

*Intelligent Control Architecture For Motion  
Learning in Robotics Applications*

JAIME EDUARDO BELTRAN PARDO  
ELECTRONIC ENGINEER  
CODE: 300102



UNIVERSIDAD NACIONAL DE COLOMBIA  
FACULTAD DE INGENIERIA  
DEPARTAMENTO DE INGENIERIA DE SISTEMAS Y  
COMPUTACION

*Intelligent Control Architecture For Motion  
Learning in Robotics Applications*

JAIME EDUARDO BELTRAN PARDO

ELECTRONIC ENGINEER

CODE: 300102

FINAL PROJECT TO OPT FOR THE TITLE OF  
MASTER IN COMPUTER SCIENCE

ADVISOR

JONATAN GOMEZ PERDOMO, PH.D.

COMPUTER SCIENCE PH.D



UNIVERSIDAD NACIONAL DE COLOMBIA  
FACULTAD DE INGENIERIA  
DEPARTAMENTO DE INGENIERIA DE SISTEMAS Y  
COMPUTACION

**Title in English**

Intelligent Control Architecture For Motion Learning in Robotics Applications

**Título en español**

Arquitectura de Control Inteligente para el aprendizaje de movimiento en aplicaciones de Robotica

**Abstract:** The investigation of this Thesis was focused on how motion abilities can be learned by a robot. The main goal was to design and test a control architecture capable of learning how to properly move different simulated robots, through the use of Artificial Intelligence (AI) methods.

With this purpose, a simulation environment and a set of simulated robots were created in order to test the control architecture. The robots were constructed with a simple geometry using links and joints. A fuzzy controller was designed to control the motors position. The control architecture design was based on subsumption and some AI methods that allowed the simulated robot to find and learn a set of motions based on targets. These methods were a genetic algorithm (GA) and a set of artificial neural networks (ANN). The GA was used to find the adequate robot movements for an specific target, while the ANNs were used to learn and perform such movements efficiently. The advantage of this approach was that, no knowledge of the environment or robot model is needed. The robot learns how to move its own body in order to achieve a determined task. In addition, the learned motions can be used to achieve complex movement execution in a further research.

A set of experiments were performed in the simulator in order to show the performance of the control architecture in every one of its stages. The results showed that the proposed architecture was able to learn and perform basic movements of a robot independently of the environment or the robot defined structure.

**Resumen:** En esta Tesis, se investiga cómo las habilidades de movimiento en un robot, pueden ser aprendidas de forma automática. El objetivo principal fue diseñar y probar una arquitectura de control capaz de aprender a mover adecuadamente diferentes robots simulados, mediante el uso de métodos de Inteligencia Artificial (IA).

Con este propósito, se diseñó un entorno de simulación y un conjunto de robots simulados con el fin de probar la arquitectura de control. Los robots fueron construidos con una geometría muy simple utilizando enlaces y uniones (actuadores), y un controlador difuso fue diseñado para controlar la posición de los actuadores. El diseño de la arquitectura de control se basa en el concepto de subsunción (subsumption) y algunos métodos de IA que permiten al robot simulado determinar y aprender una serie de movimientos basados en objetivos. Los métodos usados son un algoritmo genético (GA) y un conjunto de redes neuronales artificiales (ANN). El GA se utiliza para encontrar los movimientos adecuados que el robot debe realizar para alcanzar un objetivo específico, mientras que las redes neuronales se utilizan para aprender y realizar estos movimientos de forma eficiente. La ventaja de este enfoque es que, no es necesario conocer el entorno o tener un modelo del robot, si

no que el robot aprende cómo mover su propio cuerpo en un ambiente definido con el fin de lograr una tarea determinada. Además, en una posterior investigación, es posible utilizar los movimientos aprendidos para realizar movimientos o tareas más complejas con los robots.

Un conjunto de experimentos se llevaron a cabo en el simulador para mostrar el desempeño de la arquitectura de control en cada una de sus etapas. Los resultados muestran que la arquitectura propuesta es capaz de aprender y realizar los movimientos del robot independientemente del medio ambiente o la estructura definida del robot.

**Keywords:** robot, platform, hardware, architecture, control, artificial intelligence, learning, fuzzy, genetic algorithm, neural network

**Palabras clave:** robot, plataforma, hardware, arquitectura, control, inteligencia artificial, aprendizaje, difuso, red neural, algoritmo genetico

# Acceptation Note

Thesis Work

Aprobado

“ mention”

---

Jury  
Jorge Sofrony

---

Jury  
Fabio Gonzalez

---

Advisor  
Jonatan Gomez

Bogota, D.C., Agosto 30 de 2013

---

## Acknowledgements



I want thank all the people whose insistence, guidance, support and advice have sustained me during the development of this work. Thanks, to my advisor Jonatan Gomez, for his support and interesting ideas which highly influenced this thesis. I also would like to thank my colleagues from the research group ALife, for giving me their friendship, advices and support.

Finally I wish to thank my family who have always supported me and encouraged me to achieve my goals.

# Contents



<b>Contents</b>	<b>I</b>
<b>List of Tables</b>	<b>IV</b>
<b>List of Figures</b>	<b>V</b>
<b>Introduction</b>	<b>VIII</b>
<b>1. Background - Autonomous Robots</b>	<b>1</b>
1.1 Robot manipulators . . . . .	3
1.2 Kinematics . . . . .	3
1.2.1 Forward Kinematics . . . . .	4
1.2.2 Inverse Kinematics . . . . .	4
1.3 Control . . . . .	5
1.4 Control architectures . . . . .	6
1.4.1 Classic robot control . . . . .	6
1.4.2 Reactive control . . . . .	7
1.4.3 Hybrid Control . . . . .	7
1.4.4 Behavioural-Based control . . . . .	8
1.5 Artificial intelligence . . . . .	9
1.5.1 Genetic Algorithms . . . . .	10
1.5.2 Neural Networks . . . . .	11
1.5.3 Fuzzy logic . . . . .	16
1.5.3.1 Rule-based Fuzzy Systems . . . . .	18

---

1.5.3.2	Singleton Fuzzy System . . . . .	18
1.6	Intelligent control . . . . .	20
1.6.1	Control using genetic algorithms . . . . .	20
1.6.2	Fuzzy logic controllers . . . . .	21
1.6.3	Evolutionary Robotics . . . . .	22
1.7	Motion learning and intelligent control approaches . . . . .	24
1.7.1	Automated Learning of Muscle-Actuated Locomotion Through Control Abstraction . . . . .	25
1.7.2	Spontaneous Evolution of Structural Modularity in Robot Neural Network Controllers . . . . .	27
1.8	Summary . . . . .	30
<b>2.</b>	<b>Robot representation and simulation</b>	<b>32</b>
2.1	Simulator architecture and settings . . . . .	33
2.2	Simulated robot specifications . . . . .	34
2.2.1	Robot arm . . . . .	35
2.2.1.1	Robot arm targets . . . . .	35
2.2.1.2	Robots arm sensors . . . . .	36
2.2.2	Motor model and Controller design . . . . .	37
2.2.2.1	Motor model . . . . .	37
2.2.2.2	Fuzzy controller . . . . .	38
2.2.2.3	Fuzzy controller simulation . . . . .	39
2.3	Graphical interface . . . . .	41
2.4	Summary . . . . .	43
<b>3.</b>	<b>Motion Learning</b>	<b>44</b>
3.1	A subsumption architecture approach . . . . .	45
3.1.1	Basic movement learning architecture . . . . .	47
3.2	Finding robot's movements . . . . .	49
3.2.1	Genetic algorithms as a search method . . . . .	49
3.2.2	Genotype and Phenotype . . . . .	50
3.2.3	Fitness function . . . . .	51
3.2.4	Termination Condition . . . . .	53
3.2.5	Testing HAEA performance . . . . .	54



---

3.3	Learning robot's movements . . . . .	59
3.3.1	Neural Networks for instruction approximation . . . . .	60
3.3.2	Neural networks performance . . . . .	63
3.3.3	Improving Neural Network Structure . . . . .	65
3.3.4	Improving the learning process . . . . .	67
3.4	Result analysis . . . . .	71
3.5	Summary . . . . .	73
	<b>Conclusions</b>	<b>76</b>
	<b>Future work</b>	<b>80</b>
	<b>Bibliography</b>	<b>82</b>

## List of Tables



1.1	HaEA Algorithm[20]. . . . .	12
2.1	Rule base . . . . .	39
3.1	Control Architecture Algorithm. . . . .	49
3.2	Genotype and phenotype of 3 DOF robot arm instruction. . . . .	51
3.3	Environment and robot simulation parameters values for 3DOF robot arm. . . . .	54
3.4	Average of 30 runs of fitness, number of generations and final distance for 2 random targets in the simulation of 3DOF robot arm. . . . .	57
3.5	Average over 30 runs of total trained samples using 100 random targets in the 3DOF robot arm . . . . .	64
3.6	HAEA and NNs parameters values. . . . .	64
3.7	Average over 30 runs of 200 random targets for 2 DOF, 3 DOF and 4 DOF robot arms. Reported values are de medians of distance and fitness obtained of the targets . . . . .	65
3.8	Average over 30 runs of 200 random targets for 2 DOF, 3 DOF and 4 DOF robot arms. Reported values are the medians of distance and fitness obtained of the targets after training with 60 samples (targets). . . . .	67
3.9	Improved algorithm of the control architecture with feedback path of NN response for initializing HAEA population . . . . .	69
3.10	Average over 30 runs of 200 random targets for 2 DOF, 3 DOF and 4 DOF robot arms. Reported values are de medians of distance $DT_m$ and fitness $f_m$ . . . . .	70
3.11	Threshold adjustment process for the 2DOF robot arm. Images are arranged from left to right and from top to bottom . . . . .	75

# List of Figures



1	Comparison between industrial robots and a human entertainment robot . . . . .	IX
2	Subsumption architecture . . . . .	XI
1.1	A typical robot arm or manipulator . . . . .	2
1.2	Forward and Inverse Kinematics . . . . .	4
1.3	Block Diagram of a Feedback Controller . . . . .	6
1.4	SPA architecture . . . . .	7
1.5	Subsumption architecture diagram . . . . .	9
1.6	Neuron model expressed as a weighted sum of inputs and activation function. . . . .	13
1.7	Examples of activation functions . . . . .	13
1.8	Feed-forward neural networks . . . . .	14
1.9	Comparison between classical logic and fuzzy logic membership functions . . . . .	17
1.10	Singleton fuzzy set . . . . .	19
1.11	Non-supervisory fuzzy control diagram . . . . .	22
1.12	Supervisory fuzzy control diagram . . . . .	23
1.13	Biomechanical model of a snake. “Model consists of nodal masses (points) and springs (lines). It has twenty independent actuators (muscle springs): ten on the left side of the body and ten on the right side.” Taken from [22] . . . . .	25
1.14	Objective Function. Taken from [22] . . . . .	26

---

1.15	Simulated 2D robot. There are four different environments where the robot is tested. Difference relies on where is the circumference (left or right) and what is its size (small or large). Taken from [8] . . . . .	28
2.1	Simulator architecture . . . . .	33
2.2	Simulated robot examples . . . . .	35
2.3	Structure of simulated robots . . . . .	36
2.4	Robot sensors measures and limits . . . . .	37
2.5	Fuzzy controller diagram . . . . .	39
2.6	Input variable error $e$ membership function . . . . .	39
2.7	Input variable error change $de$ membership functions . . . . .	40
2.8	Output control variable . . . . .	40
2.9	Fuzzy controller step response . . . . .	41
2.10	Simulator environment . . . . .	42
3.1	Subsumption architecture for motion control. Arrows indicate the hierarchic flow of control . . . . .	46
3.2	Subsumption architecture for motion control . . . . .	47
3.3	Detailed model of the architecture for a single controller/actuator . . . . .	48
3.4	Genetic algorithm model . . . . .	50
3.5	3 DOF Robot Arm pareto frontier of fitness function. Total of evaluated points = 512 . . . . .	53
3.6	Fitness mean of population ( $P = 20$ ) over 100 generations of a single random target . . . . .	55
3.7	Fitness mean and variance of HAEA over 30 generations for 4 different targets . . . . .	56
3.8	Fitness mean and variance over 30 runs of the best individual for 30 random generated targets. Simulation performed for the 2DOF, 3DOF and 4DOF robot arms . . . . .	58
3.9	Distance mean and variance over 30 runs of the best individual for 30 random generated targets. Simulation performed for the 2DOF, 3DOF and 4DOF robot arms . . . . .	59
3.10	Generations mean and variance over 30 runs of the best individual for 30 random generated targets. Simulation performed for the 2DOF, 3DOF and 4DOF robot arms . . . . .	60
3.11	MP Neural Network with radial activation function and linear output. Inputs: distance and angle to the target, Output: angle of movement of the motor . . . . .	61

---

3.12	Neural network structure with actuator movement dependence . . . . .	66
3.13	Threshold measure . . . . .	68
3.14	Threshold value over 80 targets for the 2 DOF robot arm . . . . .	70
3.15	Threshold value over 120 targets for the 3 DOF robot arm . . . . .	71
3.16	Threshold value over 150 targets for the 4 DOF robot arm . . . . .	72

---

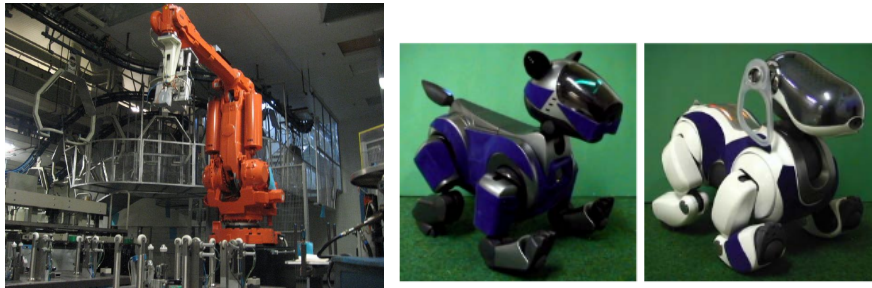
## Introduction



Robotics is an area that has grown very fast in recent times. Robots are becoming more sophisticated and autonomous machines as we want them to explore new environments and achieve new tasks. Although many advances have been achieved, the same interest has remained over the years, how to move a robot properly in real environments?

Early robots were nothing but pre-programmed machines running repetitive tasks in specific routines - basically pick and place operations in industrial environments [34] (see Figure 1(a)). In these robots the locomotion problem relied mainly on the mechanical and control design of the robot. Nowadays, robots are being moved from the structured industrial environment to the unpredictable human environment. Here, more than perform repetitive tasks, robots need to adjust their behaviors autonomously in the ever-changing physical environment (see Figure 1(b)). The design of these new *autonomous robots* involves not only the mechanical and control design, but also some artificial intelligence and sensor management modules. This will provide the robot with the ability to adapt to environmental changes. Sensing, motion control, and task planning are some of the major problems addressed by researchers when creating autonomous robots [32].

Autonomous robot design strategies have certainly achieved successful motions under the appropriate conditions for specific tasks and robot designs [7, 12, 14, 41,



(a) Industrial robot arm. Robot arm programmed to assembly TV parts in a Factory. (b) Robot Sony AIBO. Entertainment robot, equipped with multiple sensors to avoid obstacles, recognize images and interact with humans.

FIGURE 1. Comparison between industrial robots and a human entertainment robot

36, 21, 15]. Furthermore, artificial intelligence methods have been combined into the control strategies of autonomous robots in order to provide them with the ability to adapt to environmental or objective changes[45, 5, 13]. Nonetheless most of these approaches still require a specific mechanical and control design for a particular robot structure.

The robot mechanical and control design requires an analysis of kinematics, dynamics, and control theory which depending on the robot design can be very complex. These fields of knowledge have been well developed during robotics field evolution and there are a lot of methods based on these concepts to design and analyze robots[52]. Nevertheless, since those methods are based on the mathematical analysis of the robot structure, when the application or the structure in mention is changed, the robot mechanical analysis must be redone. A new set of control methods must be created in order to allow the robot to meet its new goal or to move its modified mechanical components. In other words, re-using an autonomous robot for another task with different structure can almost be as hard as building a new one from scratch.

In nature, animals and humans are able to move properly in several environments without specific knowledge about their own bodies. They do not need complex mathematical models to perform movements in a given environment, but they learn

---

how to properly move for a given purpose. In this sense, is it possible to create a robot capable of learning to move similarly?

Evolutionary Robotics (ER) is a new field where robots learn to move similarly to how we learn in nature (trial and error). Authors of this field use evolutionary strategies to create novel control methods that allow a given robot to move for a specific purpose in determined environments[27, 8, 17].

Grzeszczuk [22], proposed a similar approach using simulated annealing instead of evolutionary strategies. In his work, simulated animals use automated learning methods to find a suitable set of controllers that allowed them to move for a given purpose. In the words of the author, “It is as if the animal had a fully functional body, but no motor control center in its brain”. It will learn motor control by interacting with its environment.

In this thesis, these nature inspired ideas are applied to robotics. An ideal scenario would be to build an autonomous robot architecture capable of learning how to move a given robot properly in an environment without exact information about the mechanical design. However, overcoming this challenging problem involves not only solving the robot motion problem for multiple environments with different goals, but also recognizing and operating multiple sensors and actuators of different robots in a single control architecture.

This thesis addresses part of this problem by proposing a control architecture based on the concepts of subsumption architecture [10] (see Figure 2). In this architecture there are three different levels of control for the robots: level 0, level 1 and level 2. From the lower to the higher level, each one handles a specific function or behavior of the robot. Level 0 is low level control, level 1 is basic movement learning, and level 2 is complex movement learning.

The idea is that all levels of the control system must be automatically learned providing the robot with the ability to adapt to its own structure, the environmental conditions, and its purpose. However, this is complex. The design of learning methods for each one of the levels are currently wide research fields and combining



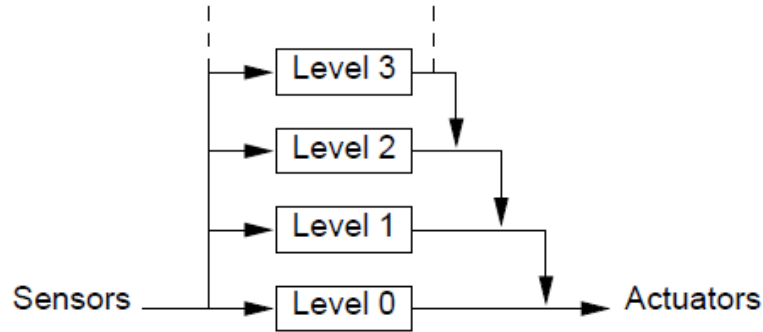


FIGURE 2. Subsumption architecture

those methods into a single architecture involves a bigger challenge. This project is mainly focused on a learning mechanism that allows a robot to learn a coordinated motion based on a specific task. Such a learning mechanism is developed only in level 1 of the architecture. In this sense, level 0 is designed to ensure the movement completion of the actuators by setting an appropriate set of static parameters in a fuzzy controller. Level 2 of the architecture is left unaddressed and open for further research.

Level 1 of the architecture is composed of two main parts: a genetic algorithm (GA) and a set of neural networks (NNs). The genetic algorithm is used to find the adequate movements of the robot for several targets in a specific environment and the set of NN is trained in order to learn the movements found by the GA. Each NN of the set controls one of the movable parts of the robot and acts as a perception-reaction unit, providing a fast response to the control system. To test the performance of the proposed method, three different simulated robots were designed in a 2D robot simulator. Finally, several experiments were performed in order to test different stages of the learning process. Results showed that with the proposed architecture the robot is able to learn appropriate movements to accomplish a specific goal in a defined environment.

---

## Thesis outline

This thesis is organized as follows:

- **Chapter 1. Literature Review**

In this chapter, a brief literature review with basic concepts about motion control is presented. Topics such as robot design, control architectures, fuzzy logic, genetic algorithms, neural networks, controllers and computation learning methods (with a special focus on motion learning) are discussed. Some interesting motion learning approaches in simulation or physical robots are presented as well.

- **Chapter 2. Robot Representation and Simulation**

In this chapter, the robot simulator is shown and the simulation environment and models, used to represent a robot, are described. Also some basic robot structures are defined.

- **Chapter 3. Motion Learning**

In this chapter the control architecture is designed. Subsumption architecture scheme is used to combine a Genetic algorithm and a set of Neural Networks. The genetic algorithm HAEA is used to find the movements of a robot for an specific set of objectives and the set of Neural networks are used in order to learn the movements found by the GA. Several experiments are performed in order to test the proposed control architecture.

- **Chapter 4. Conclusions and Future Work**

Finally, conclusions about this work are presented, as well as future work and possible applications.

---

# Chapter 1

---

## Background - Autonomous Robots

Autonomous robots are machines capable of carrying out a complex series of actions automatically. They are capable of sensing, take decisions, move and act in an environment. Also, they must be capable of persisting on time and adapting to changes, as well as to achieve some goals [46]. However, their building is not as easy task. It involves knowledge in several areas such as mechanics, kinematics, dynamics, artificial intelligence, electronics and control theory.

The standard approach of designing a robot involves a kinematic and dynamic analysis in order to define the appropriate controllers and methods for moving the robot. Depending on the robot construction, this can be one of the hardest and time consuming tasks in robotics. In a typical robot arm or manipulator (see Figure 1.1) the kinematic analysis is fundamental for moving the robot to the desired position. Such analysis allow to find the exact angles for each actuator of the robot in order to achieve some position (like reach a point or grab a object). However the analysis is performed for fixed robot structures and when the structure (i.e the dimensions or the components distribution) of the robot are changed, a new analysis must be performed and new controllers must be designed. This process is almost as hard as a building a new robot.

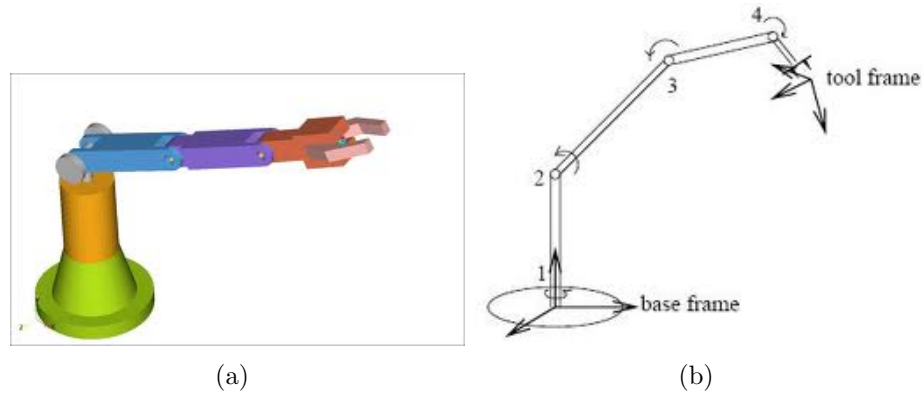


FIGURE 1.1. A typical robot arm or manipulator

The problem relies in the need of an exact knowledge about the robot construction in order to perform the dynamical and kinematic analysis. On the contrary, in nature, animals and humans do not have exact knowledge about their own bodies. Instead they learn how to properly move through a trial and error procedure. This point of view have been addressed from the Artificial intelligence field and applied in multiple areas as well as in robotics. This way, automatic learning methods can be used into the control mechanisms of the robot in order to provide it with the ability to learn movements or optimize its already learned movements. The advantage is that there is no need of exact knowledge about the robot construction and hence, if the robot structure change, no redesign process is needed.

Giving the complexity of a robot and the infinite applications of robotics, there are a lot theory and robot designings suitable for such kind of movement learning. Based on this, in this chapter, a quick review of the basic topics on robotics is performed. Then, some of the theory of artificial intelligence is presented and how it is used in the robotics field. Finally, some approaches of movement learning are presented.

## 1.1 Robot manipulators

A typical robot is a robot arm or manipulator. This kind of robot are used in industry widely to perform repetitive tasks such painting, assembling or soldering. A robot arm consists of the parts: base, joints, links, and a end effector. The joints are where the motion in the arms occurs, while the links are of fixed construction. Joints are flexible and joins two separated links. The base is the basic part over the arm which may be fix or active. The last part is the end effector, which can have different forms depending on the application. A typical end effector is a grapper which is used to hold and move objects.

Joints may be actuated by motors or hydraulic actuators, and there are several types of joints that can be used on a manipulator. One of the most used are revolute joints that allows rotary motion about an axis of rotation (e.g the human elbow). Figure 1.1 shows a typical robot manipulator.

## 1.2 Kinematics

Robot kinematics is the study of the motion(kinematics) of robots. In a kinematic analysis the position, velocity and acceleration of all the links are calculated without considering the forces that cause this motion.

Robot kinematics deals with aspects of redundancy, collision avoidance and singularity avoidance. The analysis is performed by assigning a frame of reference to each movable part of the robot (joints) and hence a robot with many parts may have many individual frames.

Homogeneous transformation is used to solve kinematic problems. This transformation specifies the location (position and orientation) of the hand in space with respect to a defined reference frame, but it does not tell us which configuration of the arm is required to achieve this location. It is often possible to achieve the same hand

position with many arm configurations. In this sense, robot kinematics analysis are mainly of the following two types: forward kinematics and inverse kinematics.

In forward kinematics, the length of each link and the angle of each joint is given and we have to calculate the position of any point in the work volume of the robot. In inverse kinematics, the length of each link and position of the point in work volume is given and we have to calculate the angle of each joint (see Figure 1.2).

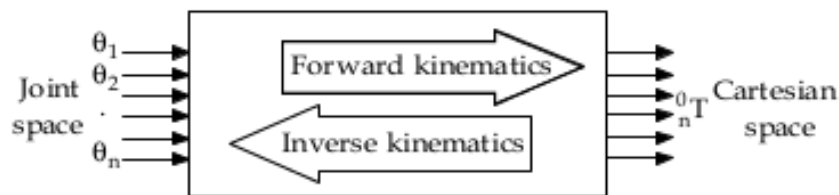


FIGURE 1.2. Forward and Inverse Kinematics

### 1.2.1 Forward Kinematics

Forward kinematics is the method for determining the orientation and position of the end effector, given the joint angles and link lengths of the robot arm. The forward position kinematics (FPK) solves the following problem: "Given the joint positions, what is the corresponding end effector's pose?". Forward kinematics problem is straightforward and there is no complexity deriving the equations. Hence, there is always a forward kinematics solution of a manipulator.

### 1.2.2 Inverse Kinematics

Inverse kinematics is the opposite of forward kinematics. This is when you have a desired end effector position, but need to know the joint angles required to achieve it. The inverse position kinematics (IPK) solves the following problem: "Given the actual end effector pose, what are the corresponding joint positions?". In contrast to the forward problem, the solution of the inverse problem is not always unique:

the same end effector pose can be reached in several configurations, corresponding to distinct joint corresponding position vectors. Although way more useful than forward kinematics, this calculation is much more complicated too.

The solution of the inverse kinematics problem is computationally expansive and generally takes a very long time in the real time control of manipulators. Singularities and nonlinearities that make the problem more difficult to solve. Hence, only for a very small class of kinematically simple manipulators (manipulators with Euler wrist) have complete analytical solutions.

The problems in IK are:

- There may be multiple solutions,
- For some situations, no solutions,
- Redundancy problem.

There are several methods to solve the inverse kinematics like the analytic method or the inverse Jacobian Method[52].

## 1.3 Control

Control systems are mechanisms whose main purpose is to return a specific output for some input or set of inputs. Robots have a huge quantity of control systems to manage different components such as actuators, sensors, signal processing units, communication and even decision making mechanisms. All these systems receive inputs of different type, process or transform them and then return an output understandable for the next system or the environment itself. In control theory, motion or actuation control systems are called controllers.

Controllers provide a way to obtain precise control of the actuators. In robotics, it is usual to use a controller for every actuator in order to get accurate movements. Controllers take a reference and move the actuator through an electrical or

mechanical process in order to get the desired movement. Negative feedback is often used to improve controllers performance (see Figure 1.3). Also, digital control, unlike analogue control, offers many advantages such as flexibility, precision and re-programmable characteristics [48]. Nevertheless, analogue controllers may have faster responses.

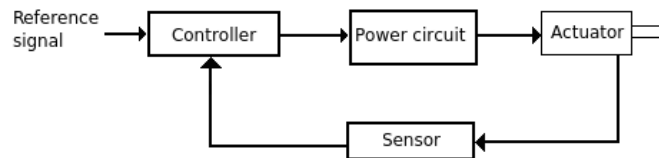


FIGURE 1.3. Block Diagram of a Feedback Controller

## 1.4 Control architectures

Control architectures are programs that control a robot or agent. Control architectures define the way how the sensing, reasoning and acting processes are performed. Basically, they are models in which all the systems in the robot are combined. Even though, there are too many possible control architecture models, fundamental approaches can be classified in a small set. The difference relies on how deliberative or reactive they are [44].

### 1.4.1 Classic robot control

Classic robot control, which was the base of early robots, corresponds to purely deliberative control architectures. In this approach, models of the world or the robot itself are the key to plan strategies, make decisions and take the appropriate actions. Essentially, the problem is decomposed in a series of functional units (see Figure 1.4). Information is acquired from the environment using the sensors and then it is used to build a model of the world. The planning module use this model to decide the actions of the actuators and execute them. This classic robot control approach is also called sense-plan-act (SPA) architecture accordingly.





FIGURE 1.4. SPA architecture

This kind of architectures are convenient for very well known environments and static conditions. For very dynamic or complex changing environments, the lack of information results in erroneous models and therefore malfunction of the robot. Nevertheless, the use of computational intelligence tools, allow robots to build complex models automatically from the information obtained by their sensors. Interesting approaches that use these methods are movement learning by demonstration [5] and motor skills learning [45]. Also, Nguyen-Tuong [40] makes an extensive review on methods for automatically generating models and how to adapt them to changes of the environment.

### 1.4.2 Reactive control

Reactive approaches have no internal-model or perform a search for optimal path. Basically they are mapping functions between sensors and actions in order to rapidly respond to the environment feedback [36]. Reactive control is optimized to achieve real-time performances. However, due to its simplicity, it is incapable of performing complex tasks in which planning is required.

### 1.4.3 Hybrid Control

Hybrid approaches attempt to combine reactive and deliberative approaches in a single control architecture. Reactive components are used to obtain a fast response to robots immediate needs, such as avoiding obstacles, processing input information or balancing the robot. Conversely, deliberative components plan strategies in longer time-scale and take actions based on the internal world-model. The big challenge

lies on the interaction between the two components i.e., how to combine them in a single architecture [44].

Nowadays, hybrid control is one of the most used architectures for robot design; interesting approaches are *Robotic Software Architecture for Multisensor Fusion System* [18] and *On three-layer architectures* [19]. Also, *Medeiros* [38] presents a survey in control architectures for autonomous robots.

#### 1.4.4 Behavioural-Based control

“Behavioural-based control is inspired in biology, and tries to model how animals deal with their environment” [44]. What is known so far is that animals do not have detailed models of the environment. Instead, a set of combined behaviours with some information is enough for them to behave efficiently in their environment. This suggests that it is not necessary to have a huge amount of information to build models for robot control, but some specific behaviours must be enough to act properly in different environments [36]. In behavior-based robotics, basic robotic behaviors are elementary building blocks for robot control, reasoning, and learning. The environment plays a central role in activating a certain basic robotic behavior throughout the execution of robotic systems.

Behavior-based design is a commonly employed approach in building autonomous robot navigation systems. The robot must autonomously coordinate behaviors in order to exhibit the complicated robotic behavior, e.g., coordinating target-tracking and anti-collision behaviors in order to reach a target position while avoiding unforeseen and moving obstacles during its motion [34].

Subsumption architecture, defined by R. Brooks [10] is one of the most prominent behavioral-based methods. This architecture is divided by layers which are defined in different levels (see Figure 1.5). Each layer is an action-reaction unit often called “behaviour” and is designed to respond in a particular and autonomous way. Lower layers are responsible for simple and vital behaviours, while higher layers manage more complex processes. A specific behaviour is usually associated to some

tasks, such as object avoidance, target acquisition and line following. The chain of command of subsumption architecture is strictly up-bottom, in which higher layers are allowed to influence or override (subsume) the outputs of lower layers but not backwards. In a typical robot, most layers receive inputs from sensors, while only the lower layers directly influence actuators [51].

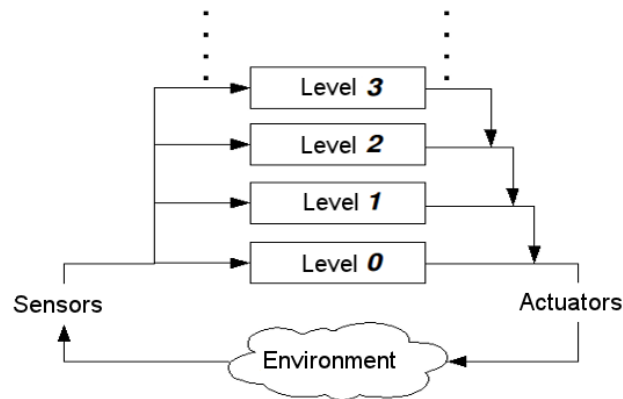


FIGURE 1.5. Subsumption architecture diagram

Subsumption architecture layers are easier to build and run faster than a scheme of serial functional units. Furthermore, expansion of the system may be done in a simple way by adding more layers or behaviours to the structure [11]. The challenge of this structure lies in finding effective mechanisms for coordination between layers or behaviours [44, 36]. The control system must decide which behaviour fit best for a giving condition. This is known as behaviour coordination problem or action selection problem. Approaches to this problem use some of the intelligent systems methods, such as Voting, fuzzy systems [6], reinforcement learning [46], among others.

## 1.5 Artificial intelligence

Artificial intelligence (AI) has been used in many fields in order to optimize processes, classify data, find solutions, make decisions and solve problems when there is lack of information. Neural networks (NN), genetic algorithms (GA), fuzzy logic and reinforcement learning (LR) are some of the most known methods of AI. Most

of these methods are biology inspired and have shown very good results for problem solving in different fields.

### 1.5.1 Genetic Algorithms

Genetic algorithms (GA) are a very useful technique for multiple-objective optimization problems and exploratory search. These methods are biology inspired and try to emulate the natural evolution processes of biological organisms. Basic concepts on these methods were first developed by Holland [26], but now further studies have improve GA concepts, definitions and applications. According to Mitchell [39], a GA has the following characteristics:

1. There is a population of possible solutions for an specific problem, in which each solution is coded according to a chosen representation (e.g. a string of bits that represent the spacial position on a circuit). The possible coded solutions are known as chromosomes and codification units (e.g bits on the bit string) are known as genes.
2. An objective function known as *fitness function* that gives a numeric value to each chromosome of the population, in order to evaluate its performance as a solution.
3. A set of genetic operators to be applied to chromosomes in order to create new populations. This process generally includes; selection, in which chromosomes are chosen according to its fitness to provide offspring; crossover, in which a pair of chromosomes recombine their genes to produce one or more new chromosomes; and mutation, in which one or more genes of the chromosome are modified in a random way.

Taking into account that possible solutions coded in the population are modified in a stochastic form, GA are knowledge domain independent, since only one fitness function is needed. Due to this stochastic nature, GA always allows finding a solution that may or may not be an optimal solution [35]. Given this, GA have

been applied to different problems like dynamic, multi-modal, multi-objective and restricted optimization, and applications are found in biology, bio-informatics, data mining, games, arts, music and others [43].

Different GA have surged for different problems with some advantages and disadvantages . One of them is HAEA.

### **Hybrid Adaptive Evolutionary Algorithm (HaEa)**

The HaEa algorithm proposed by Gomez in [20] is shown in Table 1.1. It combines some of the ideas of Evolutionary Strategies (ES), decentralized control adaptation and central control adaptation. Basically, the algorithm adapts some operator probabilities (rates) while evolving the solution of the problem. This way, each individual encodes its genetic rates and in every generation, individuals are modified by only one operator that is selected according to the encoded rates. Such rates are updated according to the performance achieved by the offspring (compared to its parent) and a random learning rate.

This algorithm has the advantage of selecting the better performance operators from a set of genetic operators. As the author states “*HaEa has temporal learning of operator rates. This adaptive property allows HaEa to have a higher chance to locate an optimal solution than another evolutionary algorithm*”. Also, compared to other algorithms, HaEa is capable of finding good solutions, as well as the most known genetic algorithms like the regular Genetic Algorithm (GA), Generational Genetic Algorithm (GGA) and Steady State Genetic Algorithm (SSGA).

### **1.5.2 Neural Networks**

Artificial Neural Networks (ANNs) are biology inspired computational tools which try to emulate the function of the brain in biological organisms. Brains are formed by neurons (cells in the brain) whose principal function is the collection, processing, and dissemination of electrical signals. When such neurons are connected in an network, they are thought to produce the brain ability of information-processing.

TABLE 1.1. HaEA Algorithm[20].

**Hybrid Adaptive Evolutionary Algorithm (HaEa)**HaEA(  $\lambda$ , terminationCondition )

1.  $t_0 = 0$
2.  $P_0 = \text{initPopulation}(\lambda)$ ,
3. **while** (terminationCondition( $t, P_t$ ) is false) **do**
4.  $P_{t+1} = \{\}$
5. **for** each ind  $\in P_t$  **do**
6. rates = extract\_rates(ind)
7.  $\delta = \text{random}(0,1)$  *learning rate*
8. oper = Op\_Select(operators, rates)
9. parents = ParentsSelection( $P_t$ , ind)
10. offspring = apply(oper, parents)
11. child = Best(offspring, ind)
12. **if**( fitness( child ) < fitness( ind ))**then**
13. rates[oper] =  $(1.0 + \delta) * \text{rates}[\text{oper}]$
14. **else**
15. rates[oper] =  $(1.0 - \delta) * \text{rates}[\text{oper}]$
16. normalize\_rates(rates)
17. set\_rates(child, rates)
18.  $P_{t+1} = P_{t+1} \cup \text{child}$
19.  $t = t + 1$

For this reason, some of the earliest AI work aimed at creating artificial neural networks (ANN) [46]. The mathematical model of a neuron was first developed by McCulloch and Pitts (1943) [37]; however more detailed and realistic models have been developed in the AI field since then.

ANNs are composed by a set of distributed units (called neurons) that compute information. These neurons are connected to other neurons by links that define a network topology. A link from neuron  $j$  to neuron  $i$  serves to propagate the activation from  $j$  to  $i$ . Each neuron  $j$  also has a numeric weight  $W_j$  associated with it, which determines strength and sign of the the connection [46]. Essentially, a neuron computes a weighted sum of its inputs and then applies some activation function in order to derive the output (see Figure 1.6).

$$a_i = g(in_i) = g\left(\sum_{j=0}^n W_{j,i} a_j\right) \quad (1.1)$$

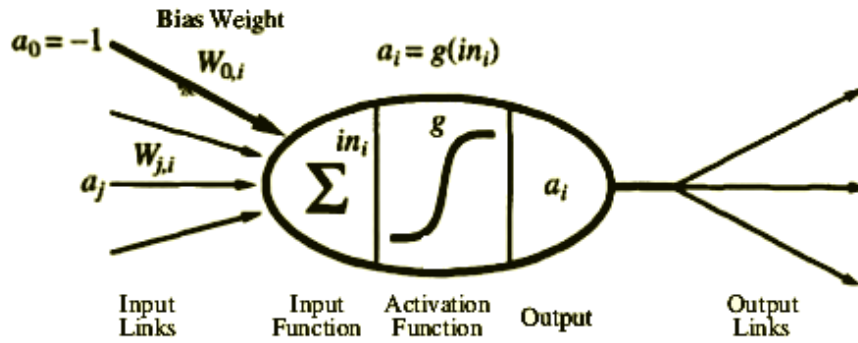
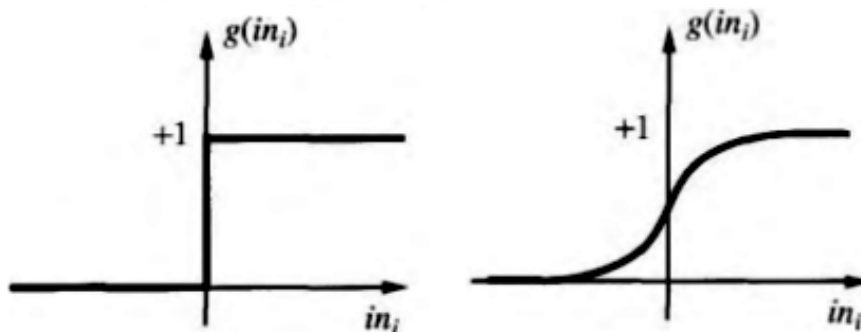


FIGURE 1.6. Neuron model expressed as a weighted sum of inputs and activation function.

Several activation functions can be used in neural networks which confer specific properties to them. Generally, those functions must meet two conditions: First, providing an “active” output (near +1) when the “right” inputs are given, and an “inactive” output (near 0) when the “wrong” inputs are given. Second, the activation needs to be nonlinear, “otherwise the entire neural network collapses into a simple linear function” [46]. Two common examples are the threshold function and sigmoid function (see Figure 1.7). The sigmoid function has the advantage of being differentiable, which is an important property for some learning algorithms.



(a) Threshold activation function. (b) Sigmoid activation function  $1/(1 + e^{-x})$ .  
 $a_i(in_i) = 1$  if  $in_i \geq 0$ , 0 otherwise

FIGURE 1.7. Examples of activation functions

Neural networks (NN) have been used widely for modelling and classification purposes. Their ability to model complex non-linear systems using only a set of neurons in some distribution, provides a great advantage. Also, training process is

generally straightforward and a particular type of NN is suitable for a given problem. However, the challenge of using neural networks is to find a suitable distribution and parameters for any given problem [25].

### Multilayer perceptron

A multilayer perceptron is a very known case of ANN. These feed-forward networks are arranged in layers, such that each neuron (called perceptron) receives input only from neurons in the immediately preceding layer [46]. This way, each neuron output is only dependant on outputs and weights of the preceding layer. The simplest case correspond to a NN in which the input layer is connected directly to the output (see Figure 1.8(a)). In this case, since the output is given by a weighted sum of the inputs and the activation function returns 1 when certain value (threshold) is exceeded (0 otherwise), this network defines an hyperplane which separates two sets of data. This network is only capable of classifying linear separable data and is often called a **linear separator**.

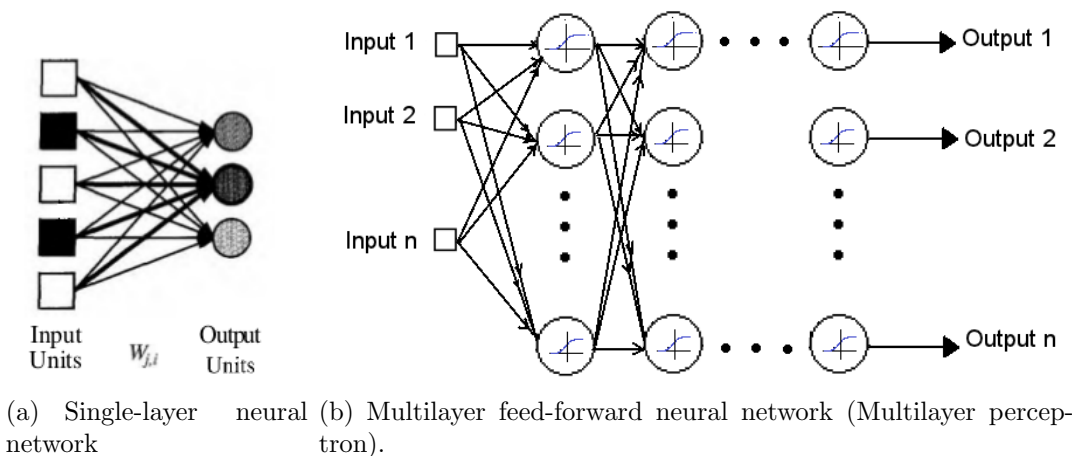


FIGURE 1.8. Feed-forward neural networks

Adding extra layers between the input and output, called hidden layers, allows the network to represent functions more complicated than a simple hyperplane. In fact, with a single, sufficiently large hidden layer, it is possible to represent any continuous function of the inputs with arbitrary accuracy; with two layers, even



discontinuous functions can be represented. Unfortunately, for any particular network structure, it is hard to characterize exactly which functions can be represented and which ones cannot [46]. A multilayer perceptron has always an input layer, an output layer and  $n$  hidden layers (see Figure 1.8(b)). In each layer, there are any number of neurons.

The training process consists on adjusting the network weights in order to get the desired output. Based on a set of known examples, it is possible to find an algorithm that iteratively adjusts the network weights. The gradient descendant algorithm reduces the total Error (sum of squared errors) of the network by adjusting the network weights. This way, in the case of single layer neural network with a single output, the adjusting weights process for a single layer is given by equation 1.2.

$$W_j = W_j + \alpha * Err * g'(in) * x_j \quad (1.2)$$

Where  $x_j$  is the input,  $Err$  is the sum of squared error between the example and the network output ( $Err^2 = (y - g(in))^2$ ),  $y$  is the expected output of the example,  $in$  is the sum of weighted inputs ( $in = \sum_{j=0}^n W_j * x_j$ ) and  $\alpha$  is the learning rate.

A multilayer perceptron with hidden layers (Figure 1.8(b)) uses a similar process to update the weights. However, whereas in the case of a single output the error  $y - g(in)$  is clear for the output layer, the error in the hidden layers is unknown, because the training data does not say what value the hidden nodes should have. It turns out that we can back-propagate the error from the output layer to the hidden layers. The back-propagation process emerges directly from a derivation of the overall error gradient. Equation 1.3 shows the weight update formula for hidden layers.

$$W_{k,j} = W_{k,j} + \alpha * a_k * \Delta_j \quad (1.3)$$

Where  $a_k$  is the input for neuron  $k$ ,  $W_{k,j}$  is the weight from input  $k$  to neuron  $j$  in the hidden layer,  $\alpha$  is the learning rate and  $\Delta_j$  is the back propagated value of the

modified error  $\Delta_i$  of the subsequent layer. Equations 1.4 and 1.5 show the values for  $\Delta_j$  and  $\Delta_i$  for a multilayer perceptron with a hidden layer. More information about back-propagation algorithm can be found at [46].

$$\Delta_i = Err_i * g'(in_i) \quad (1.4)$$

$$\Delta_j = g'(in_j) * \sum_i W_{i,j} \Delta_i \quad (1.5)$$

### 1.5.3 Fuzzy logic

Fuzzy logic is a superset of classical logic (true or false statements) that has relaxed boundaries and approximate or imprecise results (not completely true or completely false) rather than exact [24]. The importance of fuzzy logic derives from the fact that most modes of human reasoning and especially common sense reasoning are approximate in nature. Fuzzy logic systems are a suitable framework for representing qualitative knowledge, either provided by human experts (knowledge-based fuzzy control) or automatically acquired from data (rule induction, learning) [6].

The Fuzzy set theory, formalized by Zadeh (1965), defines a set of rules, definitions and regulations of fuzzy sets. A fuzzy set has some grade of membership specified by a function whose values are in the range  $[0,1]$ , which means that elements can belong to a fuzzy set with a certain degree. Due to this, fuzzy sets can be used to represent vague concepts, such as low, large, small, high, big, old, expensive, etc. [6]. Figure 1.9 shows an example of membership functions for classical and fuzzy logic sets that classifies people according to their age.

In this example, there are two sets that classify people according to their age, between 0 and 100 years. In classical logic, if a person is younger than 50, he is classified as "young", but if the person is 50 or older, he is classified as "old" (Figure 1.9(a)). However, it may be thought that a 48 years old person is not so young. Actually, other people may think that the old people set should start at 40 years,

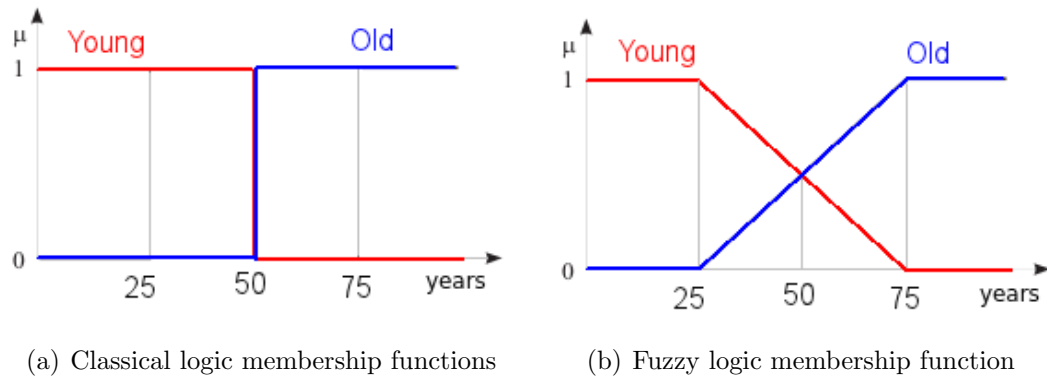


FIGURE 1.9. Comparison between classical logic and fuzzy logic membership functions

or even, should be defined another set for middle age people between 40 and 60. Anyway, there is not a clear limit between young and old people that represents the exact information. Fuzzy logic tries to solve this vagueness by relaxing the boundaries of sets. This way, a person is not totally “young” or “old”, but there is some degree of membership to each one. In the example (Figure 1.9(b)), a person who is 60 years old, is a little bit older than younger and it can not be said that is totally old until he turns to 75 years. Mathematically, he belongs in 40 percent to the set of young people and 60 percent to the set of old people. Further explanation about representation and operation of fuzzy sets may be found in [6, 24].

Based on fuzzy set theory, it is possible to describe static and dynamic systems. Basically, such systems can be described in three ways; as a collection of if-then rules, where the rules or conditions define the relation between fuzzy sets in the input and output; in the specification of system’s parameters, where the systems parameters are given by fuzzy “numbers”; or the input, output and state variables of the system that may be described by fuzzy sets [6]. The advantage of using fuzzy logic is that it allows mapping imprecise or noisy information as input into the output domain. It is particularly effective in autonomous robot navigation since the data is obtained under uncertain conditions [34]. Furthermore, fuzzy logic systems allow the easy inclusion of expert and/or a priori knowledge.

### 1.5.3.1 Rule-based Fuzzy Systems

The most common fuzzy systems are defined by means of if-then rules in the following general form:

**if** antecedent proposition **then** consequent proposition.

Fuzzy propositions are statements like  $x$  is *small*, where *small* is a linguistic label which are referred to as fuzzy constants, fuzzy terms or fuzzy notions. Depending on the particular structure of the consequent proposition, three main types of models are distinguished[6]:

- *Linguistic fuzzy model* (Zadeh, 1973; Mamdani, 1977), where both the antecedent and consequent are fuzzy propositions. Singleton fuzzy model is a special case where the consequents are singleton sets (real constants)
- *Fuzzy relational model* (Pedrycz, 1984; Yi and Chung, 1993), which can be regarded as a generalization of the linguistic model, allowing one particular antecedent proposition to be associated with several different consequent propositions via a fuzzy relation.
- *Takagi-Sugeno (TS) fuzzy model* (Takagi and Sugeno, 1985), where the consequent is a crisp function of the antecedent variables rather than a fuzzy proposition.

### 1.5.3.2 Singleton Fuzzy System

A singleton set, is a special case of fuzzy set which is associated to a crisp number  $x = x_0$ . This way, its membership function its represented by a single real value in the universe of discourse of the variable (see Figure 1.10).

When consequents in a fuzzy system are singleton sets, this special case of linguistic model is known as a *singleton model*. In this sense, the consequent sets  $B_i$  can be represented as real numbers  $b_i$ , yielding the following rules:

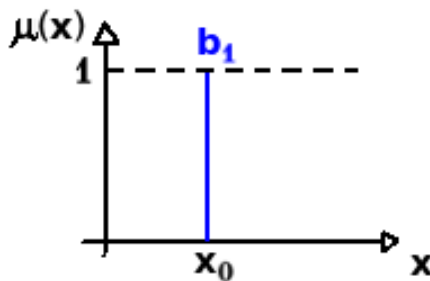


FIGURE 1.10. Singleton fuzzy set

**if**  $x$  is  $A_i$  **then**  $y = b_i$ ,  $i = 1, 2, \dots, K$ .

In this model all the  $K$  rules contribute to the defuzzification. This means that if two rules which have the same consequent singleton are active, this singleton counts twice in the weighted mean. For the singleton model, the COG defuzzification results in the *fuzzy-mean method*:

$$y = \frac{\sum_{i=1}^K \beta_i * b_i}{\sum_{i=1}^K \beta_i} \quad (1.6)$$

The singleton fuzzy model is also a special case of the Takagi-Sugeno model (**if**  $x$  is  $A_i$  **then**  $y_i = \mathbf{a}_i^T \mathbf{x} + b_i$ )[6] where the output parameter vector  $\mathbf{a}_i^T = 0$ . Also, the singleton fuzzy model belongs to a general class of general function approximators, called the basis functions expansion, as Equation 1.7.

$$y = \sum_{i=1}^K \phi_i(x) b_i \quad (1.7)$$

In the singleton model, the basis functions  $\phi_i(x)$  are given by the (normalized) degrees of fulfillment of the rule antecedents, and the constants  $b_i$  are the consequents. Multi-linear interpolation between the rule consequents is obtained if:

- the antecedent membership functions are trapezoidal, pairwise overlapping and the membership degrees sum up to one for each domain element

- the product operator is used to represent the logical and connective in the rule antecedents.

“This property is useful, as the (singleton) fuzzy model can always be initialized such that it mimics a given (perhaps inaccurate) linear model or controller and can later be optimized” [6].

## 1.6 Intelligent control

In recent years, novel mechanisms of control have emerged from the combination of artificial intelligence and classical control theory. This, known as Intelligent control has become a very important research field in the robotics applications. In it, some of the methods of computational intelligence (CI) are used in order to improve controllers performance and adaptation features. Also, CI allows controllers to automatically create models, adapt to changes and to learn specific tasks.

Artificial life, as a field of computational intelligence, has demonstrated to be a very promising solution for the control motion problem [31]. These biology inspired methods have become one of the most popular methods for motion control development and novel control architecture design. This may happen because nature has found very elegant methods to provide motion to animals in almost all the environments. So, it seems straightforward that the idea of emulating the biology processes in the motion control could be one of the best solutions. However, control theory is the base of actual movable robots and as the author states [47], there is a relation between methods and models developed in control theory and the motor motion control in biological organisms.

### 1.6.1 Control using genetic algorithms

Genetic algorithms have been widely used for control applications giving interesting results as a searching method. GAs can obtain different solutions without an specific

knowledge of the process giving the controllers the chance to explore and find a satisfactory solution for an specific control task.

Two different approaches allow using GAs for control: to directly determine the controller output or as a supervisor of some well known controller. In the first case, the GA directly produce the inputs of the system and the evaluation is done applying the input to the system and seeing how its performance is. An example could be seen on the obstacle avoiding robot control [23], where there is an unknown environment and the robot has to explore the area and to avoid the obstacles in it. Also, Genetic programming (GP), developed by Koza [29] is a useful tool for robot control. With GP, a program is evolved in order to fulfill some task. Koza [30] also proposed a method to control a robot in order to move a box to an specific position using GP for finding instructions. In the second case GA is used to schedule the parameters of a controller.

This controller could be of any kind and the variable parameters may be as many as the designer considers. Like in the first case the evaluation of the individual performance is done by applying the parameters and testing it directly in the system. Linkens[35] makes an interesting approach where the knowledge base of a fuzzy controller is determined by the GA.

### 1.6.2 Fuzzy logic controllers

A fuzzy controller is a controller that contain a (nonlinear) mapping that has been defined by using *if-then rules* [6]. Fuzzy inference systems basically use three steps to make the mapping between input and output; fuzzyfication, inference (Knowledge base-if then rules), and defuzzyfication. Control dynamic filters are added in order to obtain input and output signals corrections (Figure 1.11).

Fuzzy control may be achieved through two different approaches: (1) by obtaining a process model through the use of nonlinear techniques and then using the model as a basis for model-based control, and (2) without having a explicit model process where nonlinear techniques can be used to design the controller directly.

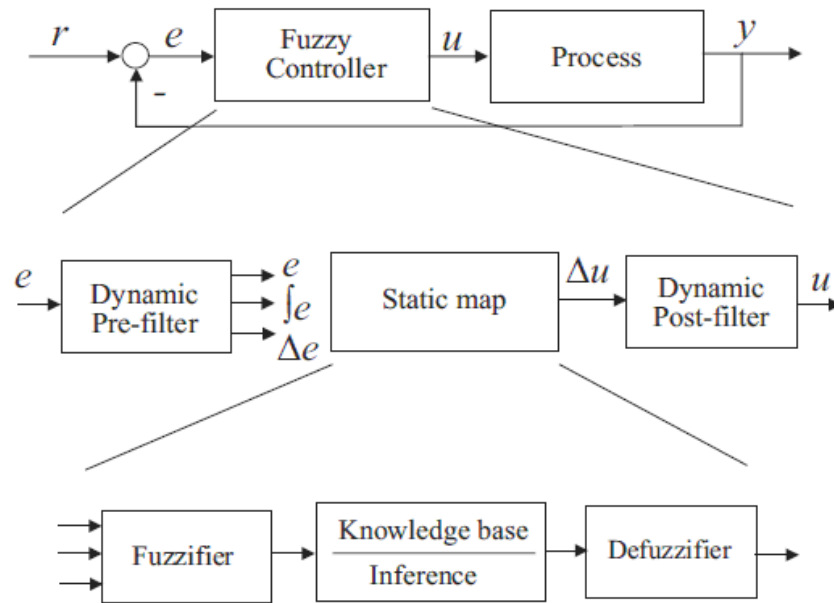


FIGURE 1.11. Non-supervisory fuzzy control diagram

Also, from the computational intelligence point of view, two different approaches are possible; supervisory or non-supervisory. Supervisory approaches, achieve control through the use of a second controller (the fuzzy controller), which augments the existing controller so the control goal can be met. Non-supervisory techniques control the system directly through the dynamic or static mapping defined by the controller (feedback and feedforward fuzzy schemes are used on this approach). Figures 1.11 and 1.12 show these approaches respectively.

The main advantage of using a fuzzy controller is the possibility to easily implement the human experience, intuition and heuristics into the controller. However an often remarked disadvantage of the methods based on the fuzzy logic is the lack of appropriate tools for analysing the controllers performance, such as stability, optimality, robustness, etc.

### 1.6.3 Evolutionary Robotics

Evolutionary robotics (ER) is a recently new area which draws inspiration from the process of natural evolution. ER is primarily concerned with the use of artifi-



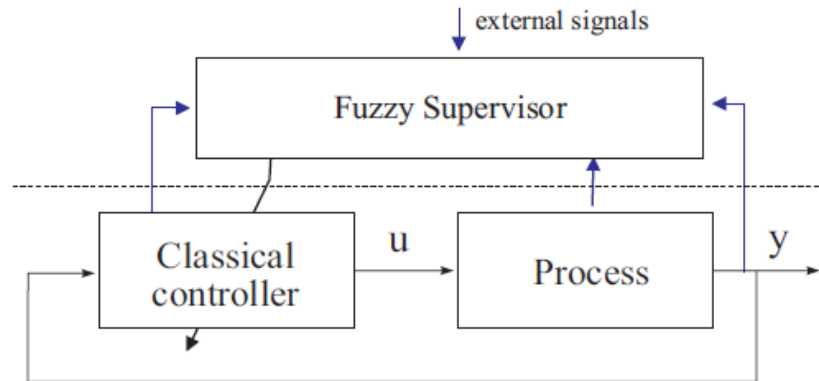


FIGURE 1.12. Supervisory fuzzy control diagram

cial evolution techniques (evolutionary strategies, genetic algorithms, evolutionary programming) for the automatic design of adaptive robots [34].

The main application of evolutionary robotics is in designing autonomous robots. Such kind of robots must be able to adjust their behavior by themselves online and make appropriate decisions under various uncertainties encountered during their motion in an independent and smart fashion. The challenge is to design control algorithms or methodologies that allow robots to function in complex, challenging and partially known environments [38]. This is not an easy task, since the environment and robot conditions and goals may change over time. Also, it is difficult to model unstructured external environments perceived via sensors in sufficient detail, since they are changing continuously. Therefore, traditional robotic controllers designed for factory automation are not suited for these types of applications anymore.

When building autonomous robots, there is a variety of robotic architectures which control the robot. Hybrid and behavioral-based control architectures are the most commonly used whether for decomposing the robotic system into high and low level layers [41, 53] or for using a set of selected behaviors [22, 15, 36], respectively. Behavioral-based control is widely used because of its similarities with biological behavior. However, the challenge lies in designing the behavioral modules and coordination mechanisms between such modules. Both the robotic behavior modules and the coordination mechanisms are designed through a trial-and-error procedure

[34]. By doing so, the designer gradually adjusts them and examines corresponding behaviors until the satisfactory robotic behavior is derived. Evolutionary Robotics (ER) attempts to use Evolutionary Algorithms (EA) [39] to automatically design those sensorimotor control systems and avoid the trial-and-error procedure. The fundamental principles of evolution mechanism include a population of individuals competing for survival, inheritance of advantageous genes from the parents and the offspring variability, which may result in a higher chance of survival. ER enables robotic system to adapt to unknown or dynamic environments without human intervention. The robotic controllers derived by evolutionary methods have advantages of relatively fast adaptation time and carefree operations [16, 42].

In evolutionary robotics, each chromosome encodes the controller or morphology of a robot. Then, physical or simulated robot acts guided by the robotic controller, and in the evolutionary process, the robotic performance in various situations is evaluated. As in any EA, a given population is evolved until an individual with satisfactory performance is attained. Under the framework of the evolutionary algorithm-based robotic controller, the sensory inputs are encoded into the genetic strings and sent to the evolutionary algorithm based robotic controller. The robotic controller makes appropriate decisions based on the real-time sensory values and sends its decision to the execution devices such as motors and actuators [34]. Other relevant techniques for autonomous robotics include artificial life [9], modular systems [27], neural networks [8, 17] and plan-based control [7].

## 1.7 Motion learning and intelligent control approaches

Particularly talking about the motion problem in robots, authors in the field have proposed methods to address this problem based in some of the concepts showed in this chapter, specially Evolutionary Robotics. The main advantage of these approaches is that it is assumed a lack of information in some aspects about the robot like its dimension, its configuration or the number of actuators, which makes the

problem very complex to solve through the traditional robot analysis. In this section, some of the most relevant approaches to the work developed here are shown.

### 1.7.1 Automated Learning of Muscle-Actuated Locomotion Through Control Abstraction

In this work, Grzeszczuk and Terzopoulos [22] proposed a method of learning locomotion skills. The system is focused on learning motion abilities for physics based models of animals with "high flexibility, many degrees of freedom bodies and considerable number of muscle actuators, such as snakes and fishes" (see Figure 1.13).

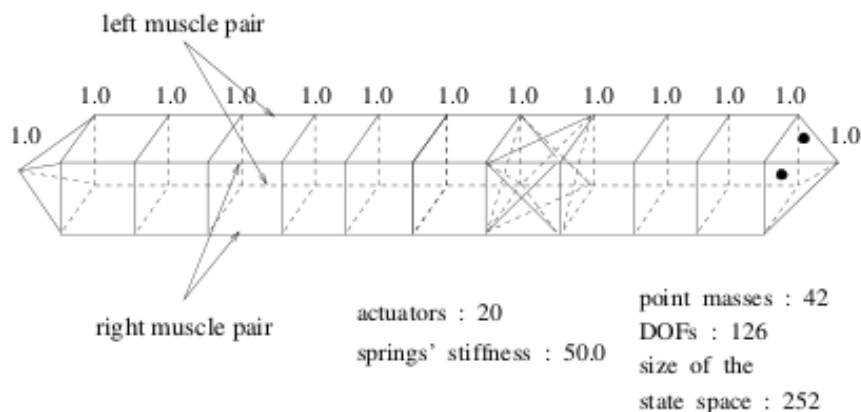


FIGURE 1.13. Biomechanical model of a snake. "Model consists of nodal masses (points) and springs (lines). It has twenty independent actuators (muscle springs): ten on the left side of the body and ten on the right side." Taken from [22]

Basically, the proposed method is divided into two stages: a low level learning process of motor skills and high level task execution (specified by the animator like *moving towards a visible target*).

The first stage is based on the fact that there is no a priori knowledge of how to locomote. As the author states "*It is as if the animal had a fully functional body, but no motor control center in its brain*". Here, a learning process is done by simulating the dynamic of a body for a given controller and motion goal. The simulation is carried out in short periods of time ( $t_0 < t < t_1$ ) and an objective functional is used to iteratively optimize controller parameters. Objective functional (see Figure 1.14)

is given by the weighted sum of terms that evaluate the trajectory and the control function (controller parameters). Controllers are discretized in time and frequency domains in order to optimize the objective function. Both simulated annealing and simplex algorithms are used to optimize the functional. Experiments with snake, ray and shark models successfully demonstrate the method ability to learn low level controllers.

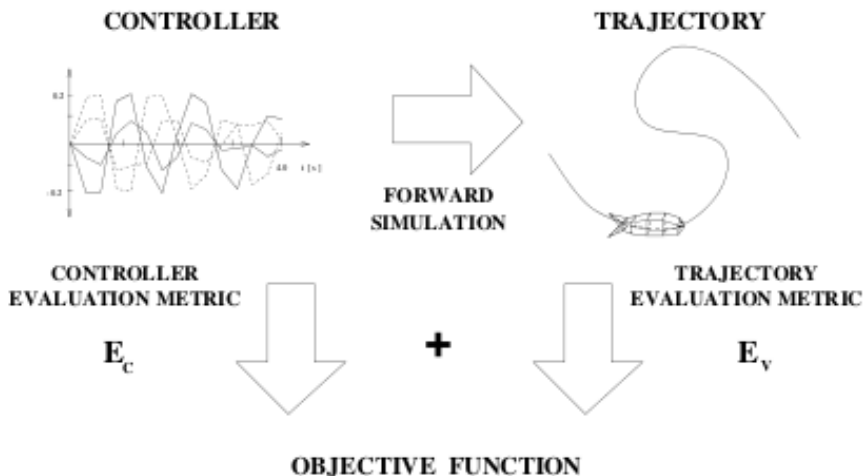


FIGURE 1.14. Objective Function. Taken from [22]

In the second stage the learned controllers are used to perform more complex tasks, for example making a dolphin stunt over the water or following a motion path. However, given the high dimensionality of the problem, the lack of gradient information and the presence of local optima, some abstraction of the controllers is done before using the low level controllers in order to simplify the complexity. In this sense, two methods are implemented to perform these tasks. In both of them, a set of learned low level controllers are applied to navigate in the direction of a target. These abstracted controllers are basic movements like, do-nothing, go-straight, turn-left, turn-right, etc. In the first case, called by authors “greedy fashion”, the animal modifies its locomotion strategy periodically by invoking the abstracted controller that gives it the greatest immediate gain over the subsequent time interval. Here, the animal discovers how to sequence these controllers, and how long to apply them in order to locomote to successive targets (path following). In the

second case, a process similar to the learning method applied for low level controllers is used. Simulated annealing is used to construct a macro controller as an optimized sequence of basic abstracted controllers. Here, “*rather than optimizing over nodal parameters or frequency parameters, it optimizes over the selection, ordering, and duration of abstracted controllers*”. This second approach overcomes problems of the greedy strategy when a careful planning process is required like *making a dolphin stunt*. However, additional terms must be included in the functional in order to achieve controls of the animal’s trajectory in the air. Also, as the authors state, at a high level of abstraction even small changes in parameters produce drastically different trajectories and no advantage of partial solutions is taken into account by the optimization process.

### 1.7.2 Spontaneous Evolution of Structural Modularity in Robot Neural Network Controllers

Broadly speaking, in this approach Bongard [8] propose a method to evolve large robot controllers based on boolean neural networks. Here, rather than create a global control system of a robot, the author focused on evolve structural modularity of those neural networks which conform the motion controller of the robot. The structural modularity specialises some parts of the robot control system into a given task, creating certain groups of neurons densely connected while other remain unconnected or with few connections. The method in particular, as the author states has the advantage that “*the user does not need to specify the groups beforehand nor formulate explicit module creation evolutionary operators*”. Besides, it could be scaled to more complex robots and tasks.

The robot used to test the proposed method is a simulated two dimensional robot, composed by seven parts and 6 degrees-of-freedom (DOF). The robots emulates an arm and hand system with 3 segments for the arm and 2 segments for each finger (see Figure 1.15). Each joint has a rotation movement constrained to the range of  $[-45^\circ, +45^\circ]$  and the purpose of the robot, in terms of the author, is to “*grasp*” the

circle in any of four possible environments. In other words, the robot must move in such way that the fingers tips meet the circumference in the environment. Concepts of mass, motor strength or momentum are not taking into account in the simulation.

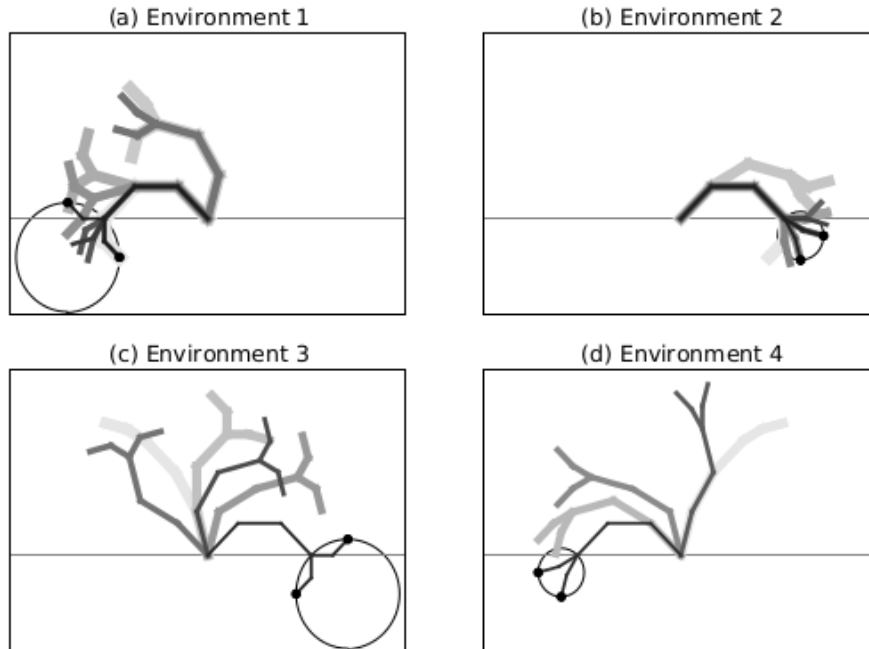


FIGURE 1.15. Simulated 2D robot. There are four different environments where the robot is tested. Difference relies on where is the circumference (left or right) and what is its size (small or large). Taken from [8]

Artificial neural networks (ANNs) control the robot movements. These ANNs are known as deterministic boolean networks and they are encoded as synchronous. This way, neurons take binary values and connections between them are binary as well (excitatory or inhibitory). The controller uses 2 nodes (neurons) for each of the seven mechanical degrees of freedom (DOFs), yielding a boolean network with a total of 14 nodes. Additionally, controller has 28 ( $2 \cdot 14$ ) excitatory and inhibitory connections in the net.

Neural networks of the controller act as both, sensory inputs, which give information about the environment, and as the controllers of the robot, commanding the corresponding movements. Thus, in the first time step, each couple of nodes indicates where the object is (left=-1 or right=+1) and the size of the object (small=-1 or large=+1). Once the robot is provided with information about the environment

in the first time step, each joint of the robot moves to one of four possible angles coded by the couple of nodes as well ( $-45^\circ \rightarrow [-1,-1]$ ,  $-15^\circ \rightarrow [-1,+1]$ ,  $15^\circ \rightarrow [+1,-1]$ ,  $45^\circ \rightarrow [+1,+1]$ ). This way, in the first time step the robot is moved to the initial position dictated by the sensor nodes, which may be perturbed by some noise (by flipping one bit of the 14 node values). Then, the controller is updated until it reaches an steady state or a maximum of 12 updates in the standard fashion for boolean networks.

An evolutionary algorithm is used to evolve a population of 10 controllers in order to find one, capable of reaching the circumference with the robots fingers in all of 60 possible conditions (fifteen possible initialization positions for each one of the four environments). Environmental conditions are not evaluated all at the same time, instead they are added subsequently in the evolutionary process. In this sense, the controllers are evolved until a controller succeeds one environmental condition. Then, a second condition is evaluated against the controllers and evolution continues until both conditions are succeeded. The process continues until all 60 conditions are accomplished.

Fitness function (see Equation 1.8) is set in order to obtain  $f = 0$  if the robot's fingertips are far from the circumference in all of the conditions tested and  $f = 1$  if the fingertips contact the circumference in all of the conditions. Terms  $D_L$  and  $D_R$  correspond to the distance from the left and right fingertip, respectively, to the circumference, and  $D_{max}$  is the maximum distance that a fingertip can be from the circumference.  $i$  and  $j$  refer to the test of the  $i$ th initial condition in the  $j$ th environment.

$$f = \sum_{i=1}^{15} \sum_{j=1}^4 \left(1 - \frac{D_L + D_R}{2D_{max}}\right) / 14 \quad (1.8)$$

The selection mechanism of the evolutionary algorithm consist on elitism; where the best controller is copied into the next generation and fitness-proportionate selection and mutation is applied to the other 9 individuals. Mutation consist on adding or removing incoming connections of an specific node  $u$ , which is selected with a

probability of 5%. In particular, removing or adding a new connection to a node is performed with a probability  $p(u)$  and  $1 - p(u)$ , respectively. The probability function  $p(u)$  is given in terms of the number of regulators  $r_u$  (incoming connections of the node  $u$ ) and is biased in order to ensure that nodes maintain an average of 2 or 3 incoming connections as observed in previous research.

Test is performed with two different shaping schedules and three different fitness constraints. Difference between shaping schedules relies on how the several initialization conditions and environments are added into the evolutionary process of the controllers. In the first shaping schedule each environment needs to complete all initialization conditions before adding a new environment to the evolutionary process. Conversely, the second shaping schedule adds a single condition of each environment simultaneously (until that condition be met in all environments) and then adds a new condition into the evolutionary process. Fitness constraints (which are achieved by adding a constant to the fitness function) affect the evolutionary process by allowing the controllers to converge on: point attractors, point and cyclic attractors or any behavior.

Bongard demonstrates with his simulations that modularity is arisen in all the evaluated cases with some differences. Specifically, this behavior happens if the controllers are boolean networks, and are selected to converge on point attractors that correspond to successful robot behaviors. According to him it seems that it may be easier to evolve modularity in dynamical systems, compared to systems that do not have their own intrinsic dynamics.

## 1.8 Summary

In this chapter, concepts about robots, control architecture, fuzzy logic, neural networks and genetic algorithms were presented. These concepts were used in the next sections in order to provide the tools to develop the proposed methods in the present work. Special attention was paid to some Evolutionary robotics approaches and Grzeszczuk work. Also, computational intelligence methods such as fuzzy logic



---

control, genetic algorithms, neural networks and subsumption architecture were used to construct the control architecture. In the next chapter, a simulator was developed in order to provide the framework in which the robotic system was simulated.

---

# Chapter 2

---

## Robot representation and simulation

In the previous chapter, concepts about control systems, artificial intelligence, control architectures, evolutionary robotics and motion learning were presented. Those concepts are used in this and further chapters in order to develop not only the control architecture proposed in this project but also to develop the necessary tools to test it.

The control architecture developed in this project is thought to be applied in robots. However, building robots is a hard, expensive and time consuming task. Simulation software is widely used to design equipment so that the final product will be as close as possible to design specifications without expensive costs in modification processes. In this project a simulation environment is developed in order to test the control architecture in simulated robots. This chapter describes this simulation environment and the simulated robots design. The chapter is divided into three main parts: simulation architecture, simulated robot specifications and graphical interface. The first part, presents the architecture of simulation software, i.e how the software manage information to perform robot's movements simulations. The second part describes how robots are modelled and represented in simulation. Finally, the last part makes a brief description about user interface software.

## 2.1 Simulator architecture and settings

Simulation purpose on this work is to represent the mechanical behavior of a robot in a given environment. This simulation software includes the robot control system, the robot motion learning algorithm and the robot motion goal. In general, the simulation architecture is divided into three basic parts: the simulation environment, the robot and the graphical interface. Figure 2.1 shows a diagram of the simulator architecture.

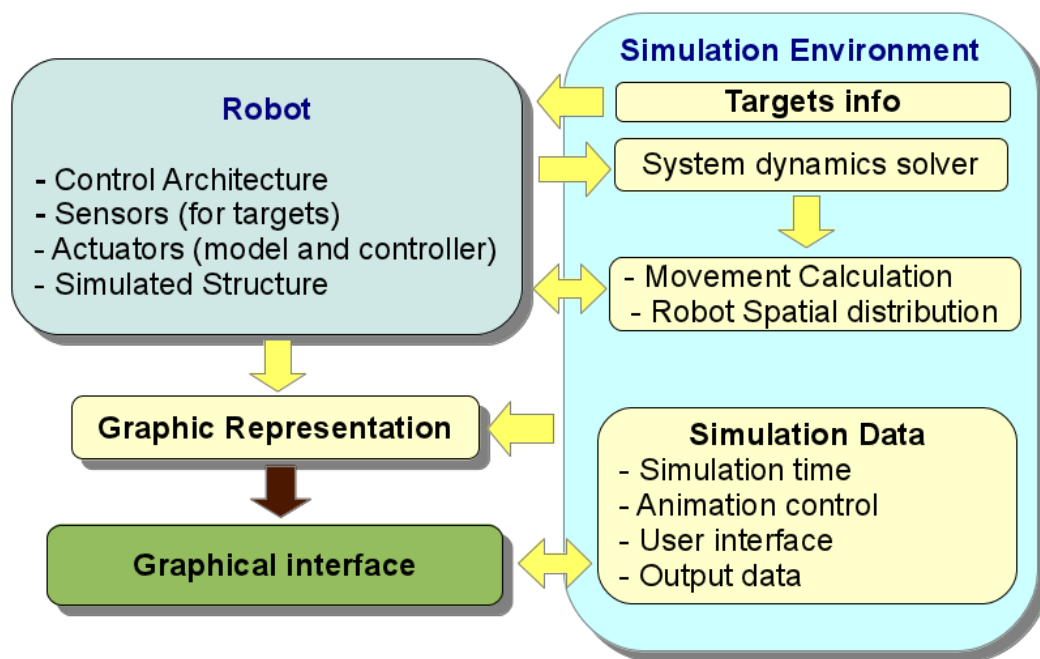


FIGURE 2.1. Simulator architecture

All simulation architecture parts act independently from the others but require to share information between them. Some characteristics of each part are described below:

- The simulation environment is the main part of the simulation system<sup>1</sup>. Simulation is totally controlled by this part which also manage outputs, time, users commands and the robot's targets. Here, the environment is simulated

<sup>1</sup>The simulation environment was implemented using Java programming language (JDK v1.6)[1]

by solving actuators dynamics and calculating robot movements for each time step. Also, it returns information about the environment and the robot to other simulation parts (for example robot sensors or graphical interface).

- The robot part in the architecture, represents a robot as a agent capable of performing actions and making decisions. It is composed by a set of virtual components representing structure, some sensors, some actuators and the control system of the robot. Each actuator of the robot requires a model (that represents the dynamical behavior of the robot) and a controller (that controls determined parameter in such actuator). Sensors of the robot gather information about the environment or the robot itself. Finally, the control system manage the robot sensors and actuators in order to achieve specific motions.
- The graphical interface interprets data from the architecture and the robot and display it on the screen.

## 2.2 Simulated robot specifications

Movement of the robot depends on two things; the purpose of the robot and the way the robot is built (how the parts of the robot are arranged). Robots purpose consist in performing an specific motion based on a given target. The motion involves moving all actuators of the robot in the right manner in order to reach this target (an actuator movement can also be “to stay still”). In addition, constraints in the actuators movement and simulated sensors measures are added to the robot simulation.

In this project the robots were defined by a 2D geometry using links and joints, similarly to a robot manipulator or to Bongard [8] robots representation (see Chapter 1). Multiple different robots can be constructed by assembling this parts in different forms. Figure 2.2 shows various examples of robot structures that can be created.

In this representation, actuators of the robot corresponded to DC motors under the control position scheme (which is equivalent to servomotors). These motors

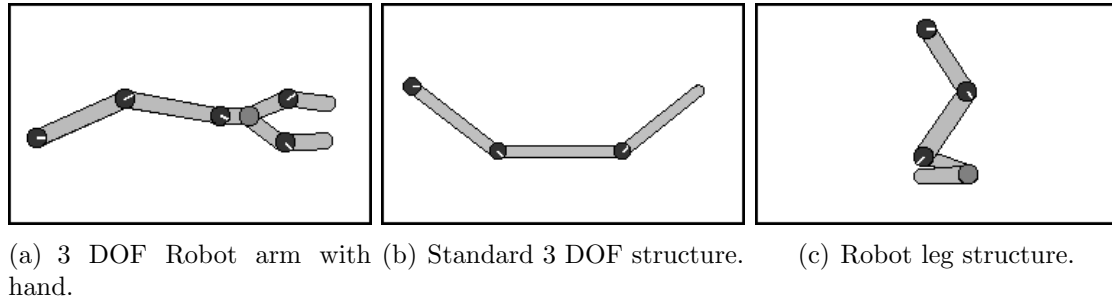


FIGURE 2.2. Simulated robot examples

were represented by a round point with a line. This line depicts for the reference angle of the motor which in this case is 0 degree for convenience, but it may be any angle. Additionally, simulated robots were built taking into account a fixed point (the darker one) which remains static at its position and is the base to calculate the movements of the robot. Simulation has no gravity or forces models. Also, simulated robots can not move from the fixed point (base). This may seem unreal, but the idea was not to provide a real simulation of the world, but just a system in which we can prove it was capable of learning how to move properly in a given environment.

### 2.2.1 Robot arm

For this project, a simple structure of a robot arm or manipulator was defined. This arm is composed by the union of some actuated joints and links. The number of joints and links were determined by the degrees of freedom (DOF) of the arm. Figure 2.3 shows robot arms with 2, 3 and 4 DOF. Notice that the first motor (the one in the left most side) corresponds to the fixed point or base of the structure (which is actuated as well).

#### 2.2.1.1 Robot arm targets

Targets were defined as points in the space that the robot arm must reach. The goal of the robot is that the furthest part of the arm  $A_p$  (see Figure 2.4(a)) must

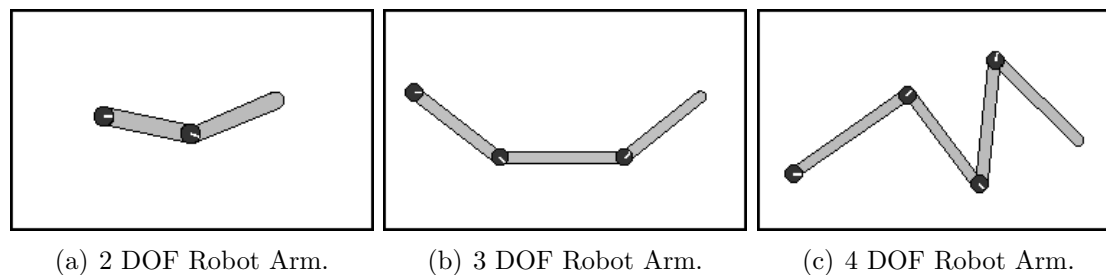


FIGURE 2.3. Structure of simulated robots

be as close as possible to the target after the movement is completed. This way the goal of the robot is to find the correct angles for each joint that makes arm reach the target.

This is a classic problem in the kinematic analysis (see Chapter 1) and can be solve easily through this methodology or just by applying geometry analysis. However, as stated before, those methods require exact knowledge about the robot dimensions and construction. Hence they are not applicable for the purposes of this thesis. Instead, artificial intelligence methods are applied to allow the robot learn to move through a trial and error procedure. However, to achieve this learning mechanism, the robot needs a way to measure how close of the target are its movements. In this sense, some sensors must be added to robot to determine where is the target and how close is the robot to the target.

### 2.2.1.2 Robots arm sensors

In simulation, it is assumed that the robot has adequate sensors to obtain three different measures: the distance from the point  $A_p$  to the target and, distance and angle to the target from its position. This position was defined as the fixed point or base. Figure 2.4(a) shows measures taken by the robot for a given target. Angle measure was taken in relation to the fixed point orientation (0 degrees for convenience) and distances are Euclidean. Robot sensors and actuators have limits, so a maximum movement angle  $\gamma_m$  for each motor was defined as well as a maximum vision distance of the robot  $V_m$  (see Figure 2.4(b)).

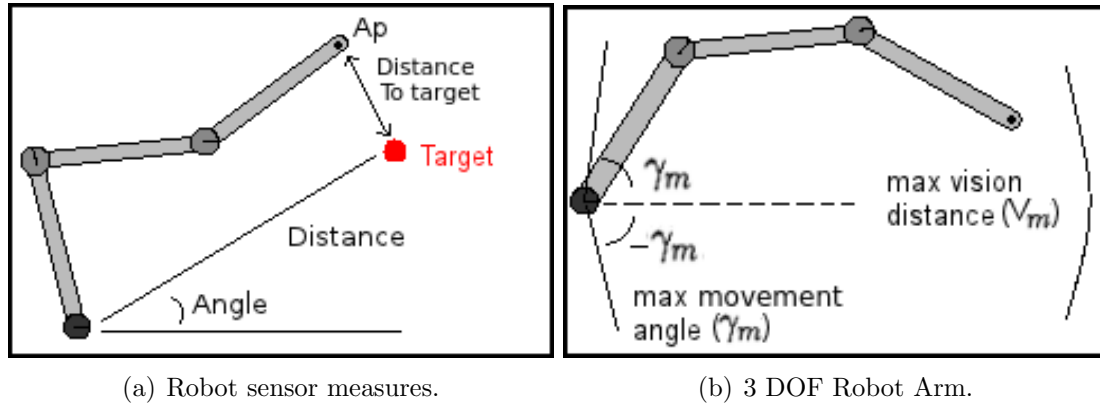


FIGURE 2.4. Robot sensors measures and limits

## 2.2.2 Motor model and Controller design

Motor dynamics were represented by a second order model (Equation 2.1) which is controlled by a fuzzy controller in closed-loop schema. The controller was chosen to be as simple as possible and also to provide a good enough response for several motors with similar models. This means that the controller is not limited only to one DC motor in determined structure, but it is possible to control different motors in different conditions.

### 2.2.2.1 Motor model

A second order model was used to represent the dynamic behavior of the DC motors. These models are used very often for DC motors modelling as they can approximate very well, real motor behaviours and their controllers design is straightforward. Complex models may be more accurate but require more computation and its use is not justified for the purposes of this work. Equation 2.1 shows the transfer function of a second order model.

$$H(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \quad (2.1)$$

It is assumed that the motor behaviour is stable so the values of  $\omega$  and  $\zeta$  were chosen in the convergence region. Each motor can and should have different models, but for convenience, initially the same model was used in every motor of the robot. Equation 2.2 shows the motor model used in the simulations.

$$H(s) = \frac{4}{s^2 + 5s + 4} \quad (2.2)$$

The plant parameters were chosen as  $\omega_n = 2$ ,  $\zeta = 1.25$

Runge-Kutta method [33] to solve ordinary differential equations (ODEs) was used in order to solve the implicit equations in the second order model of the motor.

### 2.2.2.2 Fuzzy controller

A direct fuzzy controller was implemented in order to control the position of the motors<sup>2</sup>. These controllers are often simple to define and provide great flexibility to improve control systems in several ways. Also, because there is some knowledge about the behavior of DC motors, it is possible to easily combine adaptive characteristics of fuzzy systems and DC motor dynamics previous knowledge.

Inputs error ( $e$ ) and change of error ( $de$ ) were chosen to be inputs of the fuzzy system. The output of the system is the control signal  $u$ . The controller output is incremental in order to get zero stationary error. Figure 2.5 shows the controller diagram.

The fuzzy controller is a singleton inference system. The fuzzy system is conformed by membership functions showed in Figures 2.6, 2.7 and 2.8 and the rule base showed in Table 2.1. These functions were chosen in order to provide to the controller a stable response for a common second order system. Values of the rule base table correspond to consequent constant output parameters.

---

<sup>2</sup>The fuzzy controller was implemented into the simulator using jFuzzyLogic libraries (Open Source Fuzzy Logic library and FCL language implementation) provided as open source by Sourceforge[2]



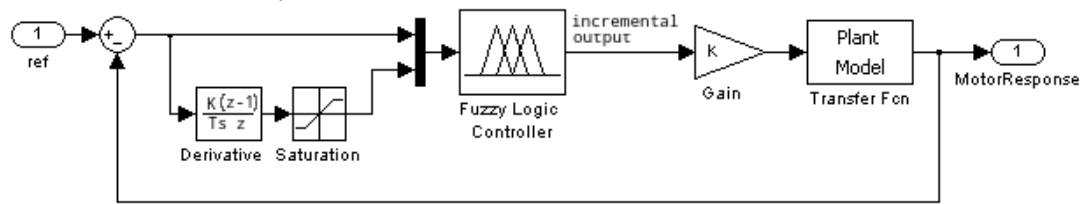
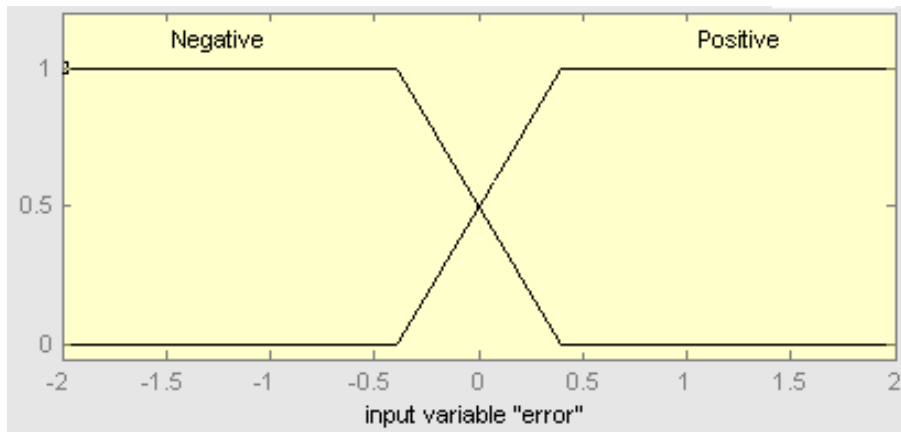


FIGURE 2.5. Fuzzy controller diagram

FIGURE 2.6. Input variable error  $e$  membership function

### 2.2.2.3 Fuzzy controller simulation

In order to test the fuzzy controller response, a simple simulation of step response was performed. The second order model is set as Equation 2.2 and the controller gain is set to  $gain = 0.175$ . Reference signal was varied between -1 and 1 with different values in order to see the controller response. Figure 2.9 shows the response of the fuzzy controller.

Rule base	error derivative $\frac{de}{dt}$	
	<i>Negative</i>	<i>Positive</i>
<i>Negative</i>	0	-1
<i>Positive</i>	1	0

TABLE 2.1. Rule Base - Rules for the fuzzy inference system

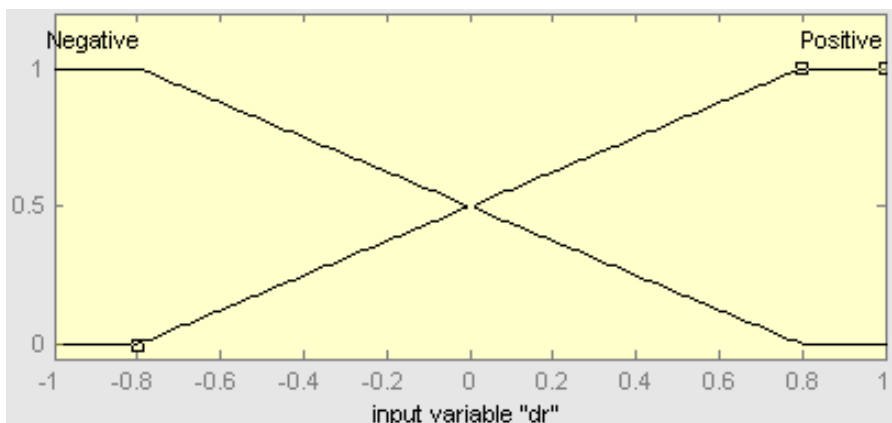
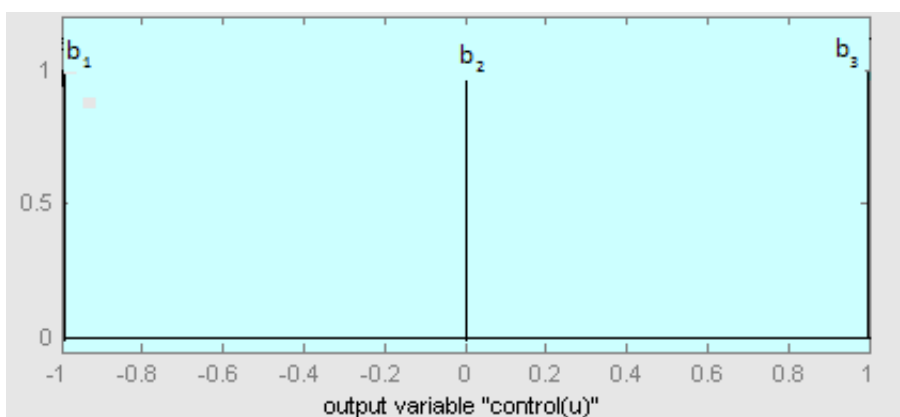
FIGURE 2.7. Input variable error change  $de$  membership functions

FIGURE 2.8. Output control variable

In the simulated environment, the reference signal in degrees  $ref_{degrees}$  is transformed into the range of -1 to 1 by normalizing the angle  $\gamma_m$ , then it is applied to the controller and the output signal is then re-transformed to its original range. Equations 2.3 and 2.4 show the equations used in this process.

$$ref = \frac{ref_{degrees}}{|\gamma_m|} \quad (2.3)$$

$$output_{degrees} = ref * \gamma_m \quad (2.4)$$

Although, this response is not the best, in terms of settling time and oscillation, it has zero steady state error and its response is almost the same for different references.

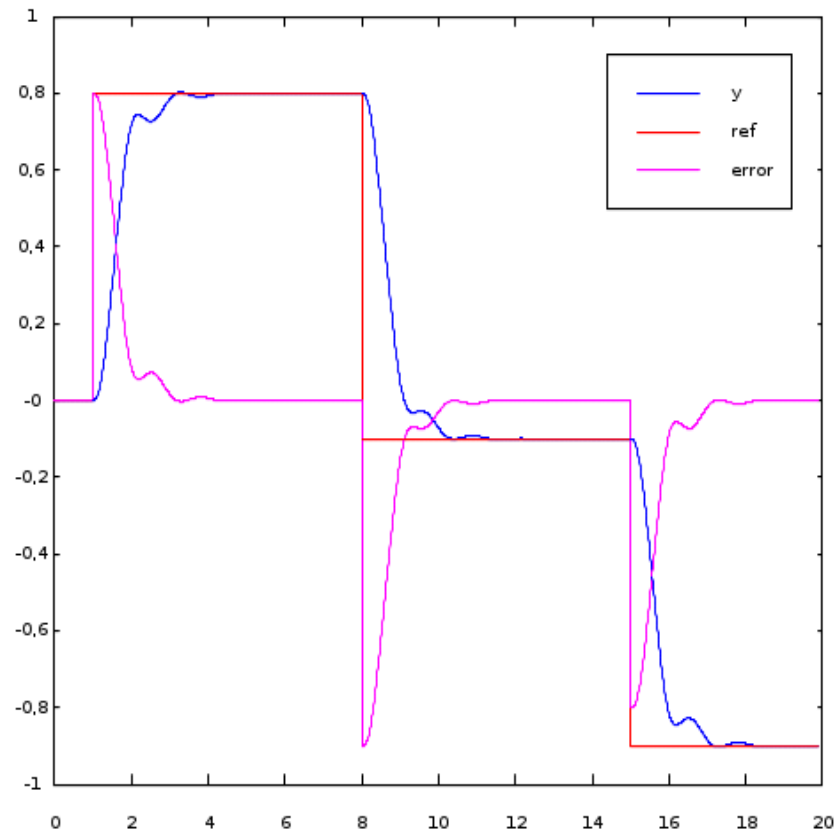


FIGURE 2.9. Fuzzy controller step response

In this project, the controller purpose is only to ensure the movement of the motors in a determined lapse of time. The fact of not getting a accurate response is overcome by the learning process developed in the next chapter.

## 2.3 Graphical interface

The graphical interface provides a 2D simulation environment which allow the user to watch and control simulations. Figure 2.10 shows the graphical interface designed.

The simulator is composed by the following parts:

1. **Animation window:** is where the animation of the robot's motion is performed.

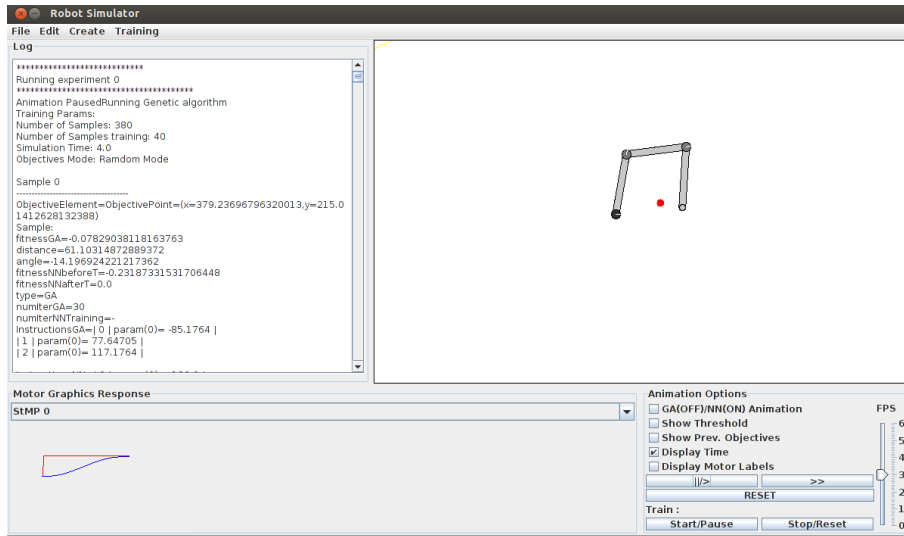


FIGURE 2.10. Simulator environment

2. **Log:** It is place at the left of the simulator and it shows the text output of the running algorithm.
3. **Motor response graph:** On bottom side of the environment a graph of the motor response and reference is shown with analysis purposes.
4. **Animation Options Panel:** It is place at the bottom-right of the simulator and it allows to control the animation and some of the items of the animation panel.
5. **Menu bar:** This bar allows to save and load files, and change simulation parameters and control architecture parameters.

This interface has been very useful during the development of this project since it has allowed to easily interpret the motion of the robot in order to test programming errors in the simulator design, and later in the architecture. Also the characteristic to load, save and change parameters in the simulator makes quite feasible the job providing great accessibility at moment of controlling the robot.

## 2.4 Summary

A simulation environment was designed in order to provide a framework in which it is possible to simulate a robot behavior for an specific control architecture. A simple representation of a motor based in DC motor model and a fuzzy controller under the position control scheme were used for simulation. The robot was constructed with a series of joints and links arranged in some geometry which conform its structure. Some movement constraints and sensors measures were added to the structure as well. Additionally, the purpose of the simulated robot was defined as a set of targets that the robot must reach.

In the next chapter, the control architecture will be implemented in the simulated robot in order to learn a set of movements based on the defined targets. This architecture is based on the subsumption concepts and uses some artificial intelligence techniques to achieve motion learning.

---

# Chapter 3

---

## Motion Learning

In the previous chapter, many components of the simulation software (architecture, graphical interface and the robot characteristics) were described in detail. Also, some aspects of the robot control system were mentioned, however its detailed explanation was intentionally left unaddressed. The present Chapter describes this component of the robot (the control system) which is called *the control architecture*. The motion learning approach developed in this project is encoded in this system. Remember that the final goal of this thesis is to provide a robot with a mechanism that allow it to learn the proper movements for given task without an exact knowledge about its own construction (i.e its dimensions or actuators).

This Chapter is divided into four sections: A subsumption architecture approach, finding robot's movements, learning robot's movements and improving the learning process. The first section presents a general description of the control architecture, its components and its design, which is based on the subsumption concepts. Second and third sections implement methods for finding robot movements and learning those movements, respectively by using AI techniques. The fourth section defines and tests some improvements to the finding and learning motion processes of the robot.

Except for the first section, each section, implements (in software) and simulates different components in the control architecture. Simulation of each component is performed taking into account a dependence between subsequent sections (2;2 and 3;2,3 and 4).

### 3.1 A subsumption architecture approach

An advantage of a subsumption architecture is that layers are often easy to define and expansion of the control system can be achieved by adding more layers. The designing process is usually bottom-up, where each layer, from the simpler (lower layers) to the complex (higher layers) is a human-designed piece of hardware or software responsible for a particular behaviour [51]. These characteristics make subsumption architecture suitable for modularity and learning. Behaviors for each layer can be automatically learned and added to control system in order to improve not only the robot performance but also to add more functionality on the fly. Figure 3.1 shows a scheme of the subsumption control architecture proposed in this project.

The architecture is divided into three different levels; level 0, level 1 and level 2. Operations of each level are described below.

- **Level 0 - Low level control:** This layer directly controls actuators of the robot and tries to keep actuators in a less energy consumption state. Behaviors of this layer correspond to low level control systems such as position control or speed control. Also, controller learning algorithms can be added to improve actuators performance.
- **Level 1 - Basic movement learning:** This layer overrides commands in layers of level 0 in order to achieve specific movements of the robot such as, close a robot hand, reach some point or turn around. These movements are based on targets and involve moving all actuators at the same time. Learning algorithms for motion learning can be implemented as well.

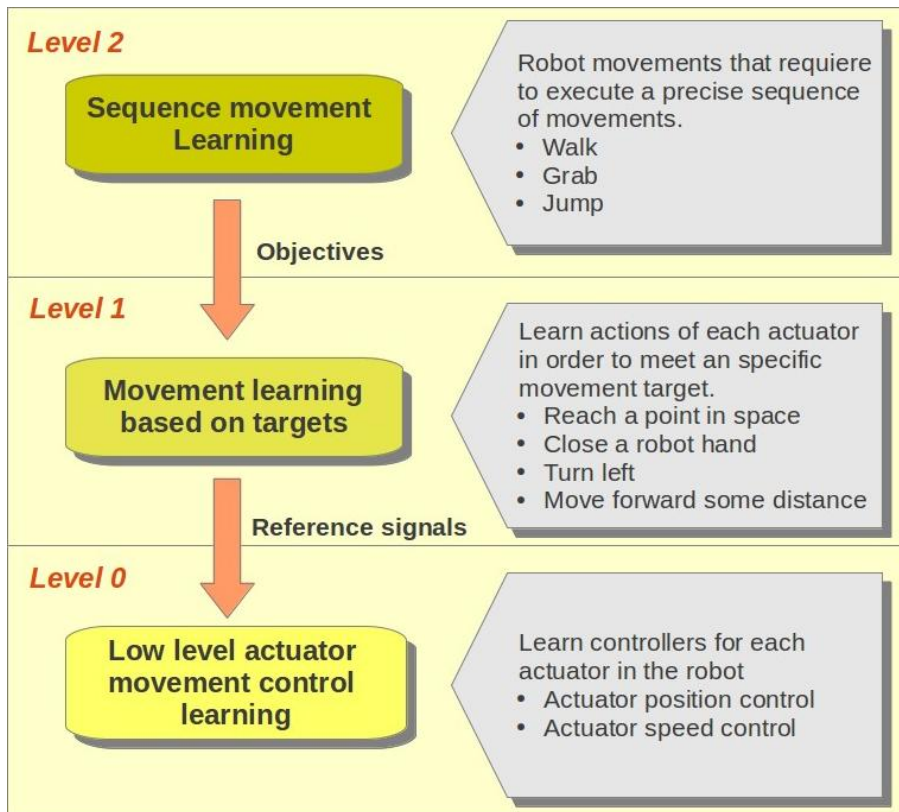


FIGURE 3.1. Subsumption architecture for motion control. Arrows indicate the hierarchic flow of control

- Level 2 - Complex movement learning:** In this layer complex movements of the robot are performed by executing a sequence of basic movements. Basic motions sequence, execution times and targets are carefully configured in order to achieve a robot movement that involves different actuator actions over time, like walk, kick, stand up, etc.

As the architecture is thought to be independent from actuators' configuration, control systems of all layers must be automatically learned. However learning methods design for each one of the layers are currently wide research fields, and even more, combining those methods into a single architecture involves a bigger challenge. This project is mainly focused on the learning mechanism of layer in level 1, but it takes into account a robust architecture design for its easy expansion in a further research. Level 0 is designed for control purposes with easy expandable features, but constraints are added in order to simplify the architecture. Controller



design of this level was showed and tested in Chapter 2. Level 2 of the architecture is left unaddressed and open for further research.

### 3.1.1 Basic movement learning architecture

Instead of programming the robot to make some specific movements in a static environment where dimensions of the robot are known, it is assumed that the environment or the robot configuration may change. Therefore, the robot must find the movements required for every motor on its own. This process is controlled in the level 1 of the architecture which subsumes level 0 in order to directly control actuator motions. The learning process is performed in two stages that involves, finding the correct movements of actuators, and then, learning those movements. Figure 3.2 shows the architecture for basic movement learning.

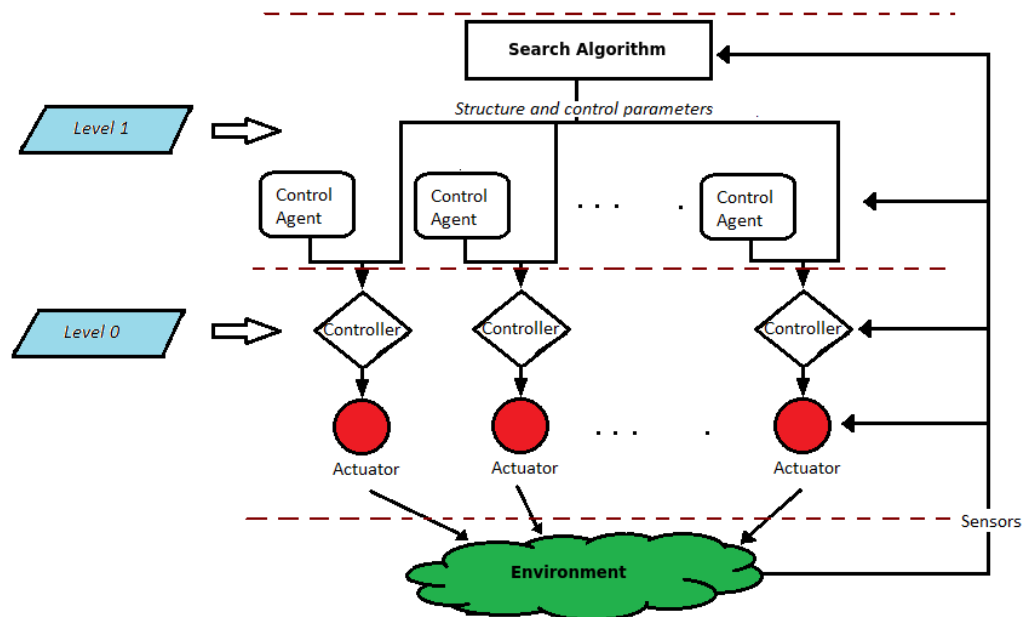


FIGURE 3.2. Subsumption architecture for motion control

In the low level (level 0) of the architecture are the controllers defined in the previous Chapter and the actuators. These controllers try to keep actuators of the robot in the less energy position, but can be altered by superior layers through a reference signal. Actuators movements directly affect the environment which is sensed through robot sensors in order to get a feedback path. Level 1 has two

different stages; in the first one, a search algorithm is used to find the required reference signal for each controller that allows the robot reaching an specific target (as defined in the previous Chapter). Sensor measures provide information about the performance of the robot motion and the accuracy of the approach to the target. The second stage is composed by a set of agents (one per actuator) which learn the movements found by the search algorithm in order to execute them quickly.

Figure 3.3 shows a detailed diagram of the architecture model for a single controller/actuator component. The approach of this model is based on the perceptions of the world that the robot is able to acquire. Both the search algorithm and the control agent use the information directly from the world to generate the appropriate inputs signals of an specific controller/actuator. This way, the controller/actuator component becomes transparent for the learning process. The advantage is that this model can correct accuracy errors in the controller, since it generates the appropriate input signal based on the outputs of the controller/actuator.

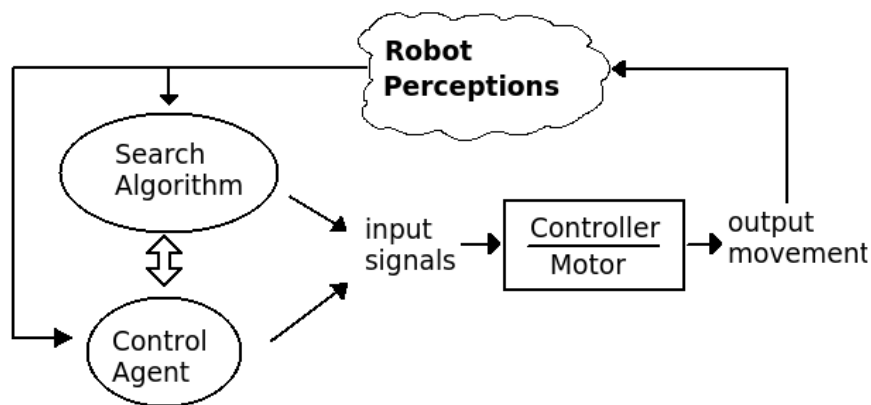


FIGURE 3.3. Detailed model of the architecture for a single controller/actuator

Several targets are generated and evaluated in the control architecture in order to cover the maximum movement area possible. When the system has learned enough examples, search algorithm is not longer needed and agents control motions of the robot, providing fast response. The control algorithm of the architecture is summarized in Table 3.1.

---

**Control Architecture Algorithm**


---

1. *generate*(*tar* = *newTarget*(*x*, *y*))
  2. **while** (*searchAlgorithmFinishCondition*() is false) **do**
  3. *Run SA* = *searchAlgorithm*(*tar*)
  4. **end while**;
  5. *solution* = *getSolutionFound*(*SA*)
  6. **for** each actuator  $A_i$  **do**
  7. *learnMovement*( $A_i$ , *tar*, *solution*)
  8. **end for**;
  9. **go to 1**
- 

TABLE 3.1. Control Architecture Algorithm.

## 3.2 Finding robot's movements

Given the defined robot, environment and targets (see Chapter 2), it is possible to find several solutions depending on the robot's degrees of freedom (DOF). Thus, for the 2 DOF robot arm there are 2 solutions, for the 3 DOF robot arm there are 4 solutions, for the 4 DOF robot arm there are 8 solutions, and so on. Basically, the number of possible solutions  $n_s$  is given by Equation 3.1.

$$n_s = 2^{DOF-1} \quad (3.1)$$

If dimensions of the robot are known within this simulation environment, it is possible to mathematically find solutions that make the arm to reach the target. However, it is supposed that the robot does not know those dimensions and the environment is subject to changes. Therefore, the robot must learn how to perform these movements using a search technique. A genetic algorithm, is chosen for this purpose.

### 3.2.1 Genetic algorithms as a search method

Given the multiple solutions available, GAs are suitable for this problem because they are domain independent and they have great exploratory features. Further-

more, GAs' stochastic nature, allow them to find different solutions on each run, which provides adaptability features to the control system.

Among the genetic algorithms, there are a lot of possible choices that may be implemented to this specific problem. Essentially, what differences a genetic algorithm from another are the individuals codification, the selected operators and the fitness function. As matter of convenience, HAEA algorithm proposed by Gomez [20] is used as a search method in the architecture.

Algorithm outputs are the reference signals for each actuator that make the robot to reach a target (see Figure 3.4). In the simulation environment, this reference signal is traduced into an angle for each motor. The population evaluation process is done directly in the environment by executing the actions of motors for certain amount of time and testing the final movement. For simulation purposes, several targets are generated and evaluated in the control architecture.

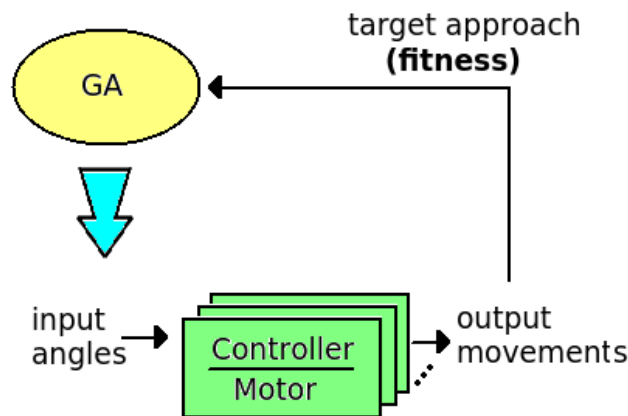


FIGURE 3.4. Genetic algorithm model

### 3.2.2 Genotype and Phenotype

Individuals in HAEA must encode all actuators movements of the robot. This means that the length of the individuals is determined by the quantity of actuators in the robot. Specifically for the simulated robot, phenotype codes the angles of movement that each motor must perform in order to reach the target. These angles were taken

to be absolute from the fixed point of the robot, therefore, initial position of the arm does not affect the result.

A binary codification was chosen to form the genotype. Number of bits of genotype and range of movement of each motor determine resolution of the movement, i.e how accurate the capacity of movements of every motor is. Greater accuracy of movements may be achieved by reducing the motors range of operation and increasing the number of bits. This way, for example for a range of -120 to 120 degrees and a resolution of 8 bits, there are 256 possible movements with a minimum step of 0.93 degrees. Depending on the application, greater resolutions may be needed, but for general movements of the simulated robot arm, 8 bits is good enough. Table 3.2 shows the phenotype and genotype of an instruction for 3 DOF arm robot using a range of -120 to 120 and 8 bits resolution.

Genotype	Phenotype
01110100,10111011,10100110	-10.82, 56, 36.23

TABLE 3.2. Genotype and phenotype of 3 DOF robot arm instruction.

As a result of this codification, for example with a resolution of 8 bits, an individual of the population is represented by strings of 16, 24 and 32 bits for the 2 DOF arm, 3 DOF arm and 4 DOF arm respectively.

### 3.2.3 Fitness function

In order to reach a target and obtain an accurate movement, the robot evaluates its final position respect to the target and the motion performance, after executing a movement during a fixed interval of time. This way, two components form the fitness function; a measure of how close is the arm to the target given by the Euclidean distance ( $d_T$ ) and the total amount of movement ( $T_m$ ) that the robot performs. Finally, the two terms are normalized and linearly combined to form the fitness function. Equation 3.2 shows calculation of total amount of movement, where  $n$  is the number of motors and  $a_i$  is the respective angle for each motor and Equation 3.3 shows the fitness function. Normalization of  $d_T$  and  $T_m$  is performed based on

Equations 3.4 and 3.5, respectively.  $V_m$  indicates the maximum vision distance, which has to be greater or equal to the maximum motion scope of the robot, and  $T_{mmax}$  corresponds to the sum of the maximum angle of movement ( $\gamma_m$ ) of each motor ( $T_{mmax} = \sum_1^n \gamma_m^i$ ).

$$T_m = \sum_{i=1}^n abs(a_i) \quad (3.2)$$

$$fitness = \alpha * d_n + (1 - \alpha) * T_n \quad (3.3)$$

$$d_n = \frac{d_T}{V_m} \quad (3.4)$$

$$T_n = \frac{T_m}{T_{mmax}} \quad (3.5)$$

This fitness function tries to minimize the distance to a target ( $d_T$ ) as well as movements of the robot arm ( $T_m$ ). However, there is a trade-off between these two variables; if the robot moves in order to minimize distance to target position, total movement of the robot is necessarily increased. Conversely, if the robot does not move in order to minimize total movement, distance to target would probably be different from zero (depending on the initial distance to target). This is a multi-objective problem, where finding a balance between these variables is the main goal.

The constant  $\alpha$  where  $0 < \alpha < 1$ , controls which of the variables are more relevant for the optimization process. Variables must be normalized in order to have similar scales in the evaluation process of fitness function.

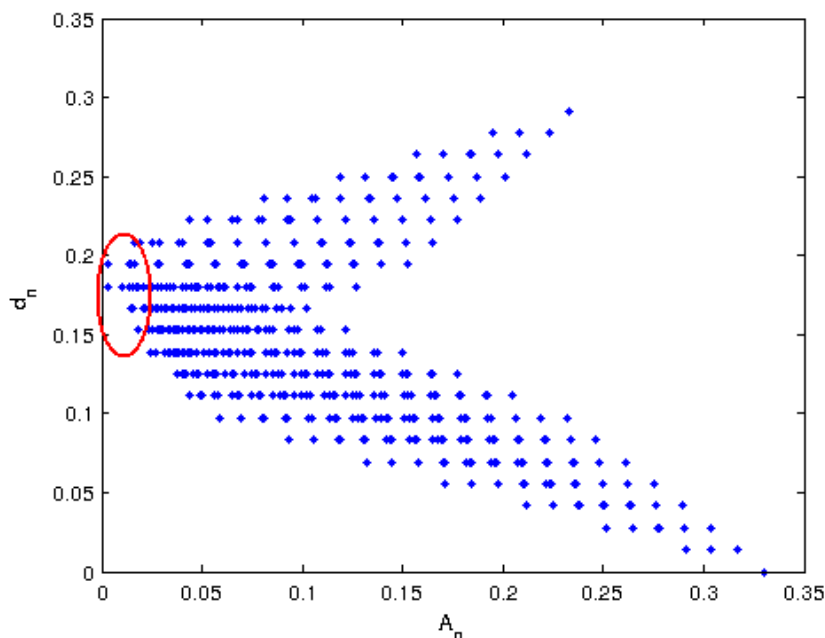


FIGURE 3.5. 3 DOF Robot Arm pareto frontier of fitness function. Total of evaluated points = 512

Relation between variables and control constant values in multi-objective problems are often easier to determine by using a Pareto frontier. Figure 3.5 shows a graphic of Pareto frontier of the fitness function in a 3 DOF robot arm. Since the goal is to reach an specific target, the desirable part of the Pareto front would be points near to 0 in  $Dn$  axis, and of course the ones that has a minor value of  $An$ , as it is showed in the graph.

### 3.2.4 Termination Condition

Termination conditions are defined for the algorithm as a maximum number generations ( $G$ ) and a fitness limit measure ( $f_{min}$ ). In each generation, after population is evaluated, the best fittest individual is compared to  $f_{min}$ . If either fitness limit is reached ( $bestFitness > f_{min}$ ) or the number of generations is reached, the algorithm stops running. Finally, after termination condition is achieved, the best individual (better fitness measure) is chosen as the answer of the problem. Both, the num-

ber of generations and the limit measure are experimentally chosen by running a simulation and analysing convergence of the algorithm.

### 3.2.5 Testing HaEa performance

Basically, the algorithm randomly initializes a population of  $n$  individuals (each individual encoding the angles of movement for all motors in the robot). Then, every individual is evaluated in the environment for a determined amount of time ( $t$ ) and fitness ( $f$ ) is computed. The amount of time was set in order to ensure actuators movements completion. The algorithm uses genetic operators of crossover, mutation and transposition, and selection mechanism is tournament. When termination condition is reached, the best individual is chosen as a solution.

Two different tests were performed using the 3DOF robot arm as a generalized case of robot and an extra test was performed to all robots (2DOF, 3DOF and 4DOF). In these tests, parameters values of the environment and robot characteristics were set as shown in Table 3.3. The first two tests were performed using different genetic algorithm parameters values (maximum fitness ( $f_{min}$ ), population ( $P$ ), number of generations  $G$  and number of bits per parameter ( $B_xP$ )), in order to analyse its behavior and determine an appropriate set of parameters values. An extra test was performed with the purpose of showing that the algorithm is able to find a solution for all three robots using an specific set of parameters values.

	<b>Value</b>
<b>Simulation time</b> $t(s)$	4
<b>Step time</b> $\Delta t(s)$	0.01
<b>ODE solver tolerance</b>	0.0001
<b>Simulated robot</b>	3 DOF Robot Arm
<b>Robot links length</b> $s(mu)^1$	$s_1 = 80, s_2 = 80, s_3 = 80$
<b>Motors movement range (Degrees)</b>	$m_1 = [-120, 120], m_2 = [-120, 120]$ $m_3 = [-120, 120]$
<b>Maximum vision distance</b> $V_m(mu)$	<i>Robot's max scope</i> : $\sum_{i=1}^3 s_i$
<b>Motor model</b> $H(s)$	$\frac{4}{s^2+12s+4}$

TABLE 3.3. Environment and robot simulation parameters values for 3DOF robot arm.



In the first simulation, parameters were set to values that allow the author to have a general idea of the behavior of the GA. As the robot purpose is to reach the target (minimize distance to target) the parameter  $\alpha$  (which defines a priority to the distance variable) in the fitness function was set to  $\alpha = 0.9$ . Bits per parameter was set to  $B_x P = 8$ , population was set to  $P = 20$  and the number of generations was set to  $G = 100$ . In this simulation the fitness limit condition was not taken into account and only the maximum number of generation is used for stopping the algorithm. This was done in order to determine the convergence region of the GA and the maximum fitness achieved by the algorithm (as HAEA tries to maximize fitness function, fitness in Equation 3.2 is multiplied by -1). Figure 3.6 and 3.7 shows the mean of population on each generation over 30 runs of HAEA for a single random target.

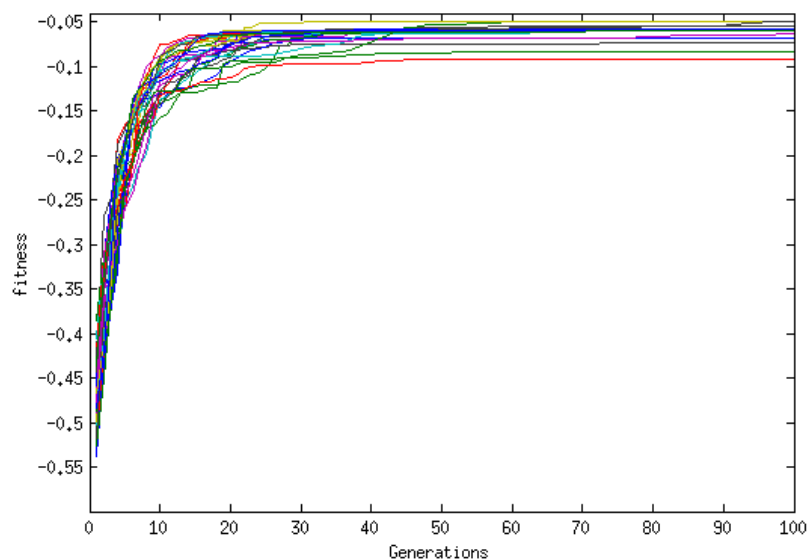


FIGURE 3.6. Fitness mean of population ( $P = 20$ ) over 100 generations of a single random target

As the graph of Figure 3.6 shows, fitness convergence values are between ranges of  $-0.1 < fitness < -0.05$ , which in terms of distance are equivalent to  $6 \pm 6mu < d < 15 \pm 10mu$  depending on the quantity of movement (specifically for the 3 DOF

<sup>1</sup>*mu* are *measure units* which in the context of simulations performed in this work correspond to pixels, however it may represent any unit of measure

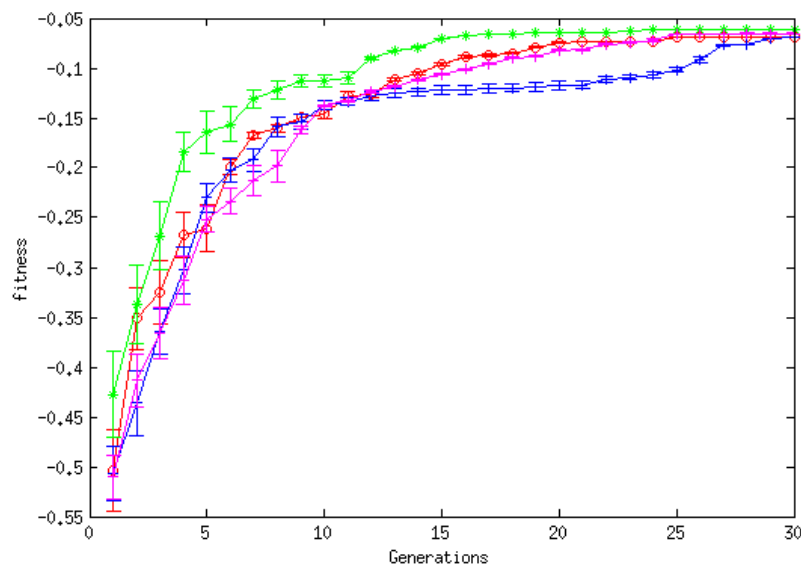


FIGURE 3.7. Fitness mean and variance of HAEA over 30 generations for 4 different targets

robot arm). Considering the dimension of the robot, the obtained distances are relatively low, which shows that HAEA is capable of finding a good solution with the given parameters values. In addition to this, convergence of the algorithm is achieved in short period of time (around 20 to 30 generations). Figure 3.7 shows a zoom of the previous graph by showing only four runs over 30 generations with their respective population variances.

In the second test, six simulations were performed using populations ( $P$ ) of 5, 10 and 15 individuals and number of bits per parameter ( $B_xP$ ) of 5 and 8. In these simulations the number of generations  $G$  was set to  $G = 20$ . Additionally, the fitness limit termination condition is included in the GA and set to  $f_{min} = -0.08$ , since after firsts tests, that value of fitness showed a good approaching to the target (around  $5mu$  of distance). This last condition causes the algorithm to finish early, since it reaches a minimum distance to the target, before reaching the max number of generations. In this simulation, reported values are the mean and variance of the fitness of the best individuals ( $f_a$ ), the number of generations reached ( $G_a$ ) and the distance to the target ( $DT_a$ ). Table 3.4 shows the results as the average of 30 runs of the algorithm for 2 random generated targets.

	Parameters		Results		
	$P$	$B_xP$	$f_a$	$DT_a(mu)$	$G_a$
<b>Target 1</b>	5	8	$-0.0906 \pm 0.0305$	$16.6504 \pm 14.7148$	$13.4333 \pm 7.555$
<b>Target 1</b>	5	5	$-0.112 \pm 0.0561$	$29.7513 \pm 27.293$	$17.566 \pm 9.852$
<b>Target 1</b>	10	8	$-0.0658 \pm 0.0137$	$14.001 \pm 5.3883$	$6.1333 \pm 5.888$
<b>Target 1</b>	10	5	$-0.06751 \pm 0.021$	$20.2586 \pm 8.1419$	$9.1666 \pm 6.4703$
<b>Target 1</b>	15	8	$-0.06559 \pm 0.01037$	$15.709 \pm 7.349$	$5.800 \pm 4.8947$
<b>Target 1</b>	15	5	$-0.06512 \pm 0.02088$	$16.108 \pm 9.588$	$6.16666 \pm 5.7179$
<b>Target 2</b>	5	8	$-0.0816 \pm 0.0305$	$14.1871 \pm 8.2612$	$13.4333 \pm 5.992$
<b>Target 2</b>	5	5	$-0.0721 \pm 0.02201$	$18.903 \pm 9.852$	$11.8666 \pm 5.9754$
<b>Target 2</b>	10	8	$-0.0733 \pm 0.0104$	$13.2405 \pm 6.3165$	$9.4 \pm 7.0934$
<b>Target 2</b>	10	5	$-0.06462 \pm 0.01236$	$12.8928 \pm 5.9057$	$6.733 \pm 6.638$
<b>Target 2</b>	15	8	$-0.0703 \pm 0.0084$	$14.7522 \pm 5.1284$	$4.1666 \pm 5.2396$
<b>Target 2</b>	15	5	$-0.07155 \pm 0.0089$	$11.5325 \pm 5.6823$	$7.001 \pm 4.5334$

TABLE 3.4. Average of 30 runs of fitness, number of generations and final distance for 2 random targets in the simulation of 3DOF robot arm.

Results show that there is no big difference among the distance and fitness obtained for the given parameters values of the two targets. With exception of two or three values most of the distances and fitnesses obtained ranges from 10 to 15  $mu$  and from -0.075 to -0.055, respectively. This suggest that any of tested values for  $P$  and  $B_xP$  can be used in order to obtain an appropriate result. Nevertheless, it must be taken into account that the fact of using lower resolutions and small populations reduces the search space and hence, the computation time.

Regarding to the number of generations, the population parameter value affects in some way the number of generations reached. Larger populations reach the fitness limit faster than small populations, therefore the number of generations needed to find a solution is reduced. Based on this, the parameters values of  $P = 12$  and  $B_xP = 5$  provide a balance between population size and computation time.

The last test was performed to all robot arms; 2DOF, 3DOF and 4DOF robot arms. Parameters values were set to  $G = 20$ ,  $P = 12$ ,  $f_{min} = -0.08$ ,  $\alpha = 0.9$  and  $B_xP = 5$  and a total of 30 random generated targets were used for the test. Figures 3.8, 3.9 and 3.10 show the average over 30 runs of the best individual fitness ( $f$ ), distance to target ( $DT$ ) and number of generations reached ( $G$ ) for 30 random targets.

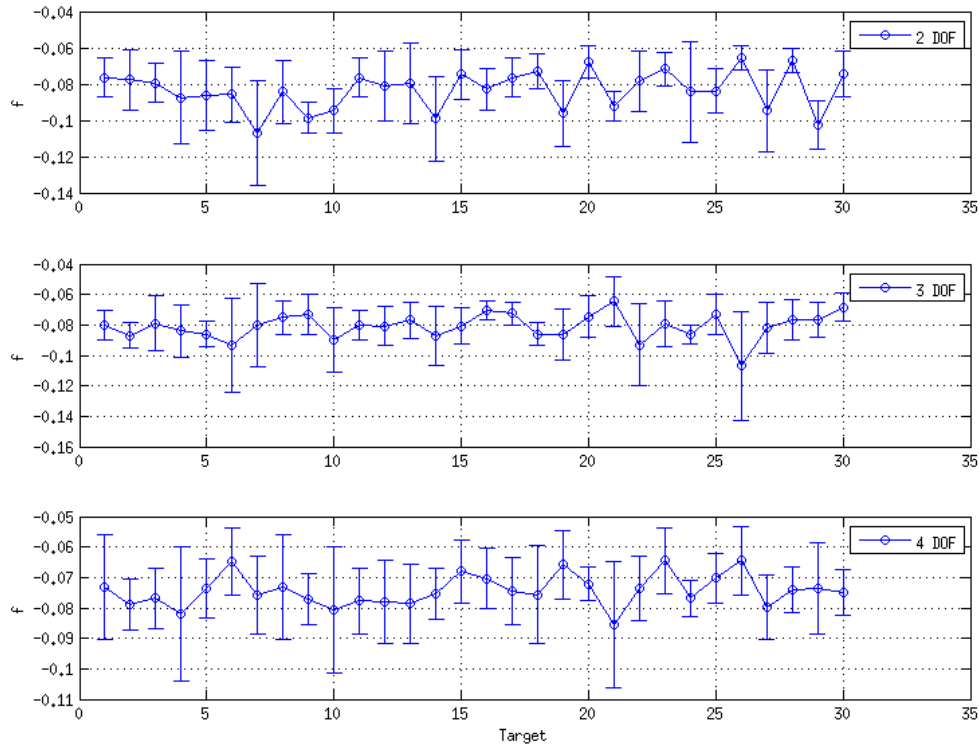


FIGURE 3.8. Fitness mean and variance over 30 runs of the best individual for 30 random generated targets. Simulation performed for the 2DOF, 3DOF and 4DOF robot arms

The obtained results showed that the proposed configuration used in the three cases (2DOF, 3DOF and 4DOF) are appropriate to accomplish the approaching to the targets. Even when standard deviations tend to be high in some targets the average fitness tends to be around  $-0.08$  which is the expected result (as it is the max fitness limit). In the worst case, that is the 4 DOF robot arm, average of all targets does not exceed distances greater than  $30mu$ , which is very low, considering the dimensions of the robot ( $320mu$ ). Also, the average of generations in all cases tend to be less than 15 for almost all targets which proves that an expected solution is found quickly.

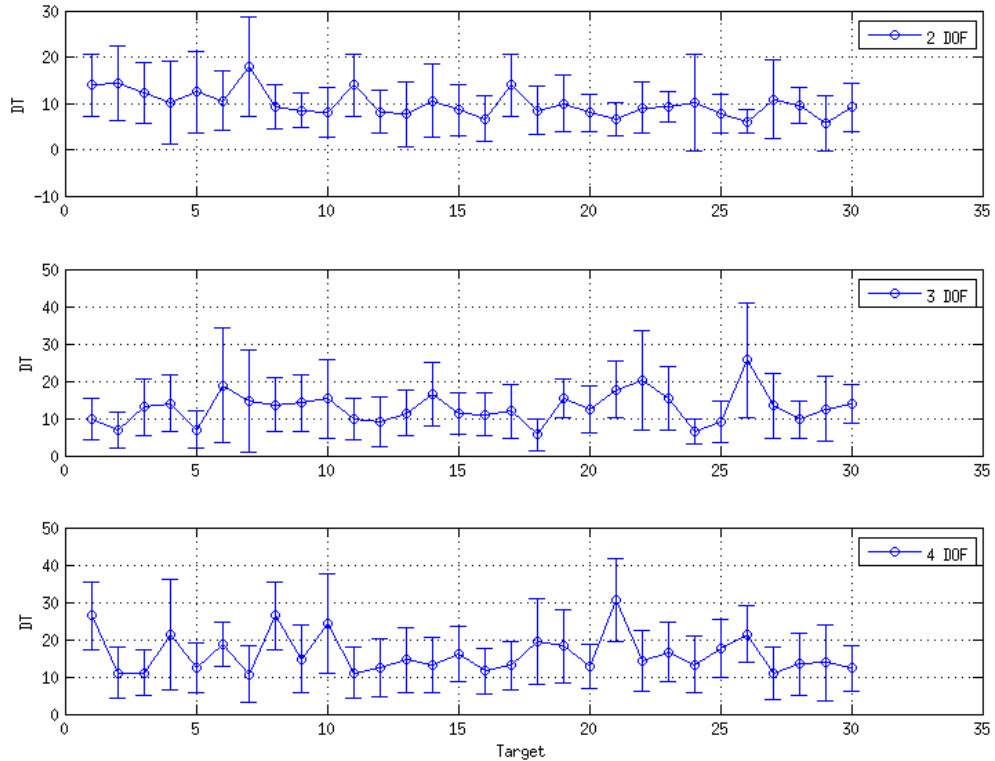


FIGURE 3.9. Distance mean and variance over 30 runs of the best individual for 30 random generated targets. Simulation performed for the 2DOF, 3DOF and 4DOF robot arms

### 3.3 Learning robot's movements

The purpose of the learning system is to memorize movements found by the search algorithm and execute these actions rapidly in order to improve robot's execution of motions. While the search algorithm uses an exhaustive process to determine which are the best movements for an specific target, the learning system must act as re-programmable reactive system, which responds almost immediately to any input of the environment. This way this system takes actions based on perceptions of the environment, but additionally, it can be subsumed in order to act differently or learn new movements.

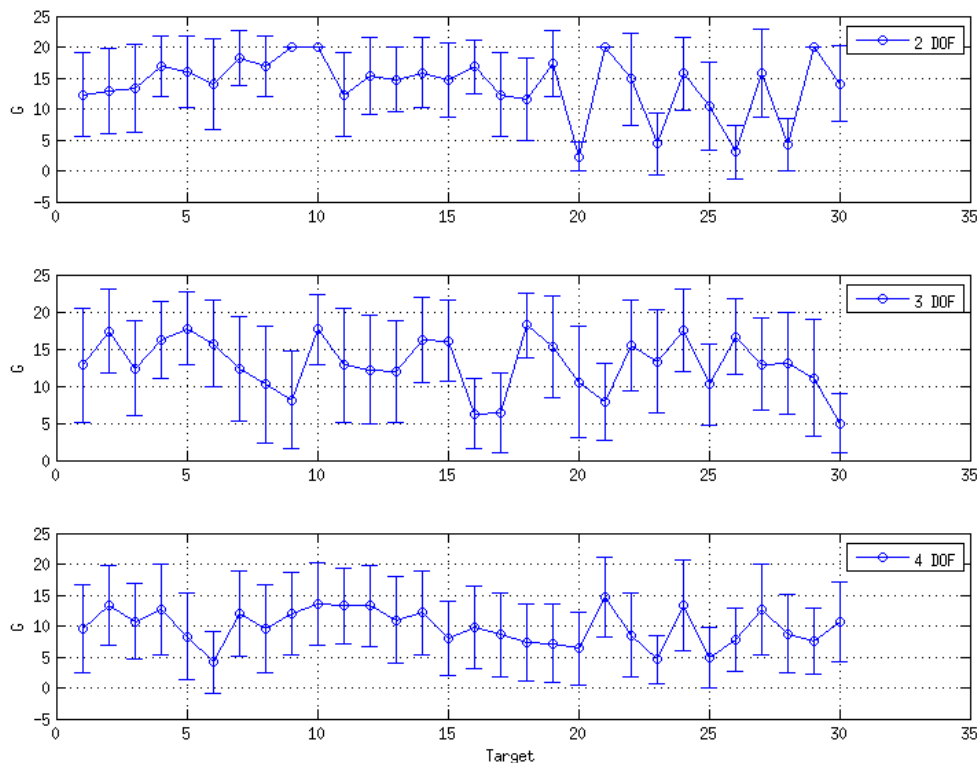


FIGURE 3.10. Generations mean and variance over 30 runs of the best individual for 30 random generated targets. Simulation performed for the 2DOF, 3DOF and 4DOF robot arms

### 3.3.1 Neural Networks for instruction approximation

Neural Networks (NNs) not only can store/represent abstract information in form of weighted sums, but also they are known to be very good pattern associators[4]. Furthermore, due to NNs fast calculation, they act almost as mapping functions where determined input sample, triggers an specific output pattern [28].

NNs are used in this work to represent agents in the proposed architecture. In this way, NNs are subsumed by the search algorithm in order to find a valid movement, and then, they are trained with that found movement. After some limited number of training examples, the search algorithm is no longer needed and the set of NNs take control of the robot's movements. NNs not only learn specified movements by the search algorithm, but also approximate values between samples; i.e if the NN have learned the movements required to reach an specific target, and then, a new

target came near to this, a similar solution to the first target movement is expected. In this sense, there must be a limited number of targets in different positions to be learned which will cover all space of solution.

Multilayer perceptron (MP), that has been used widely for function approximation, pattern classification and recognition [49], was used in this architecture<sup>2</sup>. The MP was configured with a radial basis activation function and linear output in order to get interpolation features [50]. Activation function is Gaussian and back-propagation algorithm was used to train samples. Figure 3.11 shows a general diagram of the neural network.

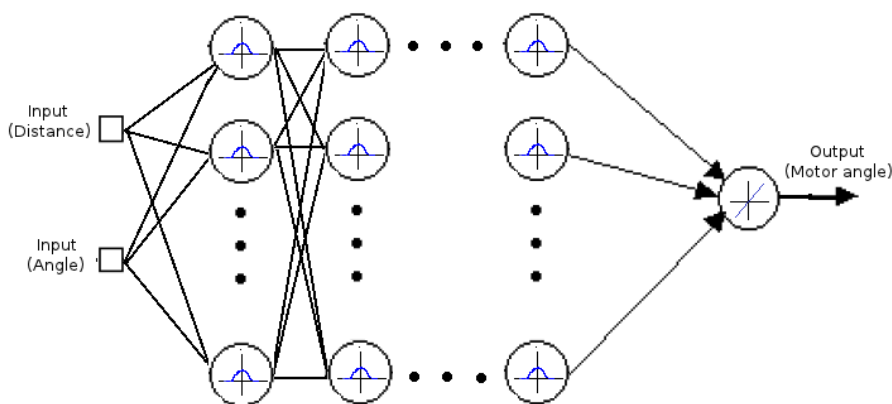


FIGURE 3.11. MP Neural Network with radial activation function and linear output. Inputs: distance and angle to the target, Output: angle of movement of the motor

The idea, is that every motor has its own NN which learns specific movements based on targets. The targets, which correspond to the inputs of each NN, are given in terms of distance and angle from the robot position (fixed point). The output corresponds to the angle of movement of each motor. In the learning process, samples of the motions found by HAEA are trained using back-propagation algorithm. The detailed training process is resumed below:

1. Generate a empty training set for each Neural Network (one per motor) in the robot.

<sup>2</sup>The Neural Networks were implemented into the simulator using Neuroph libraries (Java Neural Network Framework) provided as open source by Sourceforge[3]

2. Generate a new target to train (it can be manually or random).
3. Use HAEA to find movements that each motor must perform in order to reach that specific target.
4. Once a valid motion is found by HAEA, create a new training sample for each NN based on the angle and distance to the target (input) and the angle of movement for each motor (output).
5. For each NN, add the new training sample to the set of training samples of each one and start back-propagation algorithm.
6. Stop back-propagation if the total network error is less than a maximum network error ( $e_{NNmax}$ ) or a maximum iterations number ( $I_{NN}$ ) is reached.
7. If the maximum iterations number is reached on any NN, remove that training sample of all training sets, re-train the NNs and start from 3 again (in order to retry training that example).
8. If total network error is less than the maximum network error go to 2, until the number of training samples ( $TS_{size}$ ) (size of the training set) be reached.

Considering the results of the first tests of the NNs using this process, it was noticed that after training some number of samples (targets), the NN was no longer capable of reducing the network error ( $e_{NN}$ ), and given the defined loop, it kept iterating over that example. Bigger NNs were able to train more samples, but there was always a limit where the system collapses in an endless loop.

In order to avoid this unwanted behavior, the training process was modified by adding a limit condition in the number of times that removed training samples are re-trained. This condition is included by modifying the process in the following way: instead of creating a loop to re-train a removed target, the target is skipped and stored in a *re-training queue*. When all samples ( $TS_{size}$ ) have been evaluated, samples of the *re-training queue* are given to the algorithm until it empties. The quantity of samples added to the *re-training queue* is counted and limited to a



maximum value ( $TS_{RQ}$ ). Taking this into account some steps in the learning process are changed as follows:

7. If the maximum iterations number is reached on any NN, remove that training sample of all training sets, re-train the NNs and store that target in the *re-training queue*. Targets are only added to the queue if the quantity of samples added are less than the maximum value of added targets ( $TS_{RQ}$ ).
8. If total network error is less than the maximum network error go to 2, until the size of the training set ( $TS_{size}$ ) be reached.
9. If the number of training samples ( $TS_{size}$ ) is reached and there are targets in the *re-training queue*, pop one target of the queue and go to 3 until the queue empties.

### 3.3.2 Neural networks performance

Tests were performed in order to define what structure and parameters values to use for the neural networks and the capacity of those networks to be trained with certain number of samples. First experiments were performed with a single hidden layer <sup>3</sup> for each actuator, but results showed that the robot was not able to learn more than a few examples and hence, the learned samples were unable to cover the space of movement (even when using a lot of neurons in the hidden layer).

Then, a second hidden layer was added to the neural network and tests showed that the robot was able to learn enough samples to cover most of the movement space of the robot. However, it was noticed that quantity of samples that an specific robot was able to learn depend on the size of the neural network structure. A neural network structure of two hidden layers of 4 neurons each showed to be able of learning around 30 samples, while a neural network structure with 10 neurons for layer was able of learning around 80 samples. Table 3.5 shows the average of samples trained

---

<sup>3</sup>The neural network structure is composed by an input layer (which has two neurons for the angle and distance from the robot to the target), the hidden layers and an output layer (one neuron with linear activation function)

using hidden layers of [4,4],[6,6],[8,8] and [10,10] neurons. Results are the average over 30 runs of the 3DOF robot arm for 100 random training targets. Parameters values used for HAEA and the neural networks are summarized in Table 3.6 and simulation environment parameters values were set equal to the previous simulations (see Table 3.3).

$HL$	$TS_{Total}$
[4, 4]	$31.3426 \pm 3.6801$
[6, 6]	$43.4502 \pm 3.7364$
[8, 8]	$65.4171 \pm 2.865$
[10, 10]	$78.768 \pm 3.3216$

TABLE 3.5. Average over 30 runs of total trained samples using 100 random targets in the 3DOF robot arm

HaEa parameters		NN parameters	
<b>Population (<math>P</math>)</b>	12	<b>Maximum net error (<math>e_{NN}</math>)</b>	0.0001
<b>Generations (<math>G</math>)</b>	20	<b>learning rate (<math>l_R</math>)</b>	0.02
<b>Fitness max limit (<math>f_{min}</math>)</b>	-0.08	<b>Maximum iterations (<math>I_{NN}</math>)</b>	25000
<b>Resolution (<math>B_{xp}</math>)</b>	5	<b>Activation function (<math>A_f</math>)</b>	Gaussian( $\sigma = 0.5$ )
<b>Fitness constant (<math>\alpha</math>)</b>	0.9	<b>Max samples training queue (<math>TS_{RQ}</math>)</b>	$\frac{5}{4} * TS_{size}$

TABLE 3.6. HAEA and NNs parameters values.

The number of hidden neurons on the neural networks may be increased in order to train as many samples as possible. However, the purpose is not to train a lot samples but just the necessary to cover all the movement space of the robot. In fact, the smaller the net (which uses fewer training samples), the better and faster the learning system.

Since robots require more training examples as the number of DOFs is increased, different neural network structures and number training samples were used for each robot. This way, neural networks of [6,6],[8,8] and [10,10] hidden neurons and training sets of 60,80 and 100 were used for testing the 2DOF, 3DOF and 4DOF robot arms, respectively. Here, given the fact that fitness in HAEA measures performance of movement and neural networks are trained to perform such movements, this fitness is used to measure performance of the neural networks based control system as

well. However, final distance to targets after performing the movement is presented as well since it is still a good and clearer criterium (as the goal is to reach the target).

Parameters values of the NN and GA were set just as before (see Table 3.6) and reported results are the average over 30 runs of the median of distance ( $DT_m$ ) and fitness ( $f_m$ ) obtained of all testing samples. The median is an estimator that is more robust than average. Table 3.7 summarized the results for 200 random targets used to test.

	$HL$	$TS$	$DT_m$	$f_m$	$TS_{Total}$
<b>2DOF</b>	[6, 6]	60	$42.98 \pm 12.4192$	$-0.2911 \pm 0.07106$	$38.6 \pm 2.891$
<b>3DOF</b>	[8, 8]	80	$132.5243 \pm 22.6851$	$-0.30838 \pm 0.06$	$61.4 \pm 1.825$
<b>4DOF</b>	[10, 10]	100	$151.5981 \pm 6.206$	$-0.28374 \pm 0.0119$	$78.1 \pm 3.6514$

TABLE 3.7. Average over 30 runs of 200 random targets for 2 DOF, 3 DOF and 4 DOF robot arms. Reported values are de medians of distance and fitness obtained of the targets

Results obtained show that NNs are able to learn the movements generated by HAEA and approximate some of the targets that were not trained. Also, data showed that the number of trained samples is increased for larger neural network structures. This way, the NN is able to cover more space as the robot structure gets bigger.

However, the resulting trained neural networks are still far from the expected learning capability. Distance error obtained is too big and it becomes worst as the number of DOFs are incremented. This may due to the fact that the NN structure is not the appropriate for the given robot since it does not have into account relations between actuators. The number of possible solutions is incremented with the number of DOFs and using the same NN for all actuators restricts the possibilities of learning different solutions for a given target.

### 3.3.3 Improving Neural Network Structure

Depending on the robot construction (arm, leg, wheels, etc.) there is always dependence on the movements that each actuator performs. In particular, the simulated

robot arm used in this work has a dependence between subsequent actuators. In this way, movements of the first actuator (fixed point) affects movements of other actuators, and movements of second actuator affects movements of third and posterior actuators, and so on.

It is possible to include this dependence relation by adding extra inputs to the NN of each actuator. This way, taking 2 DOF robot arm as an example, it has an independent actuator (the one that is located in the fixed point) and a dependent actuator. The neural network structure of the first one remains unchanged since there are no dependencies in movement of other actuators. On the contrary, the second actuator neural network structure is changed by adding an extra input, which corresponds to the output of the first actuator. This process can be repeated for the cases of 3 DOF, 4 DOF arm and other robot structures as well. Figure 3.12 shows a diagram of the neural network input/output structures for each actuator in the 3 DOF robot arm case.

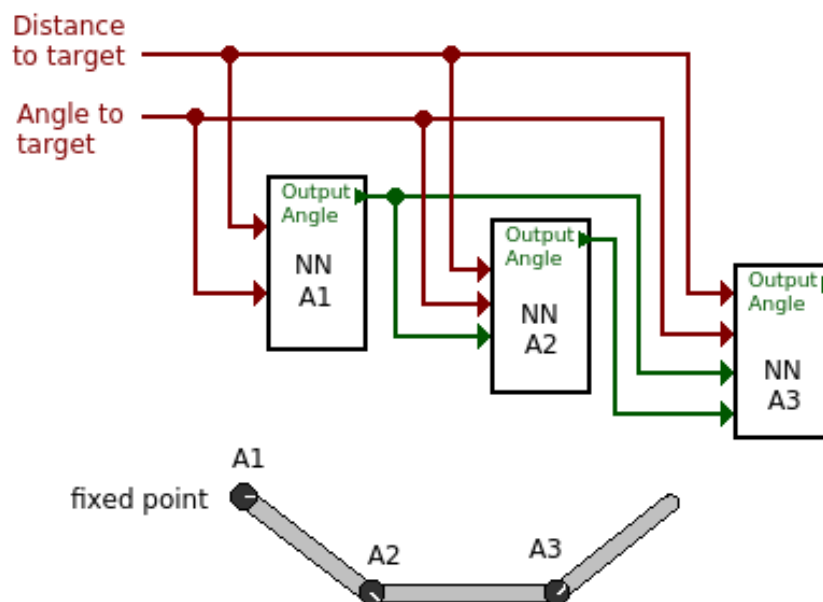


FIGURE 3.12. Neural network structure with actuator movement dependence

A single test was performed to this new structure in order to compare its performance with the previous structure. Parameters values were set equally to the

previous simulation (see Table 3.6) and simulations were performed to the 2 DOF, 3 DOF and 4 DOF robot arms as well. Table 3.8 summarizes the results obtained.

	$HL$	$TS$	$DT_m$	$f_m$	$TS_{Total}$
<b>2DOF</b>	[6, 6]	60	$24.1275 \pm 4.3804$	$-0.1296 \pm 0.0145$	$30.72 \pm 3.2676$
<b>3DOF</b>	[8, 8]	80	$31.3889 \pm 5.8497$	$-0.1036 \pm 0.01199$	$38.2857 \pm 5.8629$
<b>4DOF</b>	[10, 10]	100	$54.8220 \pm 13.8892$	$-0.1266 \pm 0.023$	$55.5454 \pm 9.2451$

TABLE 3.8. Average over 30 runs of 200 random targets for 2 DOF, 3 DOF and 4 DOF robot arms. Reported values are the medians of distance and fitness obtained of the targets after training with 60 samples (targets).

There is a clear improvement on the results obtained respect to the first neural network structure. The median distance error is drastically reduced and lower standard deviations are obtained. Here the robot movements are well approximated to targets, even when training data is random and we are not sure of covering all movement space. Similarly to previous simulation there is still a relation between the number of DOFs and the distance obtained. Less DOFs get better approximations and fitness measures.

In addition, it seems that the training data is carefully selected since in this simulation, less training samples are trained, compared to the previous simulation (the one that does not have into account the relation between subsequent DOFs).

### 3.3.4 Improving the learning process

During the training process neural networks are improving its response for each new target trained. Now, if a new target is close to a previous trained target, it is expected that movements that reach the new target to be similar to movements of the trained target. Then as the neural network (NN) have learned this movements, it is able to feedback information to the genetic algorithm (GA). Specifically, outputs of the NN can initialize population of GA in order to improve not only the convergence time of the GA, but also the training process of neural networks. As a result the training time is drastically reduced, optimizing the whole process.

For this purpose, a measure computing how close a target is from another, is added to the architecture. This value is computed as the distance between targets and it is assumed that the robot is able to measure it. Now, in order to determine if a target is near to another or not, a threshold  $T_H$  measure is added to each learned target. Given the two dimensional environment, this threshold corresponds to a circle (see Figure 3.13) in which new targets inside of it would be considered to be nearby.

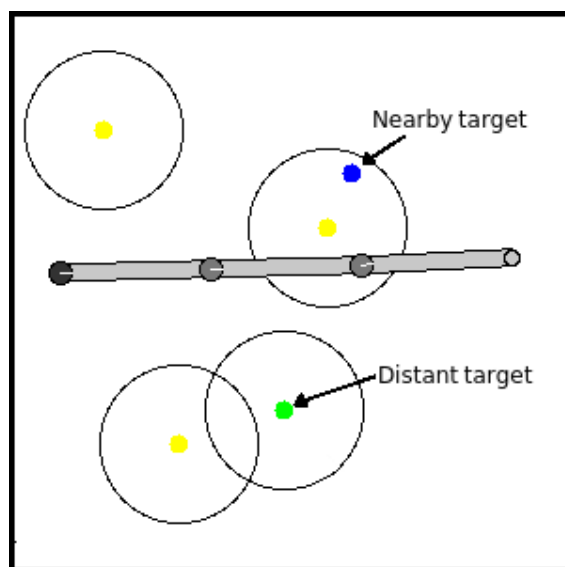


FIGURE 3.13. Threshold measure

In addition, a target classified as “nearby”, must be near enough to ensure that population initialization process really speed things up for HAEA. If the threshold measure is too big, HaEa may be initialized with very different individuals from the ones needed by the current target and therefore the convergence of the algorithm would become slower. Taking this into account, the threshold is not set to a specific value, but it is adjusted dynamically depending on the results of the initialization process. It is assumed that if the initialization process is successful, the algorithm must converge very quickly, around 5 generations. Hence, after initializing the population of HAEA, if the algorithm reaches a portion  $G_p = G/D$  of the maximum number of generations  $G$  then the threshold is decreased. Conversely, the threshold is increased when new targets are not near to any of the current trained targets.

Now, if population is initialized and fitness limit  $f_{min}$  is reached instead, threshold remains the same. Finally, the threshold is increased or decreased respect to the nearest target distance  $d_{NT}$  from the evaluated target according to Equation 3.6. The improved control architecture algorithm is summarized in Table 3.9.

$$T_H = T_H \pm \beta * d_{NT} \quad (3.6)$$

---

**Motion Control Architecture Algorithm Improved**

---

1. *generate*(*tar* = *newTarget*(*x*, *y*))
  2. *closestTar* = *getClosestTarget*(*NNTrainedTargets*[ ], *tar*)
  3. **if** (*distance*(*closestTar*, *tar*) < *currentThreshold*)
  4. *motorMoves* = *runNN*(*tar*)
  5. *initializeHAEAWithMotions*(*motorMoves*)
  6. *newMotorMoves* = *runHAEA*(*tar*)
  7. **if** (*HAEAreachedGenerationsPortion*( $G_p$ ) is true)
  8. *decreaseThreshold*(*currentThreshold*, *distance*(*closestTar*, *tar*))
  9. *initializeHAEARandomly*()
  10. *newMotorMoves* = *runHAEA*(*tar*)
  11. **end if**;
  12. **else**
  13. *increaseThreshold*(*currentThreshold*, *distance*(*closestTar*, *tar*))
  14. *initializeHAEARandomly*()
  15. *newMotorMoves* = *runHAEA*(*tar*)
  16. **end if**;
  17. **for** each actuator  $A_i$  **do**
  18. *learnMovement*( $A_i$ , *tar*, *newMotorMoves*)
  19. **end for**;
  20. **go to 1**
- 

TABLE 3.9. Improved algorithm of the control architecture with feedback path of NN response for initializing HAEA population

Here tests of 2-DOF, 3-DOF and 4-DOF robots were performed in order to compare the results with previous simulations. In these tests, all parameters values were set equal to previous experiments (see Tables 3.6 and 3.3), and the proportion is set to  $D = 4$ . Since number of generations is  $G = 20$ , it is expected that the genetic algorithm converges quickly,  $G_p = 5$ . At the begging, the threshold is set

randomly to a value between (10,100)  $\mu u$ . Table 3.10 resumes the results obtained after 30 runs.

	$HL$	$TS$	$G$	$DT_m$	$f_m$	$TS_{Total}$
<b>2DOF</b>	[6, 6]	60	$3.656 \pm 0.907$	$13.284 \pm 1.905$	$-0.797 \pm 0.007$	$35.909 \pm 3.910$
<b>3DOF</b>	[8, 8]	80	$3.086 \pm 0.542$	$22.708 \pm 2.185$	$-0.087 \pm 0.008$	$43 \pm 5.09$
<b>4DOF</b>	[10, 10]	100	$3.061 \pm 0.0455$	$42.089 \pm 14.451$	$-0.104 \pm 0.0317$	$59.333 \pm 6.875$

TABLE 3.10. Average over 30 runs of 200 random targets for 2 DOF, 3 DOF and 4 DOF robot arms. Reported values are de medians of distance  $DT_m$  and fitness  $f_m$

Clearly, the population initialization really improves the convergence of the genetic algorithm. While convergence was around 13 generations for the algorithm with no initialization, here, an average of 3 generations for all cases was obtained. Some improvement of distance error and fitness was achieved as well.

Regarding threshold parameter, it tends to converge to values around 10. Figures 3.14, 3.15 and 3.16 show graphs of the threshold value for a determined number of targets (depending of the robot). Table 3.11 shows a sequence of threshold adjustment for the 2dof robot arm.

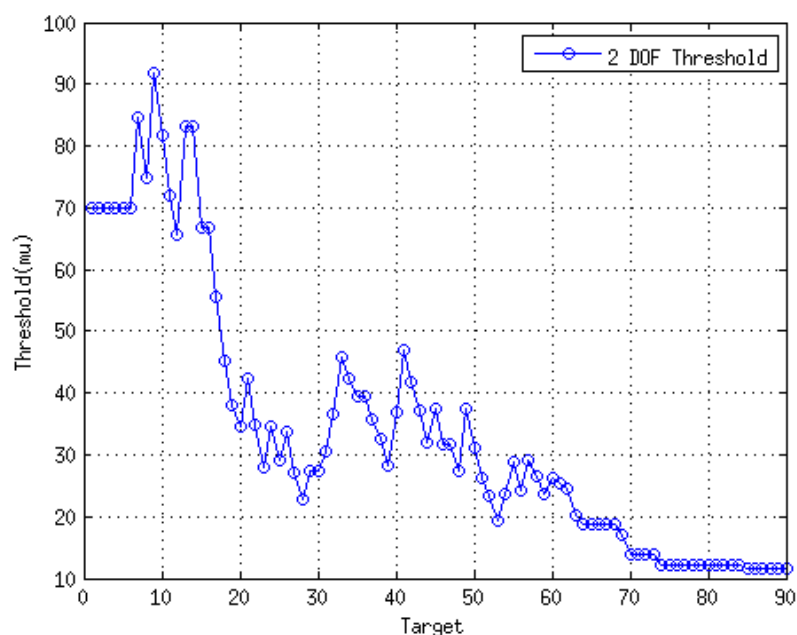


FIGURE 3.14. Threshold value over 80 targets for the 2 DOF robot arm



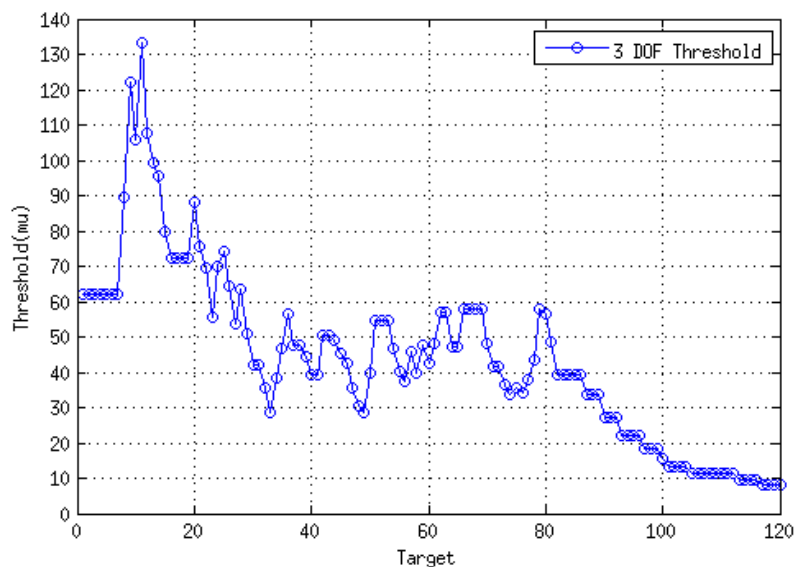


FIGURE 3.15. Threshold value over 120 targets for the 3 DOF robot arm

### 3.4 Result analysis

As can be seen, previous simulations showed that the proposed control architecture is able to find and learn some movements based on targets by combining two different process: a genetic algorithm and a set of neural networks. In one hand, the genetic algorithm HAEA is able to successfully find a set of movements that allow the robot to reach an specific target. Also, the genetic algorithm converges quickly and results present very low distance errors. On the other hand, the neural network is able to learn the movements found by the genetic algorithm and approximate targets that had not been trained. However, results showed that some fundamental aspects about the number of samples trained and the neural network structure must be taken into account in order to obtain good approximations of no learnt targets.

In this sense, the number of samples to train the NNs must be incremented as the number of DOFs grows in order to cover the space of movement. Also, it seems that, for a determined NN structure (size) there is some limit in the number of samples that the NN is able to learn (memorize) and therefore, the neural network structure (size), must be incremented as well in order to train more samples. Additionally,

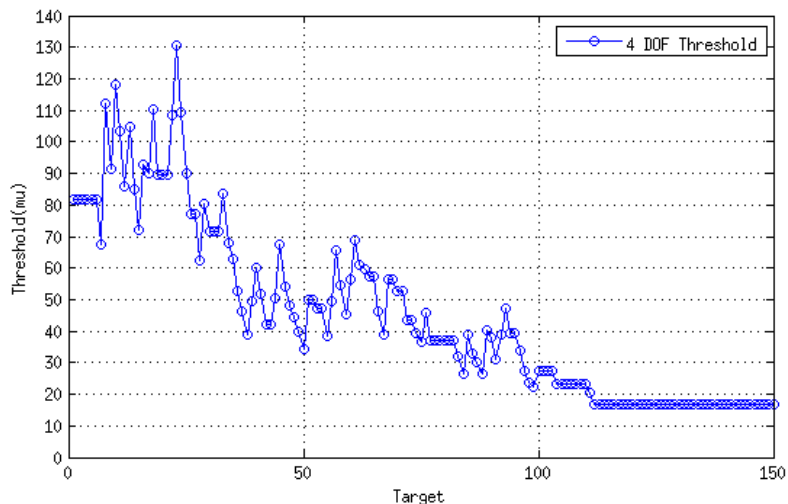


FIGURE 3.16. Threshold value over 150 targets for the 4 DOF robot arm

DOFs relations must be included in the neural network structure in order obtain more precise results. However, not only adding this dependence relation but also changing the neural network structure for each robot, implies that some previous knowledge about the robot, that must be added externally. This causes a reduction of the generalization capability of the control architecture. Nevertheless, the proposed method does not change the hidden neurons structure drastically and is possible to include this modification dynamically in a further research.

Respect to the population initialization process, it really speed up the convergence of the genetic algorithm and improves the training process as well. With this last improvement the computing time is reduced almost in a 50 percent and final results are more approximate to the targets.

It is necessary to take into consideration that the learning process was always performed using random targets, therefore, it may be possible that the targets were not covering all the space of movement, which increase the error obtained for the tested targets. Also, since the number of possible solutions is increased exponentially depending on the DOFs of the robot arm, the genetic algorithm may find different movements for similar targets, yielding an incoherence in the learned movements. This may explain the fact of obtaining higher errors when incrementing the DOFs

of the robot, plus the fact of not knowing the necessary number of training targets for an specific robot.

Using a re-training queue attempts to solve this problem by giving a second chance to the GA to find similar movements to a previous similar target. In the same way, the population initialization process reduce even more the possible solutions of an specific target, since it uses the movements learned of previous targets to find a similar movement. Experiments showed that these methods improved the results obtained of robots tested and the architecture is able to move the robots successfully in almost all the space. However, all experiments were performed with a given number of training samples (targets) which was determined experimentally. In this sense, in order to make the architecture applicable for any environment some method must be added to determine when training must stop or re-start (when the movement space has been covered).

Finally, it is clear that the threshold value converges to an specific value for a given environment. At the end of the training process each training target with its respective threshold defines an area in which the robot is able to approximate the movements properly. This way, this threshold may be seen as a measure of the cover space of movement of the robot. Based on this, it is possible to determine the cover space and therefore, the number of targets needed to train the NNs by using this threshold in a given environment. The key point relies on letting the threshold to converge by providing enough training samples.

### 3.5 Summary

A motion learning approach based on subsumption architecture was presented. The architecture design was focused on one level of the architecture while other levels were left open intentionally for further research. The purpose of the control architecture at this stage was to learn a set of basic movements based on random targets.

---

The presented architecture uses a genetic algorithm in order to find the movements that the robot must perform to reach a specific target, and then a set of neural networks are trained in order to perform such movements efficiently. Experiments showed that the proposed architecture is able to learn such set of movements and execute them efficiently using the set of neural networks.

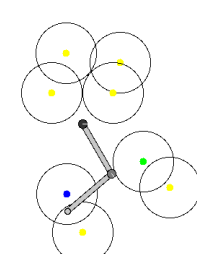
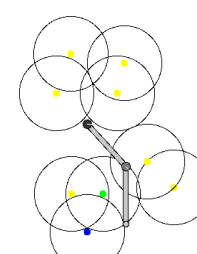
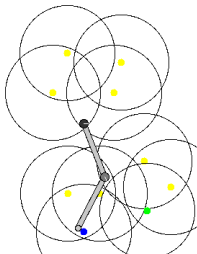
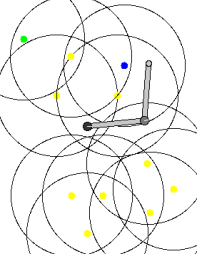
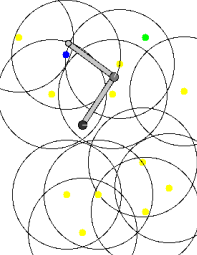
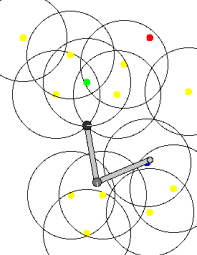
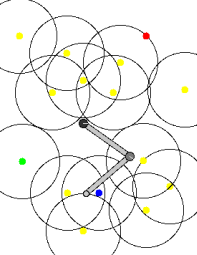
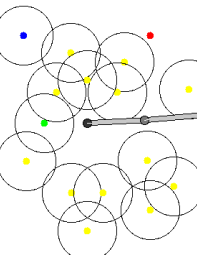
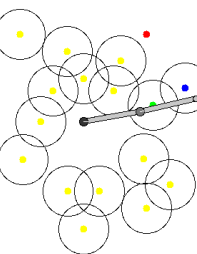
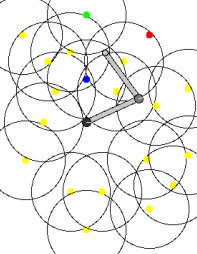
<p>Time (s)= 1.28 Executing instructions:2 [ 0   param(0)= 65.80645   [ 1   param(0)= 69.03225  </p> 	<p>Time (s)= 0.60 Executing instructions:3 [ 0   param(0)= 65.80645   [ 1   param(0)= 50.32258  </p> 
<p>Time (s)= 2.20 Executing instructions:3 [ 0   param(0)= 65.80645   [ 1   param(0)= 50.32258  </p> 	<p>Time (s)= 0.83 Executing instructions:4 [ 0   param(0)= -3.87096   [ 1   param(0)= -104.516  </p> 
<p>Time (s)= 1.46 Executing instructions:6 [ 0   param(0)= -59.0945   [ 1   param(0)= -96.7741  </p> 	<p>Time (s)= 1.96 Executing instructions:7 [ 0   param(0)= 81.29032   [ 1   param(0)= -104.516  </p> 
<p>Time (s)= 2.35 Executing instructions:8 [ 0   param(0)= 34.63970   [ 1   param(0)= -104.516  </p> 	<p>Time (s)= 0.20 Executing instructions:10 [ 0   param(0)= -120.0   [ 1   param(0)= -11.6129  </p> 
<p>Time (s)= 2.89 Executing instructions:11 [ 0   param(0)= -11.6129   [ 1   param(0)= -3.87096  </p> 	<p>Time (s)= 1.44 Executing instructions:13 [ 0   param(0)= -27.0957   [ 1   param(0)= -112.258  </p> 

TABLE 3.11. Threshold adjustment process for the 2DOF robot arm. Images are arranged from left to right and from top to bottom

---

## Conclusions



This Thesis addressed a specific problem in the robotics field: how robots can learn motion abilities. The work was based on the belief that motion skills can and must be learned in order to approximate the human motion ability. In this sense, the purpose of this Thesis was to create a control architecture capable of learning to properly move some simulated robots within a given environment. In other words, it is like giving a robot the ability to learn to move its own body without any specific knowledge about itself (its dimensions or construction). This approach, not only has the advantage of avoiding the redesign of the robot controllers when the robot structure is changed, but also it can allow a reconfiguration of the robot on the fly. Furthermore, the fact that the robot bases its movement on a learning procedure, instead of an exact mathematical calculation, gives it the ability to adapt better to the environment and its purpose.

To achieve this, a simple simulated environment was designed and used to simulate a robot with a specific purpose. The advantage of such an environment is that it helps to obtain data from simulations, is a visual tool that allows the user to observe the robot movements, and it can also control the simulation flow and other features through a graphical interface.

Based on the designed environment, a set of simple robots were also designed in order to test the control architecture. A second order dynamic model and a fuzzy

---

controller were used in these robots, in order to move them in a realistic way within the simulated environment. The chosen fuzzy controller provides some advantages over a more common controller as a PID, since it provides more flexibility in the quantity and quality of control parameters, and better adaptability features for specific movement conditions or purposes.

Among the different architecture models available, the subsumption model used in the architecture provides a great advantage in the way it combines the action-reaction modules with the long-term planning modules in a layered structure. Furthermore, the architecture improvement process is achieved through the inclusion of additional behaviors or layers. In this way, this control architecture, is able to control, from the simpler, to the complex robot, and the improvement of the system is feasible. However, as any subsumption architecture, the challenge relies on the designing of the interconnection mechanism between different behaviors and layers. In the control architecture presented in this project, this connection mechanism was represented by the training process and the reference angles given to the fuzzy controllers from the superior layer. While the training process is performed in the same layer, the reference angles are passed from the level 1 to the level 0 of the architecture. In fact, based on the used learning model, the behavior of "reaching a target" was performed from two different perspectives: from the GA and from the set of ANN, which were successfully combined inside the architecture.

The combination of two AI methods: the GA and the NNs, provided a successful mechanism of automated learning for the given motion purpose. Thus, while the GA used an exhaustive process to determine which were the best movements for reaching a specific target, the NNs were able to perform such movements rapidly once they have been learned. The advantage of this design is that the simulated robot, based on its own experience, is able to move efficiently. Besides, the use of random targets to train the neural networks, showed that the robot was able to learn appropriated movements without a specific learning path; or in other terms, without some external aid. Nevertheless, the results presented in this work showed that the generalization of this learning process is not easily achieved and it needs

---

the addition of some external information, as the DOFs relation within the neural networks, in order to improve the learning mechanism results.

The GA as a search method, certainly provided appropriate results and makes the architecture independent from the environment. In this sense, based on some goal and a suitable fitness function, it is possible for this architecture to approximate almost any behavior. However, it is important to point out, that the algorithm search space is not so large and consequently it finds a solution in few generations but, for very large solutions spaces, this search method may not be useful, since it may require too much time for the algorithm to find a feasible solution. Furthermore, the fitness function in the simulations is computed fast, but implemented robots require to perform all movements coded by a population on each generation, and taking into to consideration the time needed for a robot to perform a single movement, the time needed to learn a set of movements could be increase greatly.

The set of ANN are useful to memorize a set of movements and execute them efficiently. However, finding a suitable ANN structure for this purpose represent a complex task, since there are several factors that affect the performance of the ANN. In the architecture, a multilayer perceptron with Gaussian activation function and linear output was used for this purpose. This model has a simple structure, good approximation features and the training process is very simple (through back-propagation). Based on the different tests performed to this part of the control architecture, it is possible to highlight three main aspects that should be taken into account in order to obtain appropriate results: the training process, the neural network structure and the number of training samples. For the training process, it was necessary to use a defined algorithm to improve the results of the training targets. In this process, the re-training queue plays a fundamental roll since it give to targets, a second chance to be retrained. This way, the quality and quantity of trained targets was improved in the whole training process providing better results. Respect to the neural network structure, it was defined experimentally using two hidden layers of different sizes for each robot. The results showed a clear relation between the size of the neural network and the number of targets trained. In addition, the neu-



---

ral network structure was modified by including the actuators dependence into the structure. This modification, certainly improve the response drastically but with a high cost in the generalization capability of the architecture. In general, while in the first case there was no need of adding external information about the robot construction, in the second case it required to include the exact knowledge about actuators relations (robot construction). Finally, respect to the number of samples, it was a key factor in training process. Based on this parameter, the robot was able to cover most of the movement space and it was possible to determine the neural network size as well. Although in this Thesis this parameter was set to a fixed value, in a future application it should be dynamically adjusted as the robot learn new movements. Thus, a novel method should be implemented to decide whether a new target is trained or not. In the same way, the ANNs size can be modified dynamically as the number of targets is increased.

Finally, the feedback process between the GA and the set ANNs, really improves the performance of the control architecture in several ways. Using the previous knowledge to determine new movements, not only reduce the control architecture computation time and effort, but also it helps to train similar movements for closer targets, which improves the training process as well. Thus, while in the previous scheme the robot may train similar targets with very different motion solutions, this scheme avoided this issue by using the movements of a closer target. As a result, the training process was improved by creating a more consistent training set.

To summarize, the control architecture provides a mechanism capable of learning a specific behavior and execute it efficiently through a set of trained neural networks. The advantage is that this learning process allows the robot to abstract ,not only the environment, but also, the robot itself. In other words, the robot learns how to move its own body in order to achieve a specific goal into the given environment. Accordingly, the proposed control architecture facilitates the design process of "any" robot with a particular purpose, since it avoids the need of precise models of the robot components or the environment.

---

## Future work



There are plenty of experiments and improvements to do that are beyond the goals of this thesis, but based on the results presented on this work, there are some aspects that are worth to be studied in a further research. Here, only the main points are summarized.

- The subsumption architecture model is suitable for further expansion to more complex robots. Additional layers should be added to the control architecture in order to increase the movement ability of the robot and achieve better movements. The implemented architecture is thought to be a learning mechanism whose environment, components or configuration (the way motors are arranged to form the robot structure) does not affect the learning procedure. Therefore, further components such as different motions (behaviors), sensor identification and management modules, planning components, and so on, should be added through new layers in order to complete the proposed architecture.
- The robot purpose was to approach to a specific target. This purpose was not chosen in a random fashion but instead, the robot was thought as an arm that eventually would be able to grab, push or pull objects. In this sense, a possible next step is to add new behaviors and layers into the control architecture, that allow the robot learn to perform such kind of movements.

- 
- The developed simulation environment showed that the robot was capable of measuring distance and angle respect a different targets in a precise manner. However, in order to achieve more realistic results, further simulations should be performed using noisy sensor inputs.
  - Results showed that the Neural Networks are able to learn an execute a set of movements that allow the robot to reach a set of targets. However, more precision on the approximation to targets may be achieved by changing neural network structure or parameters. A multilayer perceptron with gaussian activation function was used, but there are plenty of neural networks types and activation functions that could improve the results obtained.
  - The lower layer, which was designed using a fuzzy controller with static parameters, may be easily modified in order to control more complex systems or to provide adaptability features to the control architecture. In this sense, a possible improvement of the architecture could be to include the fuzzy controller parameters into the evolutionary process in order to obtain smoother, faster or slower movements as required for a specific movement. However, depending on the actuators dynamic system, a dynamic change of the low level controller may produce unwanted instability issues. Therefore, this process should be done carefully in order to ensure robustness of system.
  - It is clear that actuators relations (based in the DOFs) and the neural networks size must be taken into account to improve the target approximation. However, using a determined NN size or adding the actuators relations to the control architecture implies the use of some external knowledge about the robot configuration. Based on this, a possible solution is to find a mechanism that perform not only the training for a given set of targets, but also, that adapts the neural network configuration using the same targets information.

## Bibliography



- [1] *Java (jdk)*, <http://www.oracle.com/technetwork/es/java/index.html>.
- [2] *jfuzzylogic, open source fuzzy logic library and fcl language implementation*, <http://jfuzzylogic.sourceforge.net/html/index.html>.
- [3] *Neuroph, java neural network framework*, <http://neuroph.sourceforge.net/index.html>.
- [4] J.A. Anderson, *An introduction to neural networks*, Bradford book, Mit Press, 1995.
- [5] Brenna Argall, *Learning mobile robot motion control from demonstration and corrective feedback*, Ph.D. thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, March 2009.
- [6] R. Babuska, *Fuzzy and neural control disc course lecture notes*, Delft University of Technology, 2001.
- [7] M. Beetz, J. Hertzberg, M. Ghallab, and M. E. Pollack, *Advances in plan-based control of robotic agents*, Lecture Notes in Artificial Intelligence, Springer-Verlag, 2002, p. 2466.
- [8] Josh C. Bongard, *Spontaneous evolution of structural modularity in robot neural network controllers: artificial life/robotics/evolvable hardware*, Proceedings of the 13th annual conference on Genetic and evolutionary computation (New York, NY, USA), GECCO '11, ACM, 2011, pp. 251–258.
- [9] R. Brooks, *Artificial life and real robots.*, Cambridge MA: MIT Press/Bradford Books., 1992.
- [10] Rodney A. Brooks, *A robust layered control system for a mobile robot*, IEEE Journal of Robotics and Automation **2** (1986), no. 1, 14–23.
- [11] Rodney A. Brooks, *New approaches to robotics*, Science **253** (1991), 1227–1232.

- 
- [12] S. K. Chalup, M. Dickinson, R. Fisher, R. H. Middleton, M. J. Quinlan, and P. Turner., *Proposal of a kit-style robot as the new standard platform for the four-legged league*, In: Australasian Conference on Robotics and Automation (ACRA) 2006, 2006.
- [13] A. Changuel and R. Jerraya, *Design of an adaptive motors controller based on fuzzy logic using behavioral synthesis*, European Design Automation Conference. Proceedings of the conference on European design automation, 1996, pp. 48 – 52.
- [14] Sesh Commuril, V. Tadigotlal, and L. Slingerl, *Task-based hardware reconfiguration in mobile robots using fpgas*, Journal of Intelligent and Robotic Systems **49** (2007), 11–134.
- [15] Jose A. Fernandez-Leon, Gerardo Gabriel Acosta, and Miguel Angel Mayosky, *Behavioral control through evolutionary neurocontrollers for autonomous mobile robot navigation.*, Robotics and Autonomous Systems **57** (2009), no. 4, 411–419.
- [16] D. Floreano and F. Mondada, *Evolution of homing navigation in a real mobile robot*, IEEE Transactions on Systems, Man and Cybernetics - Part B **3** (1996), 26.
- [17] Dario Floreano and Joseba Urzelai, *Evolution of plastic control networks*, Auton. Robots **11** (2001), no. 3, 311– 317.
- [18] Garcia A.S. Martinez S.S Garcia J.G., Ortega J.G., *Robotic software architecture for multisensor fusion system*, IEEE Transactions on Industrial Electronics **56-3** (2009), 766–777.
- [19] Erann Gat, *On three-layer architectures*, Artificial Intelligence and Mobile Robots, MIT/AAAI Press, 1997.
- [20] J. Gomez, *Self adaptation of operator rates in evolutionary algorithms*, In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), Lecture Notes in Computer Science (Springer-Verlag, ed.), vol. 3102, June 2004, pp. 1162–1173.
- [21] Stephen Gordon, *System integration with working memory management for robotic behavior learning*, The International Conference on Development and Learning, 2006.
- [22] Radek Grzeszczuk and Demetri Terzopoulos, *Automated learning of muscle-actuated locomotion through control abstraction*, SIGGRAPH 95 Conference Proceedings, Annual Conference Series, ACM Press, August 1995, pp. 63–70.
- [23] S. L. Harding and J. Miller, *Evolution of robot controller using cartesian genetic programming*, EuroGP 2005 Proceedings, Lecture Notes in Computer Science (M. Keijzer, A. Tettamanzi, P. Collet, J. van Hemert, and M. Tomassini, eds.), vol. 3447, 2005, pp. 62–72.

- 
- [24] J. Harris, *Fuzzy logic applications in engineering science*, vol. 29, Kluwer Academic Pub, 2006.
- [25] R. Hedjar, *Online adaptive control of non-linear plants using neural networks with application to temperature control system*, *Comp. Info. Sci* **19** (2007), 75–94.
- [26] John H Holland, *Adaptation in natural and artificial systems*, University of Michigan Press, 1975.
- [27] Gregory S. Hornby, Hod Lipson, and Jordan B. Pollack, *Evolution of generative design systems for modular physical robots*, In IEEE International Conference on Robotics and Automation, 2001, pp. 4146–4151.
- [28] M. Kishan, M. Chilukuri, and S. Ranka, *Elements of artificial neural networks*, MIT Press, 2000.
- [29] John R. Koza, *Genetic programming: On the programming of computers by means of natural selection*, MIT Press, 1992.
- [30] John R. Koza and James P. Rice, *Automatic programming of robots using genetic programming*, Proceedings of the Tenth National Conference on Artificial Intelligence, The MIT Press, 1992, pp. 194–201.
- [31] C.G. Langton, *Artificial life: an overview*, Complex adaptive systems, MIT Press, 1997.
- [32] Jean-Claude Latombe, *Robot motion planning*, Kluwer Academic Publishers, Norwell, MA, USA, 1991.
- [33] John H. Seinfeld Leon Lapidus, *Numerical solution of ordinary differential equations*, ACADEMIC PRESS, INC., 1971.
- [34] Chee M. Chew Lingfeng Wang, Kay C. Tan, *Evolutionary robotics: from algorithms to implementations*, World Scientific Publishing Co. Pte. Ltd., 2006, ISBN 981-256-870-0.
- [35] D.A. Linkens and H.O. Nyongesa, *Genetic algorithms for fuzzy control. 1. of-fine system development and application*, *IEEE Proceedings - Control Theory and Applications* **142** (1995), 161 – 176.
- [36] Maja J. Mataric, *Behavior-based control: Examples from navigation, learning, and group behavior*, *Journal of Experimental and Theoretical Artificial Intelligence*, special issue on Software Architectures for Physical Agents **9** (1997), 323–336.
- [37] Warren Mcculloch and Walter Pitts, *A logical calculus of ideas immanent in nervous activity*, *Bulletin of Mathematical Biophysics* **5** (1943), 127–147.

- 
- [38] Adelardo A. D. Medeiros, *A survey of control architectures for autonomous mobile robots*, JBCS - Journal of the Brazilian Computer Society, special issue on Robotics **4** (1998), no. 3, 35–43.
- [39] Melanie Mitchell and Charles E. Taylor, *Evolutionary computation: An overview*, Annual Review of Ecology and Systematics **30** (1999), 593–616.
- [40] Duy Nguyen-Tuong and Jan Peters, *Model learning for robot control: a survey*, Cognitive Processing **12** (2011), no. 4, 319–340.
- [41] M. N. Nicolescu and M. J. Mataric, *A hierarchical architecture for behavior-based robots*, Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems, 2002.
- [42] S. Nolfi and D. Floreano, *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines*, MIT Press., 2000.
- [43] Leandro Nunes de Castro, *Fundamentals of natural computing: an overview*, Physics of Life Review **4** (2007), 1 – 36.
- [44] Mohamed Oubbati, *Neural dynamics for mobile robot adaptive control*, Ph.D. thesis, Universität Stuttgart, 2006.
- [45] Jan Peters, Stefan Schaal, and Bernhard Schölkopf, *Towards machine learning of motor skills*, Autonome Mobile Systeme 2007 (2008), 138–144.
- [46] Stuart Russell and Peter Norvig, *Artificial intelligence. a modern approach third edition.*, Prentice Hall, 2010.
- [47] Stefan Schaal and Nicolas Schweighofer, *Computational motor control in humans and robots*, Current Opinion in Neurobiology **In Press, Corrected Proof** (2005), 675–682.
- [48] W. Shepherd, L.N. Hully, and D.T.W. Liang, *Power electronics and motor control*, Cambridge University Press, 1995.
- [49] Amel SIFAoui, Afef ABDELKRIM, and Mohamed BENREJEB, *On the use of neural network as a universal approximator*, International Journal of Sciences and Techniques of Automatic control & computer engineering IJ-STA **2** (2008), 386–399.
- [50] N. Sundararajan, P. Saratchandran, and Lu Ying Wei, *Radial basis function neural networks with sequential learning*, World Scientific Publishing Co. Pte. Ltd., 1999.
- [51] Julian Togelius, *Evolution of a subsumption architecture neurocontroller*, J. Intell. Fuzzy Syst. **15** (2004), no. 1, 15–20.
- [52] L. W. Tsai, *Robot analysis: The mechanics of serial and parallel manipulators*, Wiley-Interscience, 1999.

- 
- [53] Ching-Chang Wong, Kai-Hsiang Huang Chai-Tai Cheng, and Yu-Ting Yang, *Fuzzy control of humanoid robot for obstacle avoidance*, International Journal of Fuzzy Systems **10** (2008), 1.