism can often control much more complex systems. An example of an appropriate task might be a robot navigation.

- **Visual-based** - methods are using sensors such a Microsoft Kinect, to track a motion of the operator. Opposing to the previous type, where the operator had to do unnatural hand motion for teleoperation task, this approach is using more natural motions or gestures. A good example might be a grabbing an object task where teleoperation using joystick might be very complex task. On the other hand, utilizing sensor system like Kinect or Leap Motion would simplify a task a lot, from the operator's point of view. The price for such comfort is much complex implementation.
- **Isomorphic** - approach is utilizing a piece of hardware crafted uniquely for a certain type of robot and task. As mentioned before, visual-based methods were much more comfortable, but still were lacking any feedback, which would provide, let's say, realistic feeling about the environment and robot status. For such reasons, there were developed systems such a KIST [6]. The KIST system is tracking a human hand and transferring its motion to the robot. The robot then sends back information from his tactile sensors and the KIST hand extension, equipped with tactile feedback devices, provides operator realistic feeling about the force used for e.g. holding an object.

### 2.3.2 Applications

Teleoperation is a widely applicable technique utilized in various types of tasks. As mentioned before, one of the first areas, where teleoperation took the important role was military area. The main idea of utilizing a teleoperation in this area was to protect a human from finding himself in dangerous situations. Due to this fact, Army started developing vehicles, that don't require a direct presence of a man on

board to fulfill tasks. One of the most famous examples are military drones X-47B that can even refuel themselves without the onboard presence of a pilot.

Another lifesaving example of teleoperation application are surgery robots allowing telesurgeries, which are surgeries where specialists do not need to be present at a place. Surgery robots are far more precise than a human hand, which brings medicine a step forward in neuro surgeries. The best example of such a system is well-known da Vinci system.

Nowadays, teleoperation is present almost everywhere the presence of human may be dangerous. We could find many examples where robotic systems replace a human in dangerous environments like mines, the radioactive environment of highly explosive areas.

## **2.4 From teleoperation to autonomy**

Even though the area of teleoperation is broad enough to cover more than one bachelor work, the topic of this thesis is not focused only on teleoperation. This work consider teleoperation to be a starting point for reaching a certain degree of autonomy of a teleoperator.

Word *autonomy* originating from ancient Greek is a capacity of making rational, not coerced decision. In a field of robotics, this means, that a robot, occurring in some environment, should be able to make a rational (to human supervisor) decision, based on his own knowledge or experience. Thus, this strongly human-like behavior requires much intelligence and a process of acquiring such intelligence or knowledge is called learning. Generally speaking about learning from teleoperation means, that the teleoperator is creating an own decision model based on experience observed from decisions of the operator.

When speaking about decision model, I mean some kind of policy [7] or strategy which in every state (situation) decide, what action is the most appropriate to move closer to the goal of a task. This means approximating such a probabilistic function  $\pi$  (2.1), that in each state maps highest probability to the best action according to this rule.

$$\pi = S \times A \rightarrow \langle 0, 1 \rangle \quad (2.1)$$

Learning in this manner means finding and approximating such a function, utilizing some method of artificial intelligence (Artificial Neural Networks, Fuzzy Systems and other)

## 2.5 System autonomy

However there does not exist a universal solution to achieve complete autonomy in a robotic system yet, term autonomy is closely bound to the type of task performed. Formally speaking, measuring autonomy is a task-oriented approach expressed by the equation (2.2)[8].

$$GTI = HTI + MTI \quad (2.2)$$

GTI (Global Task Intelligence) always has value of 1, and it is a sum of HTI (Human Task Intelligence) and MTI (Machine Task intelligence) which obtain values from  $\langle 0, 1 \rangle$ . According to [8] we can additionally define a Machine Task Intelligence Autonomy (MTIA) by equation

$$MTIA = \frac{MTI}{HTI} \quad (2.3)$$

MTIA equal to 0 is corresponding to the task performed completely by a human. Understanding of different levels of autonomy according to U.S. iRobot company can be seen in figure 2–2

Transferring these facts to the problem of teleoperation, task when operator fully controls behavior and decisions of a robot has MTIA equal to 0. On the other hand,

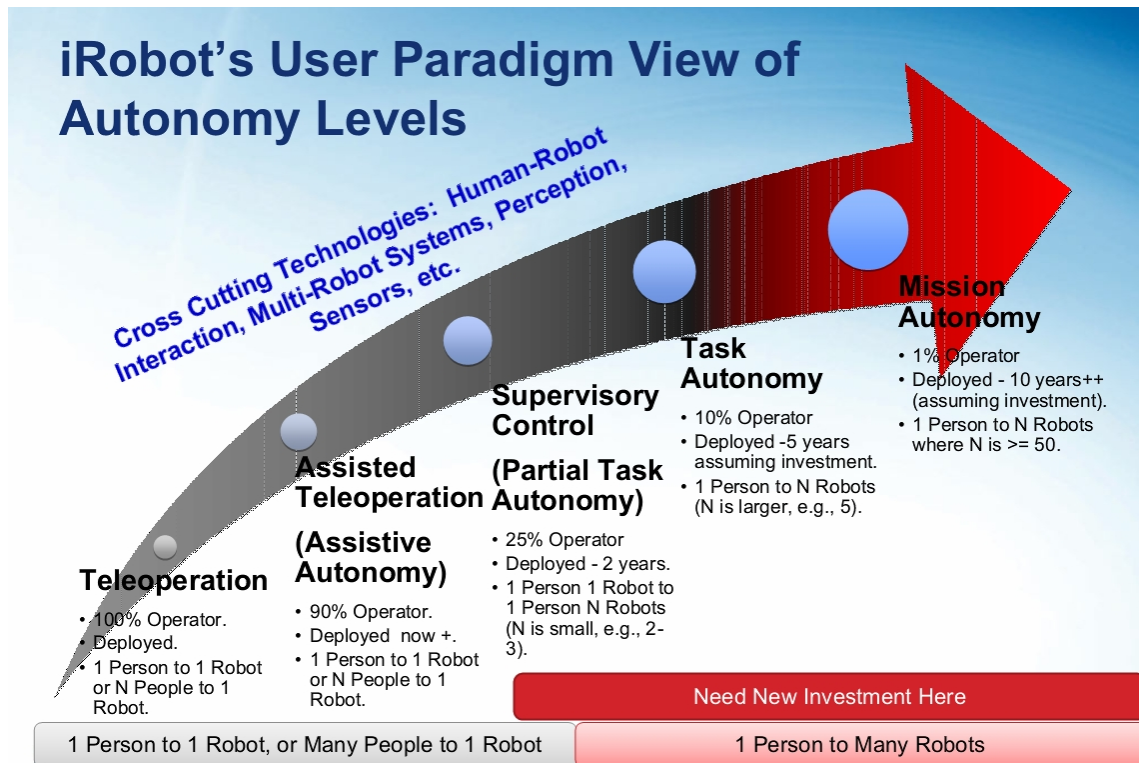


Figure 2 – 2 iRobot's view of autonomy levels.

if a robot has own inner decision model corresponding to the operator's, we are talking about MTIA equal to 1. In the following sections, the process of adapting the robot's decision model deliberately, using teleoperation to increase the level of MTIA, is called Learning from Teleoperation.



### 3 Learning from Teleoperation

*Learning from Teleoperation* means using the operator's demonstration to learn perform particular task similarly to the operator. Since the demonstration knowledge can be collected in different ways, where teleoperation is just one of them, we consider Learning from Teleoperation to be a subclass of the area called Learning from Demonstration (LfD). The best approach to acquiring and storing the experience during the learning from teleoperation is to collect information about a current state and operator's decisions in a form of tuples.

$$(state, action)$$

This means that every time the operator makes a decision, teleoperator collects information about his current state using its perception system and combines it with an action command. These tuples are then joined into a training set. The learning process then consists of two steps:

1. Demonstrating the desired behavior using teleoperation techniques and collecting training samples into the training set.
2. Generating appropriate function corresponding to operator's decision model.

The first step of the learning process consists of acquiring data from the demonstration. According to the [9], we distinct two common approaches to obtaining such data from robot:

1. **Teleoperation** - as mentioned before, technique where teacher (operator) operates a robot learner and the robot sensors directly records data.
2. **Shadowing** - a technique in which robot learner observes teacher's (operator's) motion using its own sensors.

The second step of the learning process consists of a function approximating and also extrapolating operator's behavior can be represented by different artificial intelligence methods. In the following sections, There will be introduced different approaches to this problem, explaining well or less known algorithms utilizing fuzzy systems, artificial neural networks or reinforcement learning.

### **3.1 Learning from teleoperation using Fuzzy systems**

The main task of the fuzzy system in the decision process is to take a role of a decision function, that maps a robot's state in a state space into an action it executes. The goal of the learning from teleoperation is finding such function, that corresponds to the decisions of the operator during the teleoperation. This decision function is represented by a rule set containing rules, in which predicate part is formed by fuzzified state information. Approximation of a decision function can be achieved in several ways. In addition to the mostly used iterative techniques like neural networks and evolutionary algorithms, [10] proposed another iterative approach, which extracts rules directly from the training data using hyperboxes.

#### **3.1.1 Expanding hyperboxes in the state space**

In the following algorithm, fuzzy rules extraction process consists of 2 steps. In the first step, each input vector in a state space is clustered placing a hyperbox around it and labeling it with a desired class (decision made by the operator). At this point, I need to emphasize, that the number of classes (actions) is finite. The iterative learning process is based on expanding the hyperboxes in a state space until they touch a hyperbox of a different class. When more hyperboxes of the same class are overlapped, they collapse together. Such a process minimizes an empty state space and also a number of conflicting zones. Each hyperbox has following structure:

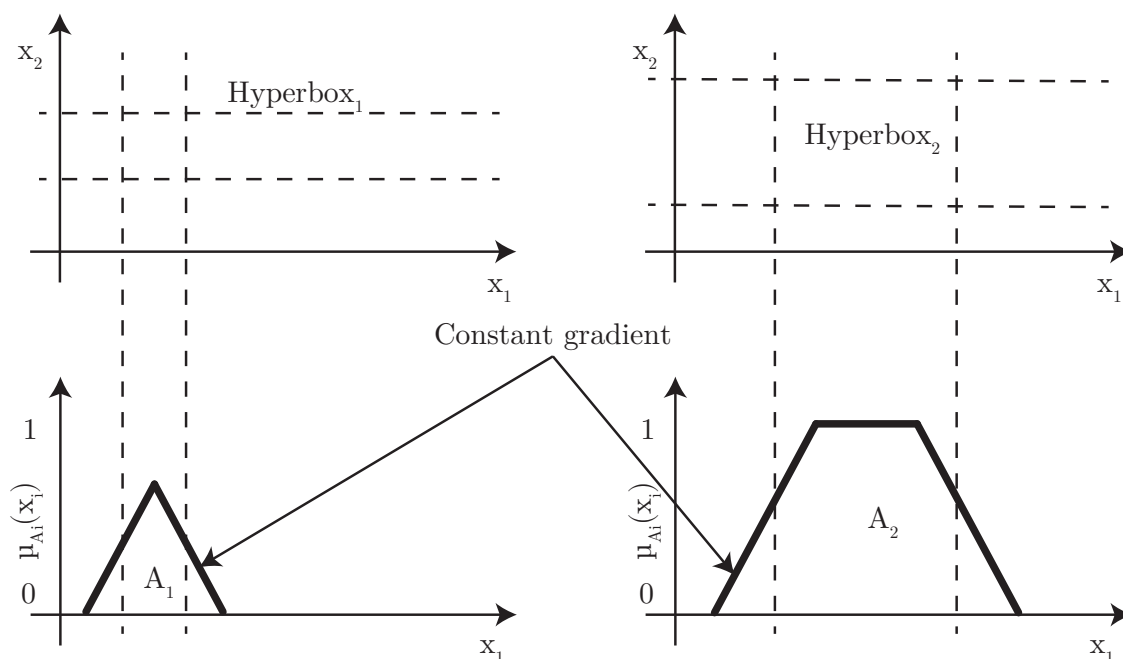
- Lower bound for each dimension in a state space.
- Upper bound for each dimension in a state space.
- Class label corresponding to an action decision of the operator.

The second step of the learning procedure is a derivation of the fuzzy rules from these hyperboxes. As can be seen in the figure 3–1, hyperboxes are transformed into trapezoidal or triangular membership functions according to each of dimensions bounds. This practically means, that if input state space has 2 dimensions, resulting fuzzy rule will be comprised out of 2 membership functions. The slope of the triangle’s (or trapezoid) side is a constant value, which guarantees the smoother operation of a robot. For a classification of an input vector with the fuzzy rules, the sum operator is used. Formal description of the classification process is expressed by equation (3.1), where  $n$  is a number of all inputs.

$$y = \frac{\sum_{i=1}^n \mu_{A_i}(x_i)}{n} \quad (3.1)$$

Where  $\mu_{A_i}(x_i)$  in the equation represents the degree of membership of input  $x_i$  to particular set (action)  $A_i$  and  $y$  is a global classification of whole input vector.

To determine the input vector’s class in the decision process, the rule which fires and accumulates the highest sum is accepted as the winner. The rule is considered to be fired, when average membership activation is higher than 0.5. If no rule is fired (empty space between hyperboxes), a default action is selected and triggered. The default action is usually by the number of occurrences in the training set. On the other hand, when several rules are fired at the same time, action is selected either randomly, or according to the previous decision to ensure consistency in the process.



**Figure 3–1** Conversion of hyperboxes into corresponding membership functions. The  $x_i$  in the figure illustrates a state variables, and  $A_{1,2}$  are particular classes(actions). The  $\mu_{A_i}(x_i)$  represents a degree of membership of state variable  $x_i$  to the corresponding class.

### 3.2 Learning from Teleoperation using Neural Networks

Nowadays, artificial neural networks are widely used in various learning tasks and learning from teleoperation is not an exception. In a previous section, I briefly introduced fuzzy inference systems as a suitable mechanism for simulating control function in robotic systems. Although fuzzy systems might seem to be effective, summary and easy-to-implement method, they usually lack a direct mechanism to learn from data. This deficiency is usually patched by utilizing other methods of artificial intelligence like neural networks.

Compared to the fuzzy systems in problems regarding learning from teleoperation, neural networks does not only take the place of a control function, but also of a universal learning approach. This learning approach can sufficiently approximate

a function from training data and also generalize it. The only deficiency is its complexity, which results in a human unreadable form of learned knowledge.

The most notable implementations of independently used neural networks are usually concerned to algorithms incorporating a haptic feedback. The main idea of these algorithms is a generation of so-called potential fields, that guide to a goal of a task simulating kind of forces dragging object closer to a target position. An example of such an algorithm is explained in [11], where authors utilize hierarchical neural networks to learn a manipulation task. It is important to mention, that these algorithms usually use a supervised type of learning.

Another approach to teaching a neural network is some kind of a transition between supervised and unsupervised learning techniques called reinforcement learning. This approach will be explained more in detail in the following section.

## 4 Reinforcement Learning

### 4.1 Introduction

The main idea of reinforcement learning is learning from interaction with an environment. An example of such learning might be a learning to play some musical instrument. At first, when one takes a violin in hands and try to make some tones, it probably ends up in a total disaster. If somebody else was listening to at a time, he would not probably praise it. However, if you practice a lot, it's natural that you improve and it is more and more probable, that you start receiving a praise. Although this example has nothing to do with artificial intelligence or computational methods, it is a strong allegory to fundamental principles of reinforcement learning.

Reinforcement learning is a subclass of the area called machine learning. Right at the beginning of this chapter, it is important to remember that problems regarding reinforcement learning are not specified by a method utilized, but by the task description. In general main subject of interest in reinforcement learning is an agent, existing in some environment with a passion for reaching a goal of a task. His performance is quantitatively evaluated by a total reward or punishment he receives while fulfilling the task. Comparing with the example, the agent is a potential violin player and the reward/punishment function is a spectator praising or squeaking the performer. The goal of reinforcement learning is to learn agent to perform a task with a maximal possible reward.

### 4.2 Basic terminology in Reinforcement Learning

Let's start with a previously mentioned terms, the agent and an environment.

**Agent** is also called a learner or decision-maker. In the process of learning, he is

the one who learns from interaction and is rated by the quality of performance. Everything else, which is during task possible to be interacted with is called an **environment**. Therefore, we can say that task is a loop starting with an agent taking an action, to which environment naturally reacts providing the agent new situations.

According to [7], despite these terms, we recognize another four necessary sub-elements. They are a *policy*, a *reward function*, a *value function* and finally a model of an environment.

A **policy**  $\pi$  is a kind of decision function, advising an agent what is the best action to do in a specific time step. Formally speaking, policy is a function  $\pi$ , that maps each state  $s_t$  into corresponding action  $a_i$ .

A **reward function**  $R$  is evaluator, that gives each state in which agent happens to be a number. The number is called a reward  $r$  and generally it tells us, how good was a transition to state  $s_t$ .

If reward signalized fitness of every immediate transition a **value function**  $V$  looks at a transition from a global scope. The transition means taking an action  $a_i$  in state  $s_t$ . If transition is suitable for the global task and brings agent closer to the target, value function generates higher values.

Finally, a **model** is a formal expression of the whole environment. It defines what happens after an agent takes an action and in what next state he appear. Knowing a model of an environment can also help us predict the future states. However, this knowledge does not have to be explicitly given, so there do exist methods that try to approximate the model in order to plan the future states and behavior.

To sum it up, when speaking of reinforcement learning we focus on a learner, an agent, who occurs in an environment described by its model, which can be unknown. Learning is a process of finding such policy  $\pi$ , which advise the agent the best action

$a_i$  in each step  $s_t$ . Evaluation of suitability of an action can be characterized either by immediate evaluator, the reward function  $R(s, s')$ , or a global, the value function  $V^\pi(s)$ . The learning process can be seen in figure 4–1

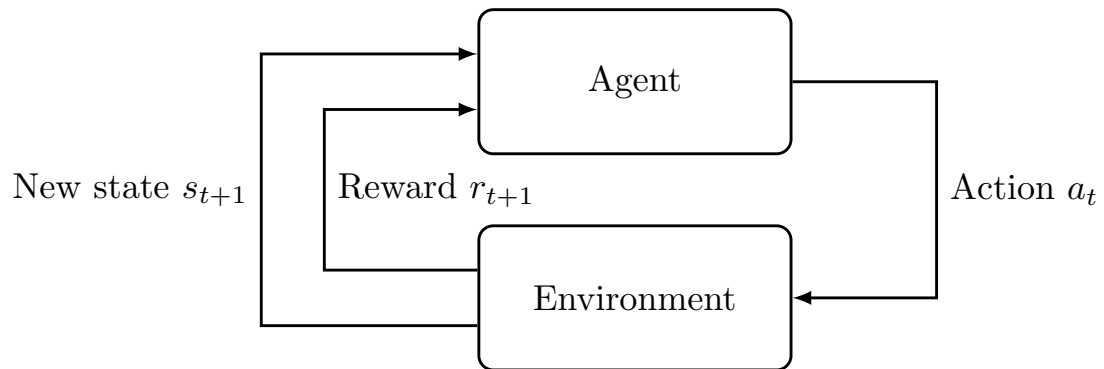


Figure 4–1 Reinforcement learning process.

### 4.3 Markov Decision Process

To formally define a Markov Decision Process (MDP), we need to supplement the previous section with some terms and properties.

Although term **actions** was used several times in the previous section, it is important to emphasize some facts about it. Actions or a set of actions is a finite set  $\{a_1, a_2 \dots a_i\}$  (might be infinite in some systems), where  $i$  is a size of action space. All actions are used to control the state, and **all** actions all applicable in each state in which an agent occurs.

However it was mentioned in a previous section, more formal definition of a term **transition** is also needed. By applying action  $a_i$  from action set in state  $s_t$ , an agent makes transition to state  $s_{t+1}$ , based on a probability distribution of a transition set. Transition function is defined by equation 4.1, and it expresses a probability of ending up in a state  $s_{t+1}$ , when taking action  $a_i$  in state  $s_t$ .

$$T : SxAxS \rightarrow \langle 0, 1 \rangle \quad (4.1)$$



Thus, *Markov Decision Process* can be formally described as a tuple  $\langle S, A, T, R \rangle$ , where  $S$  is a finite set of states,  $A$  is a finite set of actions,  $T$  is a transition function and  $R$  is a reward function. The transition function  $T$  and the reward function  $R$  together characterize a model of a system. The tuple is sometimes extended with parameter  $\gamma$  called a discount factor, which signalizes importance of current reward against future rewards.

System can be called *Markovian* if result of currently executed action is dependent only on the current state and not on the previous actions and states. In other words, this means, that every time an agent happens to be in a state  $s_t$  deciding for action  $a_i$ , probability of ending up in state  $s_{t+1}$  is always the same. This fact is also called a *Markov Property* and help us distinct MDP from other systems.

Resulting from the Markov Property, a problem generally solved by MDP can be defined. It is not finding a sufficient model of a system, as one might think, but finding a policy  $\pi$ . When a proper policy  $\pi$  is learned and combined with a decision system, we get a so called *Markov Chain*. Finding an optimal policy means finding such policy, that maximizes a cumulative reward acquired from a reward function while performing a specific task and starting from a random state  $s_0$ .

However, the theory of Markovian systems and decision processes was a subject of enormous number of papers, I consider this brief introduction to be a good starting point for the following sections.

## 4.4 General Reinforcement Learning Approaches

### 4.4.1 Dynamic Programming

The main idea of the dynamic programming (DP) methods in reinforcement learning is utilizing a *Value function*  $V^\pi$  in order to find an optimal policy. Let me define

the *Value function* more formally due to its importance in following lines.

As mentioned, a value function in reinforcement learning estimates how good it is for an agent, to be in a state  $s_t$  from the global point of view according to the task. To evaluate a state, value function approximates an expected cumulative reward if an agent is starting the task from state  $s_t$  and following the policy  $\pi$ . According to [7], formal definition of this function is expressed by equation 4.2.

$$V^\pi(s) = E_\pi\{R_t | s_t = s\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right\} \quad (4.2)$$

$E_\pi$  in this equation means an expected value returned, when an agent is following the policy  $\pi$  from time step  $t$ .

Although it wasn't mentioned before, in addition to the value function  $V^\pi$ , there exists another value function called an **action-value function** denoted as  $Q^\pi$ . Basically, if value function  $V^\pi$  approximates how good is to be in a state  $s_t$ , the action-value function  $Q^\pi$  approximates how good is to be in a state  $s_t$ , taking an action  $a_t$ , and again, following policy  $\pi$ . Meaning of *how good* is also represented by expected cumulative reward acquired at the end of the task. Formal notation of this facts is expressed in 4.3. Nowadays, the  $Q$ -function is a key element of many methods for solving the *reinforcement learning problem*.

$$Q^\pi(s, a) = E_\pi\{R_t | s_t = s, a_t = a\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\right\} \quad (4.3)$$

Back to the dynamic programming, two main components of DP methods are:

- **Policy evaluation**, which means computing a value function  $V^\pi$  from policy  $\pi$  and applying it on a task.
- **Policy improvement** that improves policy  $\pi$  based on behavior of value function  $V^\pi$

These two steps are repeated infinitely in a loop until convergence to optimal policy, however, this is possible only with a perfect system model. Even if dynamic programming methods present a necessary theoretical background for other methods, they are less utilized due to this limitation.

The first step in a dynamic programming methods is *policy evaluation*. A formal description of this process is expressed by 4.4 according to Bellman's equation. Equation 4.4 represents an iterative approach to solution of  $n$  Bellman's equations.

$$V_{k+1} = \sum_a \pi(s, a) \sum_{k'} P_{ss'}^a (r_{ss'}^a + \gamma V_k(s')) \quad (4.4)$$

$\pi(s, a)$  is a probability of taking an action  $a$  in state  $s$  according to the policy  $\pi$ . Function  $P_{ss'}^a$  represents a transition probability distribution ending up in a state  $s'$  and  $r_{ss'}^a$  is expected immediate reward.

Although we can determine the effectiveness of a current policy in a decision process, if we want to get to the optimal one, we need to make a decision whether to change the current policy or not. This decision can be made in a following way. Let an agent be in state  $s$ , evaluating  $Q^\pi$  value for each possible action. The best policy is then greedy selected as the one with highest  $Q$  value. This algorithm is described by equation 4.5 and it is called the policy improvement.

$$\pi' = \arg \max_a Q^\pi(s, a) \quad (4.5)$$

Starting with a policy  $\pi_0$ , whole process is expressed by equation 4.6 and it is called the policy iteration. Symbols  $E$  and  $I$  in the equation means the evaluation and the improvement process in the iteration.

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \dots \xrightarrow{I} \pi^* \xrightarrow{E} V^* \quad (4.6)$$

#### 4.4.2 Monte Carlo Methods

Opposing to the previously described DP methods, Monte Carlo (MC) methods does not require a full knowledge of the system in which an agent exists. The key element is on the other hand knowledge from experience, which an agent acquire while performing a task. Due to this fact, MC methods are only usable for the type of *episodic task*, which is a task where exists a final termination state. The process of navigating robot from a start state ending up in a terminal state is called an *episode*.

Although MC approach might seem entirely different from the DP, learning process again consists of two main components, the policy evaluation, and policy improvement. The main idea of the learning process is thus trying all possible options to find the best policy. According to this idea, policy evaluation process now estimates action-value function  $Q^\pi$  averaging the *results* from all trials starting from state  $s_0$ . Note that the *result* is meant to be a cumulated reward after getting to the terminal state. Policy improvement process now can also be described by equation 4.5 from the previous section.

The problem with this approach may occur when strictly following the policy. Exploration process would then ignore some states and actions that might also be very good. The solution to this problem is utilizing a so-called  $\varepsilon$ -greedy policies. Following this policy means to decide for action according to the policy  $\pi$  with a probability of  $(1 - \varepsilon)$  and for random action with a probability of  $\varepsilon$ . Appropriate selection of parameter  $\varepsilon$  (which doesn't need to be constant) then ensures the exploration process as well as an optimization process.

The control process of Monte Carlo methods is represented by the equation 4.7.

$$\pi_0 \xrightarrow{E} Q^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} Q^{\pi_1} \dots \xrightarrow{I} \pi^* \xrightarrow{E} Q^* \quad (4.7)$$

### 4.4.3 Temporal Difference methods

As written in [7], if one idea in the reinforcement learning is central and novel it is certainly the Temporal Difference (TD) learning. It is a combination of main ideas from MC and DP methods. We can define a TD learning as an approach, which does not require the strong mathematical formulation of the system and can also learn from values of successors. In addition, its importance in a real world application lies in its capability of learning continuous tasks as well.

When looking back at the MC methods, the value function iteration is described by equation 4.8.

$$V(s_t) \leftarrow V(s_t) + \alpha[R_t - V(s_t)] \quad (4.8)$$

In the case of MC methods this means that  $V(s_t)$  could be evaluated only after end of episode (when we know the final cumulated reward). On the other hand, in the case of TD learning 4.8 can be rewritten to equation 4.9.

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \quad (4.9)$$

This means that new  $V(s_t)$  can be updated after every step according to the successor represented by  $r_{t+1} + \gamma V(s_{t+1})$  in the equation.  $r_{t+1}$  in the equation is an immediate reward in step  $t + 1$  and  $V(s_{t+1})$  is a value in this step. This equation is also called *TD prediction*.

### 4.4.4 Q-Learning

Previous sections were focused on the most common approaches to reinforcement learning, which are not usually implemented, but take an important role for the most of other advanced algorithms. However, the following algorithm cannot be called a basic approach because it comes out from the previously mentioned approaches.

It is included in this section because it will take a major role in next chapters and experiments.

*Q-learning* or also referred as Off-Policy Temporal Difference Control is a method that does not rely on the policy being followed. In the other words, new  $Q$  function is approximated only by the previously learning  $Q$  function. As an alternative name signalizes, this algorithm is a member of TD methods family. We can formally describe the Q-Learning by equation 4.10.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \arg \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (4.10)$$

According to Bellman's optimality equation mentioned before, it is guaranteed that this iteration converges to the optimal  $Q^*$ , if all  $(s, a)$  pairs are updated infinitely many times.

Practically speaking about the implementation of this method, action-value function  $Q$  can be represented as a multidimensional array. If we simplify denotation of state to  $\mathbf{s}_i = (s_{i0}, s_{i1} \dots s_{in})$ , the finite set of states where  $n$  is also a finite constant, we can express the  $Q$  function in a following way according to 4.11.

$$Q = \begin{matrix} & a_1 & a_2 & \dots & a_n \\ \mathbf{s}_1 & \left[ \begin{array}{cccc} q_{11} & q_{12} & \dots & q_{1n} \\ q_{21} & q_{22} & \dots & q_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ q_{m1} & q_{m2} & \dots & q_{mn} \end{array} \right. \\ \mathbf{s}_2 & \\ \vdots & \\ \mathbf{s}_m & \end{matrix} \quad (4.11)$$

The learning algorithm goes in the following order:

- *Initialization* of learning parameters  $\gamma$  and  $\alpha$ ,  $Q$ -table randomly and a reward function.
- For each episode initialize a random state and do while target state is reached:
  1. Select an action  $a_i$  according to policy derived from current  $Q$ -function.

2. Execute selected action  $a_i$ .
3. Observe new state  $\mathbf{s}$ .
4. Update the  $Q$ -function according to the update rule 4.10.
5. Update current state with a new state.

## 5 Reinforcement learning in Teleoperation

The following section discusses algorithms that utilizes the reinforcement learning and might find application in problems regarding learning from teleoperation.

### 5.1 General scheme of learning

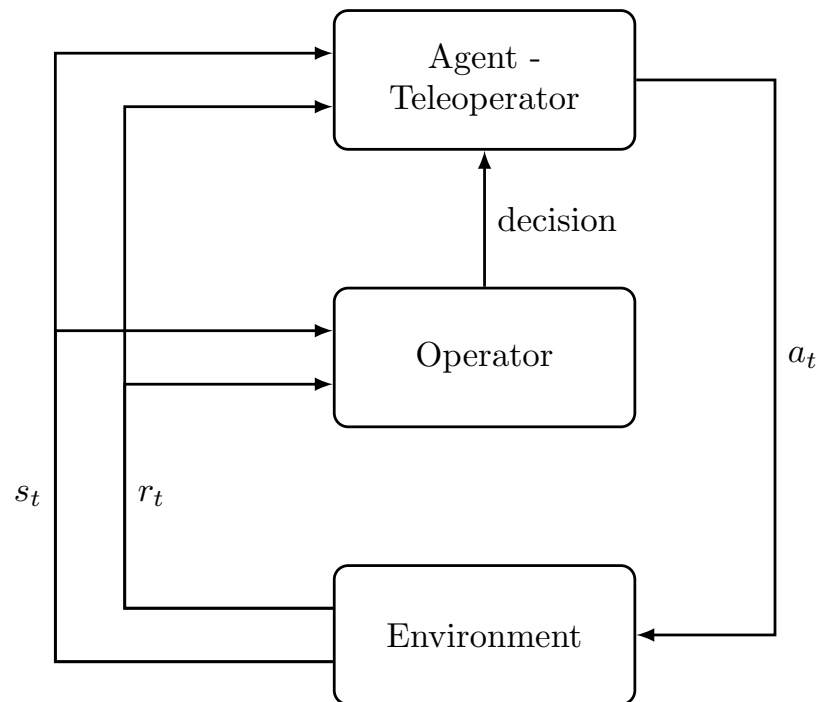
In the beginning, it seems appropriate mentioning general terminology, which is interpreted differently in both approaches, but has almost the same meaning. I would like to emphasize an allegory in terms teleoperator and an agent. In both cases, the term means some object that executes actions in its environment. In the case of teleoperator, the decision process is delegated to the operator, who is supervising the whole system. On the other hand, in the case of reinforcement learning, an agent is taking decisions on himself according to the strategy. When talking about an environment, it can be represented similarly in both cases. The question now stands, whether we can fit an operator somewhere in the reinforcement learning process described by figure 4–1. The figure can be modified in a way, which can be seen in figure 5–1 [12]. As can be seen, the operator is placed right in the middle, between the environment and an agent. Learning process then goes in the following way:

- Operator is informed about the current state in which an agent happens to be.
- Operator (usually a human) makes a decision about the action (knowing all possible actions).
- Agent executes the commanded action.
- Environment derives new state according to inner model of dynamics.



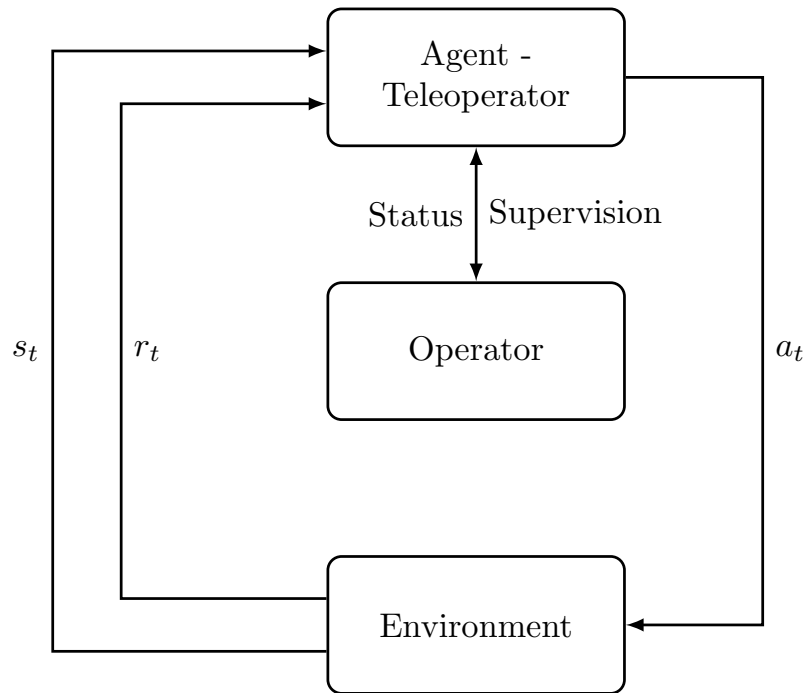
- Inner decision model of Agent is updated according to reward and implemented learning algorithm.

The goal of the learning process is visually described in figure 5–2. As can be seen, decision process does not go directly through an operator. Instead, the operator takes the position of a supervisor, who just looks after the decisions made and in a critical situation can change the decision. Despite this, the system might be fully autonomous on the particular task. In the learning process, the influence of the operator in the decision process should be decreased until an agent can decide well enough. In this step, the agent is considered to by just a supervisor of the system.



**Figure 5–1** Utilizing Teleoperation in process of Reinforcement Learning to learn decision strategy according to human.

$s_t$  in the figures 5–1 and 5–2 stands for a state observed in a time step  $t$ . The  $r_t$  is a reward observed and  $a_t$  is an action selected in the time step  $t$ . These variables corresponds with nomenclature used in chapter 4. Following sections introduce algorithms, which can be used in a learning process.



**Figure 5–2** Operator in a position of a supervisor, after system has learned the decision model.

## 5.2 GARIC

### 5.2.1 Overview

Name for this architecture comes from Generalized Approximate Reasoning based Intelligent Control, and it is something an upgraded version of ARIC architecture proposed in [13]. Generally speaking about the GARIC proposed in [14], it is a joint of 2 neural networks called ASN (Action Selection Network) and AEN (Action Evaluation Network), and a system for adjusting final action called SAM (Stochastic Action Modifier). General scheme for this architecture can be seen in figure 5–3.

### 5.2.2 Action Evaluation Network

The AEN is a network that evaluates the state of the controlled system and rate it to predict the future reward acquired. Similarly to the Q-learning, where  $Q$  value

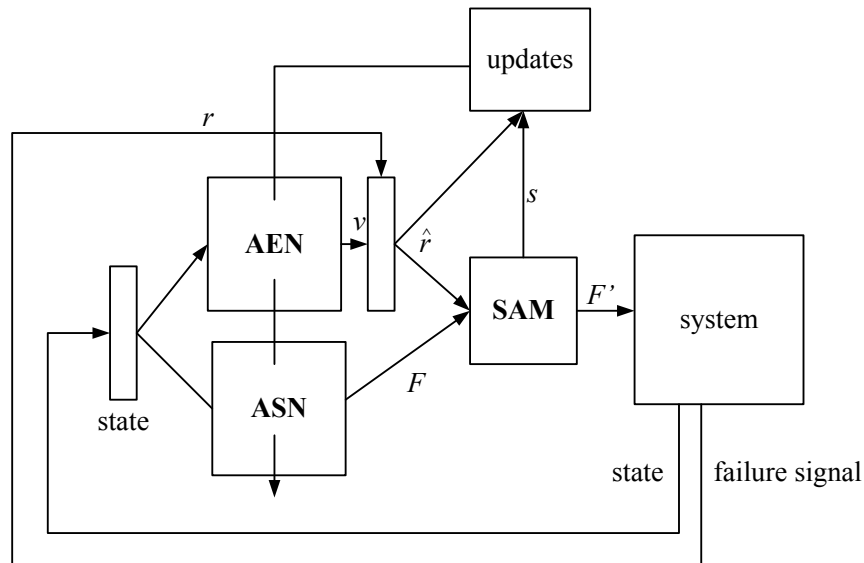


Figure 5–3 General architecture of GARIC.

predicted the final reward, this network is also used as a predictor, although it will not predict the cumulative reward, but only single future reward. The problem that this part solves is also called *multi-step prediction* and is solved by so-called  $TD(\lambda)$  algorithm derived from Temporal Difference method mentioned earlier.

The structure of the network can be seen in figure 5–4. Input layer accepts state vector and a random bias, which can help network learn desired mapping faster. Output from this network is prediction of reward  $r(t+1)$ . Atypically, output neuron is not connected to just last hidden layer, but also to the input layer.

To combine this algorithm with a **teleoperation**, the learning process could be divided into 2 stages similarly to the general scheme mentioned earlier. In the first stage, decisions are made by operator and rewards returned by this network should be higher, as far as we assume that decisions of the operator are always correct. To reach the stage two, where the operator is in a position of a supervisor, we should decrease the ratio of decisions made by the operator and an agent. In addition, the system should produce lower rewards for decisions made by the agent, due to its controversy.

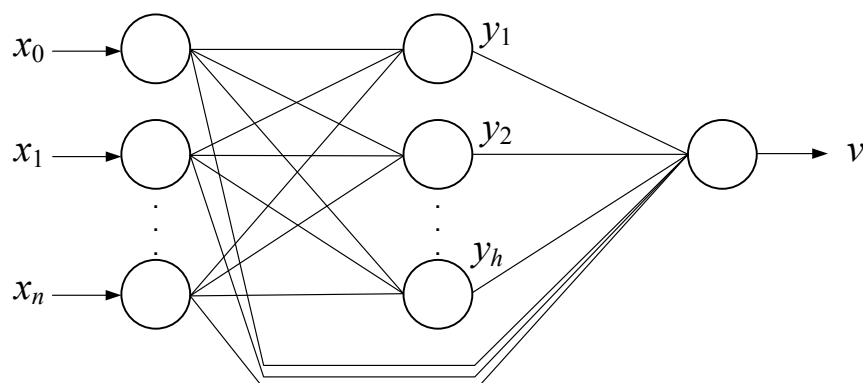


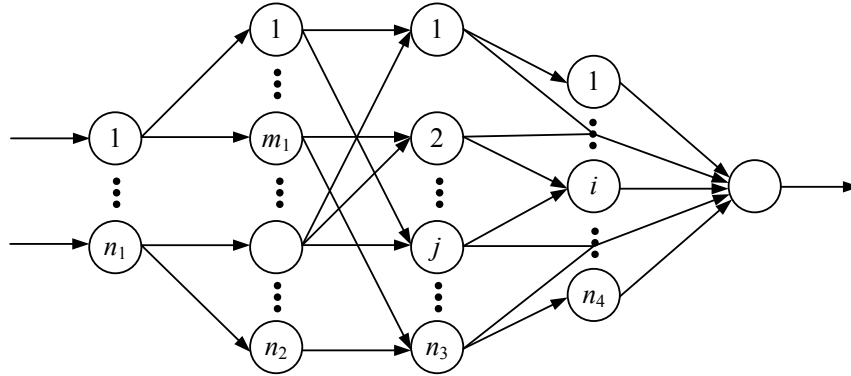
Figure 5–4 Structure of AEN module in GARIC.

### 5.2.3 Action Selection Network

The main goal of this network is to select a proper action for controlled system. In other words, this network is a fuzzy controller in a form of a neural network. The network, as it is, consists of 5 layers that can be seen in figure 5–5.

The first layer is used just for the input of crisp values. These neurons, or better-said nodes, are connected to few neurons in the second layer, where these crisp values are fuzzified. Each of the nodes in the second layer fulfills a task of being a membership function for the corresponding input value. Nodes in layer 3 are then used as a rule base for this fuzzy controller and layer 4 holds membership functions for the output variables. At the end, there is one node, which is used for a defuzzification of the output.

This network is not learned traditionally by changing the weights between nodes, but instead by adjusting the shape and position of membership functions. In the original algorithm described in [14], the network is trained without the presence of any supervisor and desired behavior is learned only according to AEN module prediction. This algorithm could be modified so that it accepted supervision from an operator. According to the scheme described in previous section, in the first stage of learning process, decisions would be made by a human operator. Information



**Figure 5–5** Internal structure of ASN module of GARIC architecture.

collected from the environment would then be used as training samples for ASN. After the network was learned, the ASN could take full control of decision process.

#### 5.2.4 Stochastic Action Modifier

The SAM combines knowledge from the AEN and ASN, using it to produce real action  $F'(t)$ . It uses a Gaussian random variable with mean  $F(t)$  – action from ASN, and standard deviation that is a function of reinforcement from the AEN. Even though originally designed system contained this part in [9], it was proved in [10] that this part can be completely omitted.

### 5.3 Neural Fitted Q Iteration

Speaking of all the reinforcement learning approaches mentioned before, it was strongly emphasized that the set of all states and actions *must* be finite. This restricts us to the narrow set of application with discrete and not too large state space. However, real world application usually requires continuous state space description with infinitely many states. This requirement thus forces us to look for learning alternatives or methods for discretization of state space utilizing various the discretization grids of fuzzy logic.

A problem with these methods arises when discretization is too firm. This may fail the whole learning process due to unrecognizable states so learning the optimal policy would be impossible. On the other hand, when discretization is too precise, a problem may occur with too many states in many dimensional state spaces and thus make system lose the ability to generalize decisions.

### 5.3.1 Main concept

As the title of the method suggests, the Neural Fitted Q Iteration (NFQ) algorithm [15] is a derivation of  $Q$ -learning mentioned earlier. According to the  $Q$ -learning, the  $Q$  function is represented by the multidimensional array and thus is also lacking the applicability in real world problem. The main idea of NFQ is the improvement of the algorithm, replacing the array function with a function approximated using a neural network. This basically removes any need for a state space discretization method without information loss.

To truly understand the meaning of the  $Q$ -value in the decision process, we need to look back at the definition of the  $Q$ -function. According to it,  $Q$ -value is a final reinforcement acquired after reaching the goal state  $S^+$ , when taking an action  $a$  in state  $s_t$  and following the *optimal* policy  $\pi^*$  after then. The update of the  $Q$ -function is then realized by expression 4.10 which will serve as a desired value in neural network training. When looking at the equation more in detail, we can see that the current  $Q$ -value is dependent on the  $Q$ -value from the next step and evolving this sequence brings us to the **recursive relation** between current state  $Q$ -value and the  $Q$ -value at the target location.

However the function is represented by a neural network, the update process has to be adjusted for its purposes. When we were talking about basic  $Q$ -learning, we updated the function each time an agent made a step. This approach is, unfortunately, not sufficient in the case of a neural network, because learning just one

sample would cause unpredictable changes in knowledge learned before. Thus, to learn the network, we first have to collect training data in a set and then learn the network on it.

### 5.3.2 Basic NFQ Algorithm

The first difficulty in this algorithm is the acquisition of sufficient data. When looking back at 4.10, we notice that to evaluate the equation, we need to collect current and the next state, action selected and optionally also a reward. Thus first step in NFQ is gathering of training tuples  $[s_t, a, s_{t+1}, r_{t+1}]$ .

The new  $Q$ -function can be trained only when the sufficient amount (differs according to the task) is collected. To do this, each training sample is transformed into neural network training pattern, so that the input information consists of state vector  $\mathbf{s}$  and an action number, which is usually encoded in some way. In my experiment, the action number is transformed into an array, which length is equal to total number of actions available. The array then contains 0's, and 1 is placed in the position of the action number, e.g.  $a_{3of4} \rightarrow [0 \ 0 \ 1 \ 0]$ .

To get the desired value of the training pattern for the output of the neural network, we evaluate the equation 4.10 using state and action information from the sample. The  $Q$ -value for the next state is then evaluated using the current neural network. A problem may occur in the evaluation process because standard sigmoid activation functions usually can't work with values higher than 1. Due to this fact equation 4.10 is transformed to the 5.1 setting the learning parameter  $\alpha$  equal 1. The main principle behind this transformation is that we do not look at the reinforcement learning problem as a maximization of a cumulative reward, but instead minimization of cumulative punishment. Thus  $Q$ -value equal to 0 is the ideal one.

$$Q(s_t, a_t) \leftarrow r_{t+1} + \gamma \arg \min_b Q(s_{t+1}, b) \quad (5.1)$$

**NFQ** algorithm then works in a following steps:

1. Training samples acquisition.
2. Random network initialization.
3. While sufficient  $Q$ -function is found or maximum epochs reached:
  - (a) Transformation of samples into training patterns evaluating the updated  $Q$ -values.
  - (b) Learning of network using these data.
  - (c) Optional acquisition of additional data samples.

### 5.3.3 Utilization of NFQ in Teleoperation

The NFQ learning scheme is an excellent mechanism for learning tasks in the real world with the continuous state space. In the previous section, I briefly described the learning mechanism and left an open space for a method of training sample acquisition. In common NFQ approach, these samples are collected during agent's greedy environment, but to increase the efficiency of the training algorithm, we can replace this mechanism with a teleoperation. This method will be described in detail in following section, which is dedicated to the implementation of this algorithm and testing in on so-called *predator and prey* task with a humanoid robot.



## 6 Experiment

In the following sections, I will describe an experiment I made utilizing the early mentioned NFQ algorithm merged with teleoperation technique. The experiment was tested in simulator environment Webots on humanoid robot NAO from Aldebaran Robotics.

### 6.1 Task description

The main idea behind the experiment is a test of a proposed method on some, ideally an easy, task. I consider an easy task to be the task, in which state representation could be easily collected from a learning process, and action set should be sufficiently descriptive even for a layman. Another parameter, which this task should fulfill is its measurability from the perspective of succeeding in the learning process. The last but not the least important parameter is the small values of initial MTIA described in 2.3.

According to these criteria, I selected a so-called *Predator-Prey* task. Agent in this task is in a role of predator following some (static or dynamic) target. Due to simplicity, at first, consider the target to be some position in a state space marked with a ball on it. Thus, the training task will be a robot trying to get to the ball. We can conclude that this task will have an initial MTIA equal one because the robot will not be able to approach the target position without any human experience deliberately. Status information will be collected in Webots simulator environment and saved into the database, what meets the requirement for easy access.

The learning process of this task will consist of *episodes*, where the initial state of a robot and position of the target will be randomly chosen. Agent will be learning from the experience observed from the human teleoperation. A human operator

will be able to control the decision process using an interface in his browser. The optimal policy is considered to be found when an agent is capable of reaching the goal state in a reasonable number of steps all by himself.

## 6.2 Technology used in experiment

### 6.2.1 Webots

Webots simulator from Cyberbotics is a development environment providing a simulation of different kind of robots. This simulation tool provides an enormous amount of customization to model the real world situation. A developer can choose from various types of objects with the ability to change their properties like shape, color or even physical properties to adjust their behavior. The whole world can be monitored with a broad spectrum of sensors.

The basic concept of programming in Webots lies in creating controllers for each object. These controllers can be programmed either using a built-in IDE or some third party application, and Webots provides a freedom in selecting one of the four accepted programming languages. A developer can choose between a C/C++, Java, Python or Matlab and the API provided is almost the same.

The most important part in Webots and a reason why I chose it for the purposes of the experiment is a module called a *Supervisor*. Generally speaking, *supervisor* behaves in a same way as a normal robot but it is not physically present in the world simulation. It can monitor the status of all other objects in the simulated world and also change their properties during runtime. In my experiment, the *Supervisor* module will be used for collecting the state data and (utilizing a programmed controller) uploading it to the database.

### 6.2.2 Redis

Redis is a fast open-source database that is a member of NoSQL databases family. This means that the data stored in the database are not ordered in tables with some relations but ,on the other hand utilizes so-called key-value approach. This method is very appropriate for purposes when the data model is often changed what my case is. Redis provides API for all of the major programming languages including Python and C, which form a base of experiment application.

Experiment data stored in a database contain state information in a numeric format, an overview of current episode and steps made and finally also a neural network structure. Complex structures like neural network weights are stored in a JSON format, which is widely accepted standard.

### 6.2.3 NAO

NAO is a French humanoid robot developed by company Aldebaran Robotics, which released the first version in 2008. This platform has been widely used in an academic field since then including our laboratory. NAO's hardware provides 25 degrees of freedom and can be programmed in c. ten languages including Python. Robot brings a user-friendly and easy-to-implement API, which is one of the reasons why I chose this robot for the experiment. As mentioned before, a fundamental part of the experiment is programmed in Python, and thus API is fully compatible with it.

NAO in the experiment will take the place of an agent, whose action set will consist of walking in different directions and rotations.

## 6.3 Reinforcement learning from teleoperation

In the previous chapter, I tried to briefly outline an idea behind the combination of teleoperation and reinforcement learning. The point was to develop a mechanism that will unburden a human in performing some task, utilizing his knowledge and also accelerating the learning process. When speaking about an utilization of human operator's knowledge, it is important to mention that the operator is often not a specialist in the field, thus it comes handy to develop a method that is user-friendly enough even for a layman. When talking about conventional machine learning approaches the knowledge usually has to be in strictly mathematically formulated. Utilization of *teleoperation* techniques in the learning process helps to overcome this problem.

On the other hand, when talking about common reinforcement learning approaches like MC methods mentioned in the 4.4.2 large amount of training experience is required to achieve at least satisfactory results. Utilizing a *teleoperation* in the acquisition process again helps to overcome such an issue.

For the purposes of this experiment, I have chosen approach described in 5.3. It is a modification of  $Q$ -learning so that it can be applied even in a continuous state space. The great advantage of this method is its independence of formal environment model and also on a policy. A fundamental part of the whole algorithm is a neural network that is expected to learn the decision model from provided training samples and generalize it to the whole state space.

### 6.3.1 Modified NFQ Algorithm

Even though an algorithm might seem perfectly ready for implementation in this task, a problem may occur in combination with teleoperation. The main concept of this algorithm is updating the  $Q$ -function infinitely often from the samples acquired

until the optimal one is found. This means that in earlier stages of learning process, when  $Q$ -function is not trained well enough, so the situations, when an agent decides for an action that might seem incorrect from the operator's point of view may occur. This might cause a serious problems in the neural network learning process in a form of inconsistency in training data. To overcome this issue, I propose utilization of teleoperation in these early stages to prevent the inconsistent data, as far as we consider every operator's decision to be the correct one.

On the other hand, if we forced an agent to accept exclusively operator's orders during the whole learning process, we would not be able to monitor its progress. Due to this fact, it seems logical to distribute the ratio between the operator's and an agent's decisions. To prevent collisions between operator's and agent's decisions, I have modified the basic algorithm so that when an agent's decision is made, training pattern is created according to the 5.3 algorithm. On the other hand, when operator's decision is made, a pattern for each of available actions is created according to 6.1. This modification has rapidly accelerated the learning process and also suppressed the inconsistency in training data.

$$Q(s_t, a_t) = \begin{cases} r_{t+1} + \gamma \arg \min_b Q(s_{t+1}, b) & \text{for action selected by an operator} \\ r_{t+1} + \gamma \arg \max_b Q(s_{t+1}, b) & \text{for every other action } a \in A \end{cases} \quad (6.1)$$

Another inconvenience that may occur during the learning process is also related to the neural network training. Although the author of the paper [15] also points at this problem, I would like to go a bit deeper. As he states, the main problem with the conventional approach is an online update of  $Q$ -function. Proceeding this way in neural network practice would cause unpredictable behavior of the network. This means that a small change in one area of state-action space may cause a huge change in the entirely different part of it. Due to this, the author suggests using more sophisticated batch learning approach, in which training samples are collected

in batch and weights are updated to the cumulative error over the whole batch. The problem may, however, persist when the sample set is generated every training episode. This might lead to loss of knowledge learned during previous episodes. Due to this, I am stacking training samples into one, and before every episode, training patterns are generated from this samples according to 5.3 and 6.1.

The fact that I am utilizing the batch learning instead of common online learning is providing me ability to use advanced learning techniques. One of them is a method called RPROP and is also suggested by [15]. Utilizing this method helped me shorten the neural network training process from 100 000 training epochs to just 500, what also helped accelerate the whole process.

### 6.3.2 RPROP Training

Name of the algorithm RPROP comes from *resilient propagation* and as an author in [16] states, it is a new approach to learning that rapidly increases the training time. In this section, I would like to briefly discuss the main idea of this algorithm and my experience with it.

In a conventional approaches in multilayer perceptron learning (like Backprop), the weights are updated according to the well-known equation 6.2. The performance of the network is so crucially dependent on a choice of  $\gamma$  parameter and the size of the error. RPROP helps to overcome this weakness by relying not on the size of this variable, but only its sign.

$$w_{ij}(t+1) = w_{ij}(t) - \gamma \frac{\partial J(t)}{\partial w_{ij}(t)} \quad (6.2)$$

The update rule is formally described by equations 6.3 and 6.4, where  $\Delta_{ij}$  stands for so-called *update-value*.  $\eta^+$  and  $\eta^-$  are learning constants experimentally set to 1.2 and 0.5.

$$\Delta w_{ij}(t) = \begin{cases} -\Delta_{ij}(t) & \text{if } \frac{\partial J(t)}{\partial w_{ij}(t)} > 0 \\ +\Delta_{ij}(t) & \text{if } \frac{\partial J(t)}{\partial w_{ij}(t)} < 0 \\ 0 & \text{else} \end{cases} \quad (6.3)$$

$$\Delta_{ij}(t) = \begin{cases} \eta^+ \Delta_{ij}(t-1) & \text{if } \frac{\partial J(t)}{\partial w_{ij}(t)} \frac{\partial J(t-1)}{\partial w_{ij}(t-1)} > 0 \\ \eta^- \Delta_{ij}(t-1) & \text{if } \frac{\partial J(t)}{\partial w_{ij}(t)} \frac{\partial J(t-1)}{\partial w_{ij}(t-1)} < 0 \\ \Delta_{ij}(t) & \text{else} \end{cases} \quad (6.4)$$

As mentioned, even though the implementation of this algorithm brought rapid acceleration of the training procedure, it also brought some inconveniences. The first problem of this algorithm is its complexity. In the algorithm's implementation this means, that instead of storing one multidimensional array of weights, one has to store also the *update-value* for each weight and also a trace of the previous error.

Another issue appeared when training on flat error function surfaces. This method has a high tendency to end up in some local minimum and no mechanism to get out of it. I solved this problem experimentally when the problem disappeared after adding another hidden layer of neurons or just enlarging the current layer. This practically meant that to reliably learn the net on training XOR problem, I had to change the configuration from 2-2-1 to 2-5-1 which also brings a performance issues. Due to this, computation part of the neural network and even of a reinforcement learning algorithm is written purely in C and built with optimization libraries. Table 6-1 describes the comparison of RPROP and conventional Backprop performance on learning simple XOR problem. The task was to reach the same precision on the same machine, with the same initialization. As we can see the performance of RPROP is dramatically better at the expense of reliability.

**Table 6–1** Comparison of RPROP and Backprop learning algorithms.

Name	Configuration	Epochs	Training time	Success Rate
Backprop	2-2-1	$10^6$	avg. 0.5s	> 95%
RPROP	2-2-1	150	< 1ms	$\approx$ 85%
Backprop	2-5-1	$10^5$	avg. 0.1s	100%
RPROP	2-5-1	85	< 1ms	> 95%

## 6.4 Experiment setup

As far as I am modeling a reinforcement learning task, let me first connect theoretical terms with specific objects and properties in the simulator.

*Environment* of the task will be formed by an infinite simulated stone floor without any obstacles in it.

As mentioned earlier, the *agent* will be represented by a simulated NAO robot and his *goal* will be to approach a target position in the *environment*. The *target* position will be marked with an orange ball on it.

*State* information will consist of 2 variables. First will be a direct distance of the agent from the target position. The second one will be an angle  $\alpha$  between the orientation vector of the robot and a flow line between the center of the robot and the ball. Values of these variables are also normalized so that they lie in interval  $\langle 0, 1 \rangle$ . In the figure 6–1, we can see the state information visually described on a screenshot from Webots environment.

*Actions set* will at the beginning consist of three motions. NAO robot API provides a method for moving the robot to the target position specifying three necessary parameters that are distance walked in the x-axis in meters, in the y-axis and a rotation of robot  $\theta$ . According to this method setup, first action will have parameter  $(0.25, 0, 0)$ , second  $(0.1, 0.1, \pi/6)$  and last  $(0.1, 0.1, -\pi/6)$ . This will represent



walking directly or turning either to the right side or the left side.

A *reward function* or maybe better said *punishment function* will have the following setup:

$$r_{t+1} = \begin{cases} 0 & \text{when } s_{t+1} \in S^+ \\ 0.1 & \text{else} \end{cases}$$

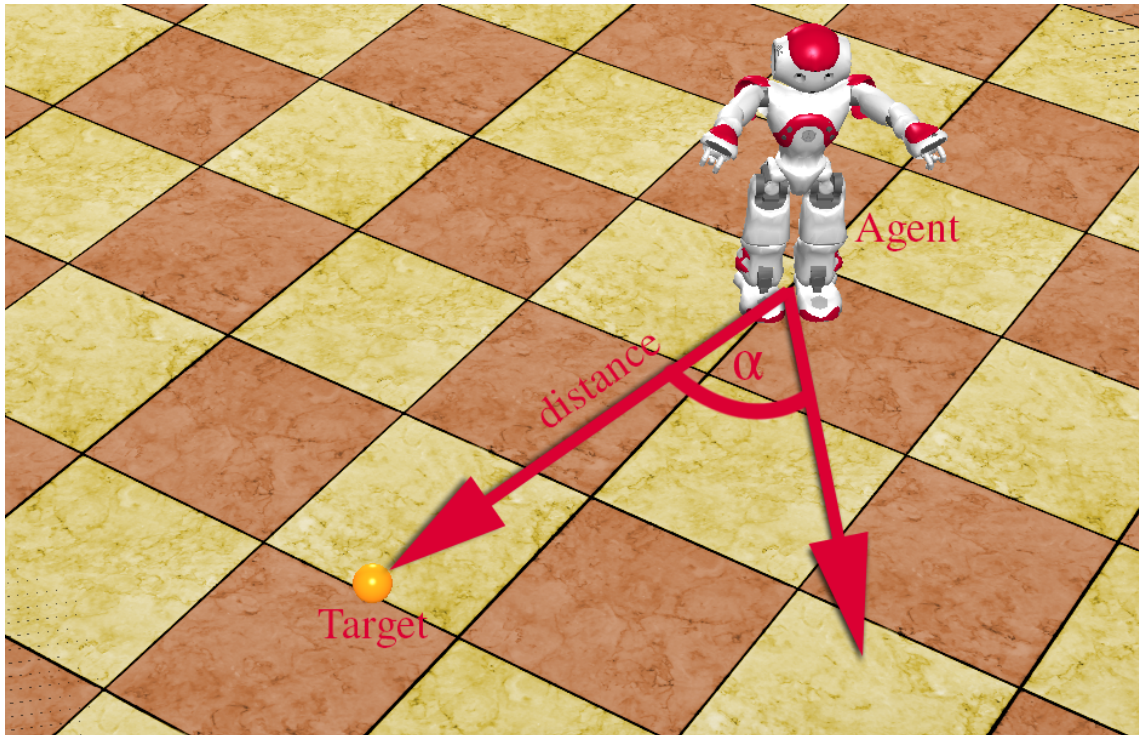
This basically means that the goal is to minimize a cumulative reward and approach a target in a minimum number of steps. State  $S^+$  is achieved if the robot is maximally 0.5m away from the ball and his rotation angle is less then  $30^\circ$  (assuming that direct angle is equal  $0^\circ$ ).

The *discount parameter*  $\gamma$  will be equal to 0.99.

As mentioned before, for the purposes of function approximation I have used a conventional multilayer perceptron trained by innovative RPROP method. Due to the predisposition of this method to end up in a local minimum, I have chosen larger network setup with 5 inputs (2 state variables, 3 action encoding places), 2 hidden layers with 20 neurons (I experimentally verified that 10 is also enough) and 1 output neural representing the  $Q$ -value for current state-action pair. Each training cycle consisted of 1000 training epochs, which also experimentally turned out to be sufficient. The fewer training epochs are also important due the fact that training samples are collected over time and stacked, so the training time is increasing exponentially with each episode.

## 6.5 Experiment evaluation

In this section, I would like to analyze the experiment from 2 main aspects. First of them is its asset in the area of incremental learning and its impact on the whole performance. The second aspect is regarded to a gradually decreased involvement by human in the achieving the goal of a task.

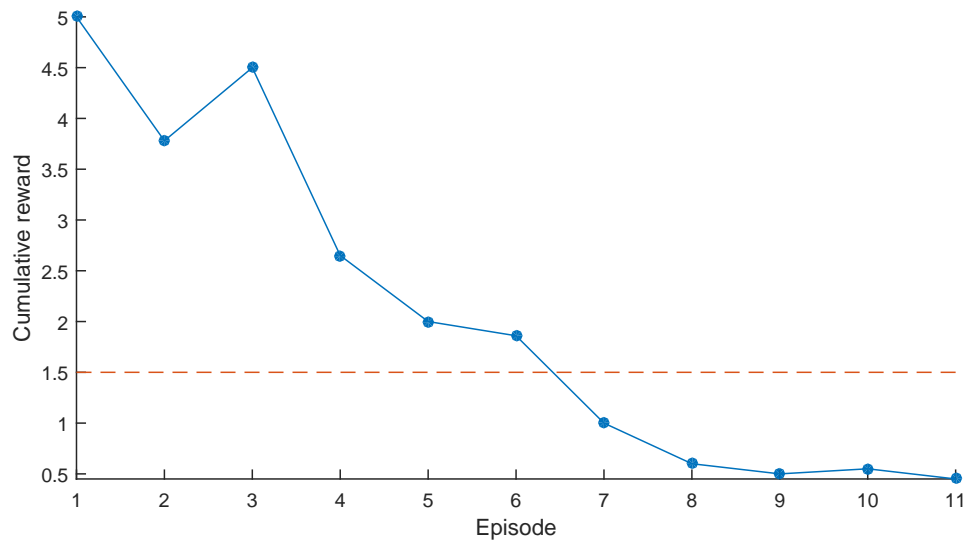


**Figure 6 – 1** State information in the experiment

According to the first aspect, the point was to evaluate objectively agent's performance after each episode. According to the definition mentioned earlier, an episode starts from a random position in a state space and ends after reaching the target area. In the experiment, the robot is placed randomly throughout the floor (also randomly rotated), and operator navigates him to the target. When the target area is reached, the neural network is updated. After each of these episodes, a testing process is also launched, with the following functionality. The robot is placed at the testing spots, and his autonomy is tested utilizing his own decision policy. A quantitative measure in this experiment is an average cumulative reward received during the test. Various initial positions are selected as a testing spots to test the policy in various situation (far and close places, facing the target or turned back). Of course, the testing spots are always the same after each of the episodes.

In the figure 6–2 we can see a progression of task learning process. The y-axis

represents the average cumulative reward acquired during the test after each of the episodes. I set the upper bound of the reward to 5, which corresponds to approximately 50 steps. If the agent was not able to approach the target in this limit, the attempt was considered to be failed, and the agent was straying. The dashed line in graphs represents imaginary border, after all testing trials were successful. After this border only few uncertainties appeared in the decision process. The local minimum in the episode 2 means that the robot was able to find the target in one or more trials, however it is more probable that it was a coincidence.

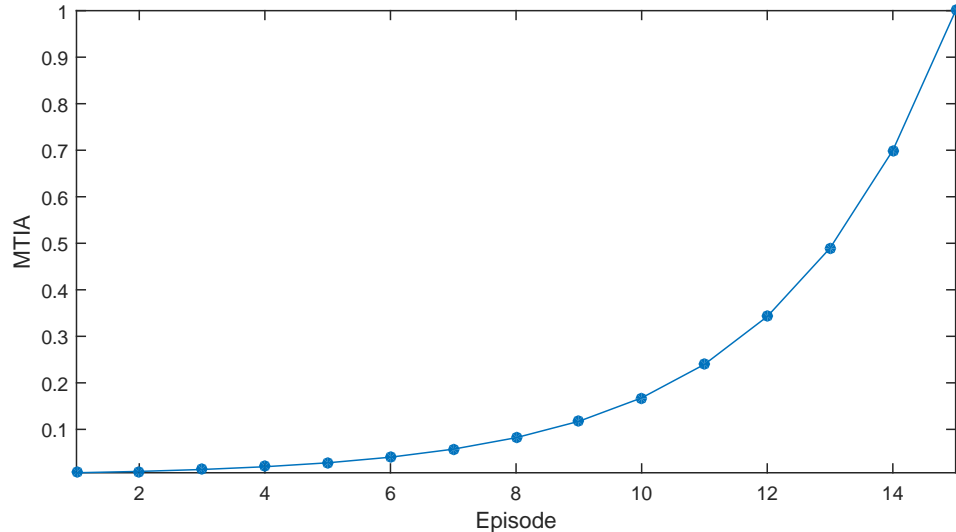


**Figure 6–2** Task learning process performance overview.

The second aspect of the learning process was regarded to increasing machine autonomy of the agent in the task. Figure 6–2 partially informs us about the status, so that after each episode robot can finish the task autonomously from different starting positions. After the sixth episode, the robot is fully autonomous in the task, and just some enhancements in the decision process are made. To increase the level of autonomy even during the learning process, I have implemented a mechanism that randomly leave an opportunity for the agent to decide. The number of agent’s decisions are also increased over the episodes so after some time, the agent can train the task absolutely independently.

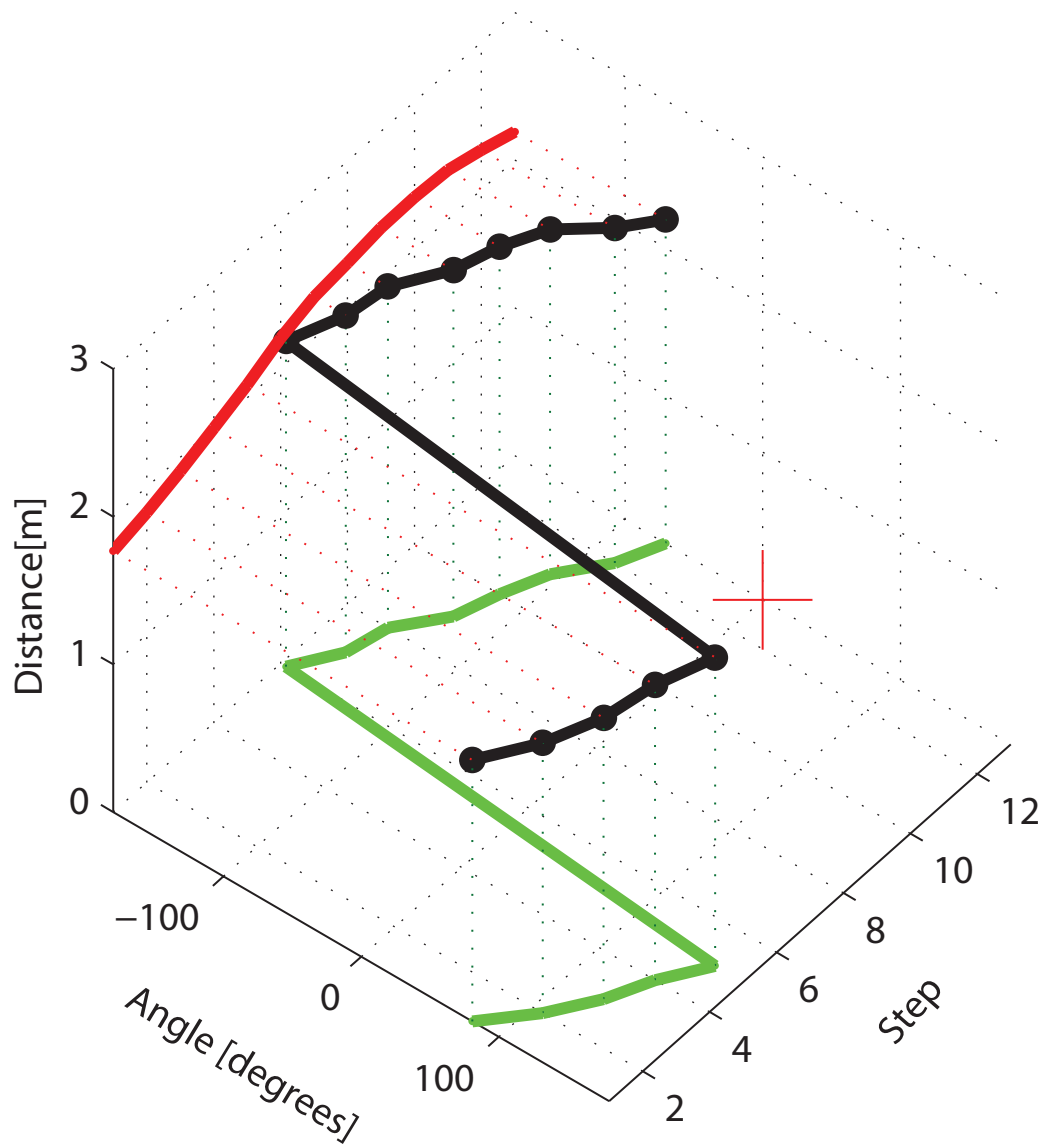
Figure 6–3 displays a dependency of an MTIA on the number of learning episodes passed. As mentioned in 2.5, low MTIA value corresponds to low agent’s autonomy and vice versa. We can also see that agent is capable of fully independent learning after episode 15.

In figures 6–4 and also 6–5 we can see robot’s performance during the task. Figures basically describe the movement of the robot in the space state, beginning in the same position. In the figure 6.4 we can see robot performing the task without any knowledge. As we can see, the robot is randomly wandering with low probability of approaching the target state, which is represented by a red cross. On the other hand, in figure 6–5, we can see that robot is doing very well when he can approach the goal state in 13 steps. For a better illustration of state variables development, I draw colored projections so that we can see how decision policy is converging.



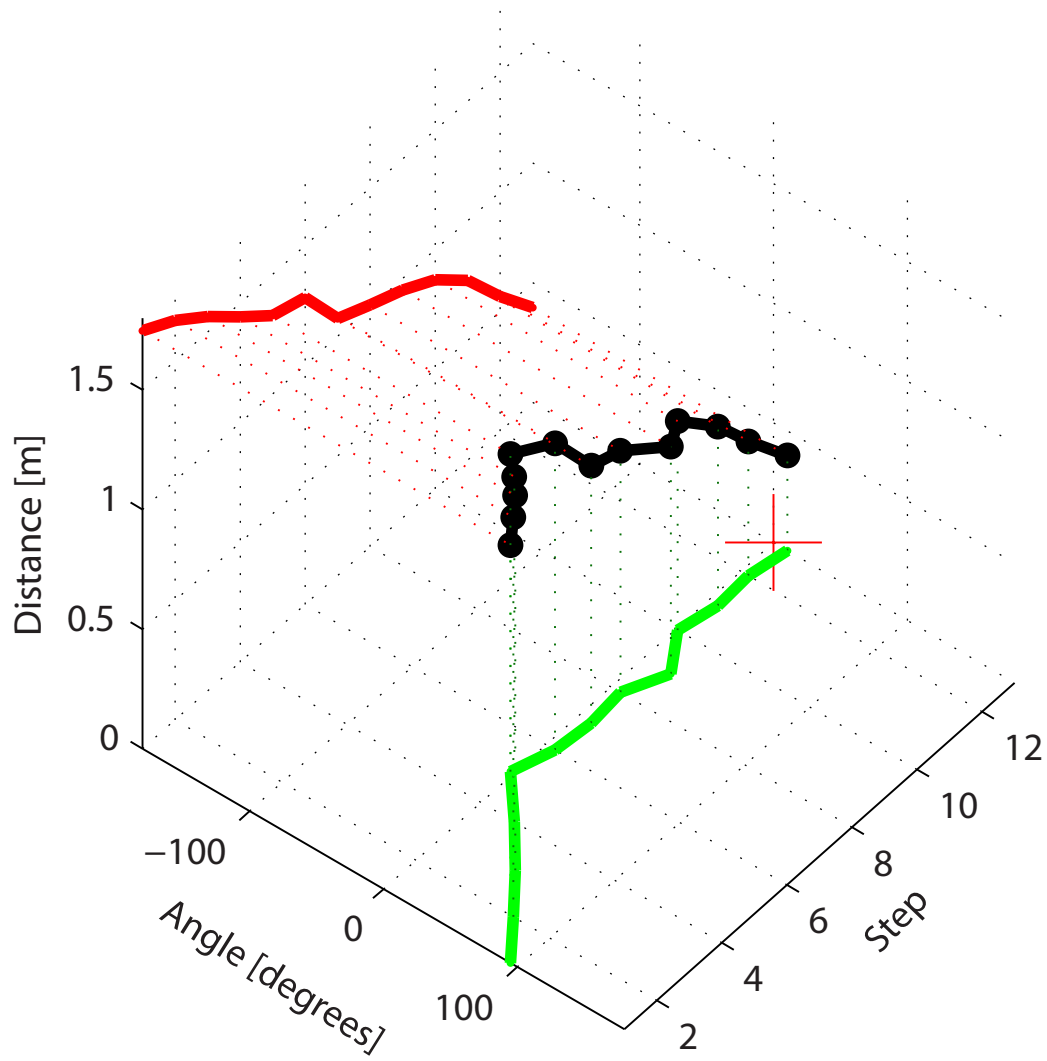
**Figure 6–3** MTIA level of an agent during learning process.

To finally evaluate the experiment, I can say that this experiment was successful in a meaning of decreasing the human operator’s involvement in performing specific tasks, but also in the learning process. This experiment is also significant from a practical point of view, due to its control simplicity. I experimentally verified, that



**Figure 6-4** State variables before learning process.

even an absolute layman can teach a robot to perform trivial task autonomously, without a knowledge of the programming language or robot's kinematics. This increases its potential in practical applications not only in the academic field but



**Figure 6–5** State variables after learning process.

also an industry.

## 7 Conclusion

I would like to summarize this thesis from the aspect of goals fulfillment as well as from the discussed topics.

According to the goals mentioned in 1 I have succeeded in fulfilling the following tasks:

1. I have reviewed the basic principles and also methods for teleoperation of robotic systems. Chapter 2 is focused mainly on the terminology and also application of the teleoperation. In addition, chapter 3 overviews some concepts of learning from a teleoperation applicable also in the area of humanoid robotics.
2. Chapter 4 discusses fundamental principles and algorithms of reinforcement learning applicable to the wide range of tasks. This overview also provides a necessary theoretical background for advanced methods described in the chapter 5. The chapter then focuses on the general scheme of the learning process in teleoperation with a utilization of a reinforcement learning techniques. The chapter also overviews some algorithms applicable to this problem and proposes an improvement of one of these methods, to make algorithm applicable to a humanoid robot in the real environment.
3. Chapter 6 then proposes and describes an experiment, which utilizes method proposed in chapter 5 and verifies its functionality in a simulated environment on a humanoid robot NAO. The task of the robot is to follow a ball according to the strategy learned from teleoperation.
4. The end of the chapter 6 finally discusses the implemented method and analyzes it from the aspect of incremental learning and also an involvement of an operator in the teleoperation process.

5. This task is satisfied by generating a complete code reference guide using a Doxygen API, which also contains a dependency diagram. The user guide and also a reference guide are enclosed to this work.

In addition to tasks described by the instructions, I have also added some own enhancements of this work. First of all, I gave a focus on the implementation of the proposed experiment and all its parts. I've given myself the challenge to create a minimalistic user-friendly application accessible from anywhere. For this purpose, I have developed a server application in Python and a client application in Javascript communicating over WebSocket for real-time responses and information update. Such an application is absolutely ready for getting up on the cloud and running as a web service. When talking about the minimalistic application, I also considered the learning time and performance of the application. Although Python is not the worst choice, for the purposes of neural network training I have implemented a super fast method called RPROP in pure C and written a C wrapper for it so it could be imported as a Python module in the server application. The build script also utilizes every possible optimization libraries, so the final performance is excellent. The whole system is described in detail in the appendix.

Not to only enhance the scientific aspect of the work, I have decided to compose also a scientific article that is enclosed in this work.



---

## References

- [1] Batsomboon, P., Tosunoglu, S. 1996. *A Review of Teleoperation and Telesensation systems* In: 1996 Florida Conference on Recent Advanced in Robotics, Florida Atlantic University, Florida.
- [2] Mebarak, E. and Tosunoglu, S. 2002. *On the Development of an Automated Design Interface for the Optimal Design of Robotic Systems* In: Proc. 5th World Automation Congress, WAC 2002 Pages: 9-13.
- [3] Zhu, M., Salcudean, S. E. 1995. *Achieving transparency for teleoperator systems under position and rate control* In: Intelligent Robots and Systems 95.'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on (Vol. 2, pp. 7-12). IEEE.
- [4] Pala, M., Sincak, P. 2012. *Towards the assisted teleoperation systems* In: Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), 2012 4th International Congress on IEEE. Pages: 490-495.
- [5] Hokayem, P. F., Spong, M. W. 2006. *Bilateral teleoperation: An historical survey* In: Automatica, 42(12), 2035-2057.
- [6] Chang, S., Kim, J., Kim, I., Borm, J. H., Lee, C., Park, J. O. 1999. *Kist teleoperation system for humanoid robot* In: Intelligent Robots and Systems, 1999. IROS'99. Proceedings. 1999 IEEE/RSJ International Conference on (Vol. 2, pp. 1198-1203). IEEE.
- [7] Sutton, R. S., Barto, A.G. 1998. *Reinforcement learning: An introduction* The MIT Press, Cambridge, MA, London, England, 1998. ISBN 0-262-19398-1.
- [8] Sinčák, P., Lorenčík, D., Virčíkova, M., Gamec, J. 2015. *Theoretical Analysis of Recent Changes and Expectations in Intelligent Robotics* In: Emergent Trends

- 
- in Robotics and Intelligent Systems. Springer International Publishing. Pages: 13-30.
- [9] Argall, B. D., Chernova, S., Veloso, M., Browning, B. 2009. *A survey of robot learning from demonstration* In: Robotics and autonomous systems, 57(5), Pages: 469-483
- [10] Ward, K., Zelinsky, A., McKerrow, P., Autumn, A., Ward, K., Zelinsky, E., Mckerrow, P. 2001. *Learning Robot Behaviours by Extracting Fuzzy Rules from Demonstrated Actions* In: MIPS, Vol6, No.
- [11] A. Howard, C.H. Park 2007. *Haptically Guided Teleoperation for Learning Manipulation Tasks* In: Robotics: Science and Systems: Workshop on Robot Manipulation, Atlanta, GA, 2007.
- [12] Smart, W. D., Kaelbling, L. P. 2002. *Effective reinforcement learning for mobile robots* In: Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on (Vol. 4, pp. 3404-3410). IEEE.
- [13] Berenji, H. R. 1992. *An architecture for designing fuzzy controllers using neural networks* In: Int. J. Approximate Reasoning, 6(2) Pages: 267-292
- [14] Berenji, H. R., Khedkar, P. 1992. *Learning and tuning fuzzy logic controllers through reinforcements* In: Neural Networks, IEEE Transactions on, 3(5) Pages: 724-740
- [15] Riedmiller, M. 2005. *Neural fitted Q iteration—first experiences with a data efficient neural reinforcement learning method* In: Machine Learning: ECML 2005 (pp. 317-328). Springer Berlin Heidelberg.
- [16] Riedmiller, M., Braun, H. 1992. *RPROP-A fast adaptive learning algorithm* In: Proc. of ISICIS VII), Universitat.
-

# Appendices

**Appendix A** User guide

**Appendix B** Reference guide

**Appendix C** CD