FUZZY ACTOR-CRITIC LEARNING BASED INTELLIGENT CONTROLLER FOR HIGH-LEVEL MOTION CONTROL OF SERPENTINE ROBOTS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

EVRİM ONUR ARI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

NOVEMBER 2005

Approval of the Graduate School of Natural and Applied Sciences

Prof. Dr. Canan ÖZGEN
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. İsmet ERKMEN
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science

Prof. Dr. Aydan M. ERKMEN Co-supervisor	Prof. Dr. İsmet ERKMEN Supervisor

Examining Committee Members

Prof. Dr. Erol KOCAOĞLAN	(METU,EE)	
Prof. Dr. İsmet ERKMEN	(METU,EE)	
Prof. Dr. Aydan ERKMEN	(METU,EE)	
Prof. Dr. Kemal İDER	(METU,ME)	
Günay Şimşek, M.Sc.	(ASELSAN)	

ABSTRACT

FUZZY-ACTOR CRITIC LEARNING BASED INTELLIGENT CONTROLLER FOR HIGH-LEVEL MOTION CONTROL OF SERPENTINE ROBOTS

ARI, Evrim Onur
MSc., Department of Electrical and Electronics Engineering
Supervisor: Prof. Dr. İsmet ERKMEN
Co-Supervisor: Prof. Dr. Aydan M. ERKMEN

November 2005, 148 pages

In this thesis, an intelligent controller architecture for gait selection of a serpentine robot intended to be used in search and rescue tasks is designed, developed and simulated. The architecture is independent of the configuration of the robot and the robot is allowed to make different kind of movements, similar to grasping. Moreover, it is applicable to parallel processing in several aspects and it is an implementation of a controller network on robot segment network. In the architecture several behaviors are defined for each of the segments. Every behavior is realized in the form of Fuzzy Actor-Critic Learning agents based on fuzzy networks and reinforcement learning. Each segment controller determines the next suitable position in the sensory space acquired using ultrasound sensors, a genetic algorithm implementation then tries to find the change of the joint angles to achieve the desired movement in a given amount of time. This allows optimization on different criteria, during motion. Simulations are performed and presented to introduce the efficiency of the developed controller architecture. Moreover a simplified mathematical analysis is performed to gain insight of the controller dynamics.

Keywords: Serpentine robots, rescue robots, planning in unstructured environment, reinforcement learning, Fuzzy Actor-Critic Learning (FACL).

ÖZ

YILANSI ROBOTLARIN ÜST SEVİYE DENETİMİ İÇİN BULANIK EYLEYİCİ-ELEŞTİRİCİ ÖĞRENMEYE DAYALI AKILLI DENETLEÇ

ARI, Evrim Onur
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. İsmet ERKMEN
Ortak Tez Yöneticisi: Prof. Dr. Aydan M. ERKMEN

Kasım 2005, 148 sayfa

Bu tezde, arama kurtarma çalışmalarında kullanılmak üzere tasarlanan yılansı bir robotun üst seviye hareket denetimi için akıllı bir kontrol sistemi mimarisi geliştirilmiş ve simule edilmiştir. Geliştirilen mimari, robotun mekanik yapı detaylarından bağımsızdır ve nesne kavrama gibi farklı hareketlerin gerçeklenmesine olanak sağlamaktadır. Buna ek olarak, geliştirilen mimari bir çok yönden paralel işlenmeye uygundur ve kontrol ağının robotun mekanik segment ağı üzerine yerleştirilmesiyle oluşturulmuştur. Mimaride robotun her segmenti için çeşitli davranışlar tanımlanmıştır. Her davranış "Bulanık Mantık" ve "Kuvvetlendirici Öğrenme" kullanan "Bulanık Eyleyici-Eleştirici Öğrenmeye Dayalı Akıllı Denetleç" yapılarıyla gerçeklenmiştir. Robotun her segment denetleci, durum uzayında kendisi için en uygun bir sonraki pozisyona karar verir. Bunun ardından "genetik algoritma" kullanan bir en iyileyici belirli bir zaman diliminde, robotu eski durumundan yeni durumuna geçirmek için yapılması gereken segment hareketlerini belirler. Bu sayede, robot hareketinin farklı kriterler gözetilerek en iyilenebilmesi sağlanmıştır. Geliştirilen mimarinin uygulama sonuçlarını ve etkinliğini göstermek amacıyla simülasyonlar yapılmış ve sonuçları tezde sunulmuştur. Buna ek olarak, denetlecin dinamiği hakkında fikir vermesi amacıyla, basitleştirilmiş matematiksel analizler yapılarak çalışmaya eklenmiştir.

Anahtar Kelimeler: Yılansı robotlar, arama kurtarma robotları, yapılandırılmamış ortamlarda yol planlama, kuvvetlendirici öğrenme, "Bulanık Eyleyici-Eleştirici Öğrenme".

To Aylin, for her endless love...

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to Aydan M. Erkmen and İsmet Erkmen for their guidance throughout this study.

I particularly thank to my mom and dad for their lifelong support.

I am grateful to my colleagues Günay Şimşek, Murat Ertek, Mehmet Karakaş and Eray Özçelik for their friendship and morale support.

Finally, I greatly appreciate Seyit Yıldırım and Bülent Bilgin since they allowed me to use powerful workstations at ASELSAN in order to implement the computationally heavy simulations, and showed great understanding whenever I spent my work time on this thesis study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Search and Rescue Robotics – Main Motivation of This Study

Since their first development in the first half of 20th century, robots played a very important role in the quality of human life. At first, they were mainly utilized in factories for automation of production and transportation oriented tasks. Afterwards, as radio frequency science, electronics production techniques, and artificial intelligence further developed, they found deployment in military, space and intelligence missions; and now they are starting to enter into our lives more and more every day. As catastrophic disasters are natural pieces of our lives, search and rescue (SAR) –in its classical context- has been a field of interest for a very long time. After a disaster, catastrophic effects make it difficult for people to investigate certain regions (e.g. under ruins of collapsed buildings) due to several dangerous factors similar to a risk of fire, structural collapse, probability of existence of dangerous chemicals in the environment etc. In recent years, usage of robots in search and rescue (SAR) missions became a popular field of robotics study; which found its first real application in the SAR mission at World Trade Center buildings after 9/11 terrorist attack [1]. However, there are still many engineering problems waiting to be solved, in this field of study.

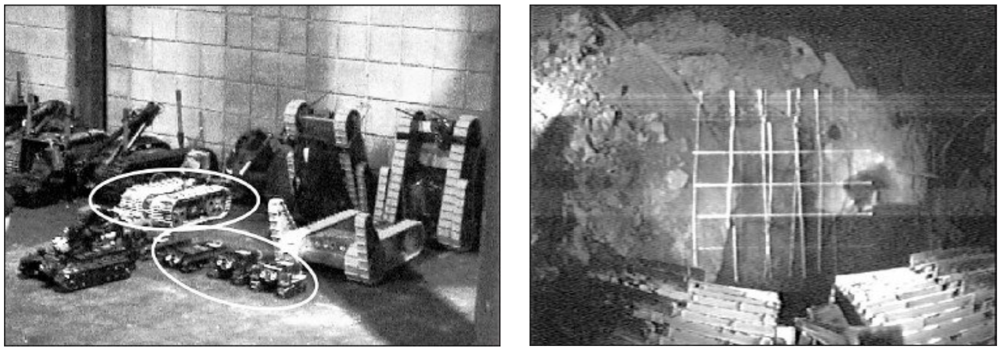


Figure 1.1 – Rescue robots employed in the rescue operation after 9/11 are shown on the left, with robots used in rubble pile are circled. A view from the camera of one of the robots during SAR operation is on the right. [1]

Recent experiences of natural disasters (earthquakes, tsunami etc.) and man made catastrophes (terrorism) have put more emphasis to the area of search and rescue (SAR) and emergency management. Technological progress in construction techniques are proven to be highly insufficient in their implementation spread, and the preparedness and the emergency response of governments have shown to be still highly inadequate in dealing with these devastations. As a result, huge human losses due to lack of immediate response with efficient technological devices forced engineering to find better solutions for SAR. As a consequence, autonomy, high mobility, robustness and reconfigurability for terrain and task adaptation are critical design issues of rescue robotics requiring dexterous devices equipped with the ability to learn from prior rescue experience, to adapt to variable types of usage with a wide enough functionality under different sensing modules and compliancy to environmental and victim conditions. Although first robots used in SAR applications were non-autonomous, tele-operated devices; autonomy has become a focused interest in order to increase the versatility of such robots over. For example a wireless tele-operated robot was lost during SAR operations after 9/11 [1], and such a situation would have been prevented by use of full autonomy in robots; at least semi-autonomy would have been a solution. Some aspects of rescue robotics are [4], [32]

- Detection and identification search of living bodies to prevent workers from damaging a victim’s limbs during evacuation of rubles.
- Routing and/or clearing of debris in accessing the victim.
- Physical, emotional, or medical stabilization of the survivor by bringing to him/her automatically administered and telemetered first aid.
- Fortification of the living body for secure retrieval against any falling debris and possible injuries.
- Transportation of the victim.

1.2 Serpentine Robots for SAR Applications

Motion mechanisms of robots used in SAR applications vary according to some specific needs such as speed, traction, payload carriage and most importantly rough terrain adaptation. Most popular mechanism of motion for mobile robots is differential steering; having two wheels/palettes driven by separate electric motors. Such mechanisms are relatively efficient, easy to steer and suited for high-speed driving on a smooth surface. However, they are not effective in rugged environments such as rough or muddy terrains. Mobile robots with legs are being actively researched for several reasons, including the fact that legs provide higher terrain adaptability than wheels. Even higher terrain adaptability may be achieved by multi-link articulated (also called as “hyper-redundant”, “tentacle-like”, “spine”, “snake-like”, “elephant trunk”, “tensor-arm” [8]) robots that “crawl” like snakes [2]. Although snakes are handicapped by having no limbs, capability of a wide range of body elongation with a highly redundant structure, and different types of locomotion modes adaptable to different environments make them very versatile in applications on different terrain characteristics [5], [7]. As given in [3], when compared to legged robots that can undertake similar locomotion abilities for SAR, snake like robots are more advantageous in the following capabilities:

Terrainability: Snake-like robots can traverse rough terrain; they can climb steps whose heights approach its longest linear dimension [9], pass soft or viscous materials, span grasps, etc. This property makes snake-like robots very distinguished.

Traction: Snakes can use almost their full body length to apply forces to the ground for generating traction. Hence, possibility to fall over is very low, in contrast to wheeled or legged counterparts.

Efficiency: Snake-like locomotion is done under low costs of body support, and no cost of limb motion, but, with high friction losses, and lateral accelerations of the body. If effective motion planning is employed, a good compromise between motion efficiency and task adaptability may be achieved.

Size: Narrow frontal area allows penetration of small cross-sectional areas than many equivalent legged or wheeled vehicles. A snake-like robot can pass any passage that its first segment can pass.

Redundancy: Serpentine vehicles consist of many similar segments. The loss of the function of some of them may be compensated for recovering the functional efficiency that may be partially lost. This property may only be achieved by building additional wheels/legs etc. for other types of robots. However redundancy is an inherent property of snake-like robots. [34],[35].

Sealing: The surface of a serpentine vehicle is small and does not need to be exposed to the environment in the same way as limbs. All body parts may be coated with protecting material without difficulty. This provides advantage for applications in hostile environments.

Although snake robots have some advantages over legged or wheeled robots, there are also disadvantages of serpentine robots, which may be given as follows [3]:

Payload: Transport of materials is difficult unless an integral platform is used. This may be considered as a big handicap in SAR applications; however, other properties of snake-like robots are usually more useful than payload carriage capability, since first aim in SAR is finding living victims, bringing material to them is a secondary issue. Moreover, special designs may be considered to increase payload capability of the robot.

Thermal Control: The long stretched form of snakes makes thermal control a difficulty for engineers. Hardware that needs cooling or heating must be treated separately. Hence, additional precautions must be taken for thermal management.

Speed: Robot snakes are slower than their natural counterparts (reaching speeds up to 3.0 m/s), and also far slower than wheeled vehicles. However, faster robots may not be able to use this property anyway in rugged environments such as those found in most SAR applications.

Number of Actuators: Due to high degree of freedom of snake robots, a high number of actuators are needed for realizing snake-like motion. This property increases cost of the robot in several designs.

Control Complexity: Due to highly redundant structure of the robot and the non-linearity of the motion mechanism; controlling the motion of a snake-like robot is a difficult engineering problem.

Having introduced search and rescue robotics and properties of serpentine robots; it is not surprising to state that serpentine robots are considered to be very suitable for SAR applications by SAR experts [1],[4],[36]. However, in order to be used in such complex missions, several design issues concerning hyper-redundant robots must be considered and solved. These issues may be stated in two categories. One is the realization of the hyper-redundant locomotion, which is defined as “the process of generating net displacements of a hyper-redundant robotic mechanism via internal mechanism deformations, i.e. without actuation by wheels, tracks, or legs” [7]. The second one is the problem of path and motion planning with a high-level controller. Most of the studies on serpentine robots have been concentrated on the realization of hyper-redundant locomotion; these include designs for joint mechanisms [33], low-level control architectures for actuators in order to realize hyper-redundant motion [36],[37],[38], mechanical designs for whole robot, modeling techniques on hyper-redundant locomotion [7], proposition of new motion types [5],[17] etc. However, studies on the high-level control offer a good ground for intelligent control applications. Path planning studies for wheeled mobile robots are abounding, and adaptation of these methods to snake-like robots may be equally attempted.

1.3 Intelligent Control and Learning Techniques

The aim of intelligent control is designing control systems that exhibit characteristics associated with intelligence in human behavior, such as understanding, learning, reasoning, problem-solving, and so on [6]. When a human being tries to achieve a certain task, he actually behaves like a controller in a closed loop control system. What makes human beings perform in learning to control a system may be summarized in a three-state loop: First they *decide* on a control action and apply it, secondly they see the results of their action and *evaluate* the resulting situation, and finally they *modify* their action in a next same situation if necessary. Several controllers have been designed using this approach. Such designs have shown great success in solving problems including high uncertainty and non-linearity. Hence, they are also suitable for application of high-level control of serpentine robots, an instance of which can be found in this thesis study.

1.4 Outline of the Thesis

In the first chapter an introduction to the topics included in the thesis, and the motivation behind this study are given. The second chapter is devoted to a literature survey, detailed enough to cite the previous work on serpentine robots, and related low-level and high-level control studies, and intelligent control basics with some detail in learning fuzzy controller architectures. During the third chapter, the high-level control architecture proposed for the SAR applications is introduced and explained in detail. In the fourth chapter, simulation results of the conducted study and the performed quantitative analysis are presented. In the fifth chapter, summary, conclusions and the possible future work are given followed by references and the appendix.

CHAPTER 2

LITERATURE SURVEY

2.1 Snake-like Robots

Snakes do not use any limbs for locomotion, and displace themselves using a special sequence of body postures, which vary according to environmental conditions. These sequences of postures, leading to a net displacement of the body are called “gaits”. Snakes achieve a typical gait by using an average of 200 degree of freedoms of their articulated body [9]. In snake-like robots, which are inspired from their biological counterparts, no active artificial limbs; like wheels or legs, are used. Instead, the robot uses its whole body for generating motion. This is accomplished in a way similar to natural snakes: Change of the body shape (which is articulated) and use of the resultant frictional and contact forces in order to generate a net displacement in space. As stated in the introduction part, such kind of a motion mechanism has several advantages in SAR applications. For limbless robots, inspiration from nature has guided the design of motion mechanisms.

Research of snakes’ very versatile motion mechanism was not conducted until 1920s. Petr Miturich, a Russian constructive artist, studied the undulation type of snake gait and pioneered the work on snake-like locomotion [3]. From late 1960s on, by the advances in robotics and researchers’ interest in biologically inspired robots, snake-like robots became an active research area in robotics.

First snake-like robot designs are those of the Japanese pioneer of serpentine robot research, Shigeo Hirose ([2],[3],[7],[13] etc.). Aim of Hirose was to formulate the motion of a snake as a combination of moving sinusoidal waves. He realized the natural serpentine motion on his first robotic snake design called “Active Cord Mechanism” (ACM) moving at a speed of approximately 40 cm/sec. The entire length of this device was 2 m., having 20 joints, each consisting of a servomechanism that can bend left and right laterally. The propulsion motion is generated by input commands, which impart sinusoidal bending motions to the head joint servomechanism. He and his coworkers, since then, developed several mechanisms for snake-like motion and conducted basic research on the mechanism design and serpentine motion characteristics. Shown in Figure 2.1.1, the ACM-R3 is one of Hirose's latest designs [11].

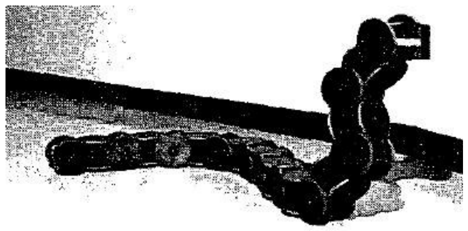


Figure 2.1.1 ACM-R3 developed by Hirose's team.

Joel Burdick and Gregory Chirikjian worked on serpentine mechanisms from the viewpoint of kinematics modeling. They developed the model of so called "backbone curve" for serpentine robots, which they fit parametrically on the body of the snake [7], [8], and perform motion planning on this hypothetical curve, which one-to-one maps to the body of the real robot. They built a snake-like robot mechanism based on variable geometry trusses (VGTs) (see Figure 2.1.2), made up of commercially available linear actuator, and demonstrated a couple of described gaits. Although the shape of the body was not exactly snake-like, the motions were identical to those of the real snakes. In their later studies they described several theoretical gaits and demonstrated different approaches on serpentine motion, like lasso-type grasping.

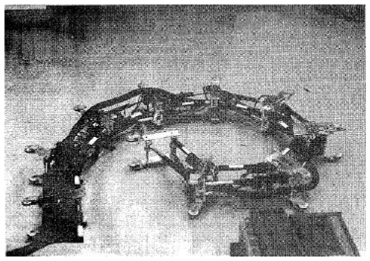


Figure 2.1.2 – A photograph of the 30 DOF snake-like robot developed by Chirikjian et al. [39].

In early 1990s Shan and Koren worked on a different approach for serpentine motion planning [10]. In its planning, the robot should not always avoid obstacles; but should also utilize them for locomotion taking them as fixtures while moving. The authors only used a kind of concertina motion described in discrete motion steps of different segments of their robot. They also used solenoids to drive vertical pins into the ground surface in order to give immobilization at desired points (see Figure 2.1.3). Such an approach establishes fixed contact points from which the rest of the mechanism relies upon during motion.

Ikeda and Takanashi developed an active universal joint, a novel form of Hooke's joint and implemented it on a seven-segment snake-like robot configuration [12] (Figure 2.1.4). A small pan-tilt video camera was also deployed at the head of the mechanism and used by the operator to assist in guiding the snake. Control is performed manually and the single gait used is similar to that of a rectilinear or inchworm gait. This class of mechanism has great promise for serpentine robots in real-world applications. The key advantage in this robot was effective packaging of the mechanism, the slim design and the modularity of the links.

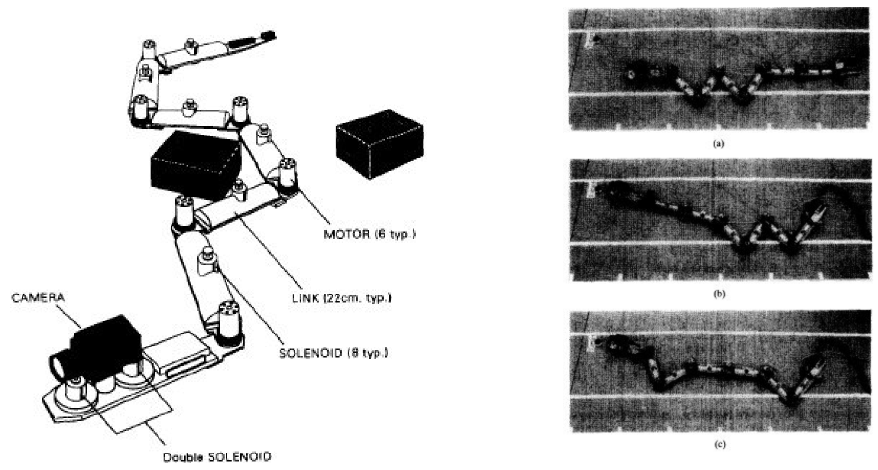


Figure 2.1.3 – Model and photographs of the snake robot developed by Shan et al. [10].

Martin Nilsson of the Swedish Institute for Computer Science has developed a novel universal serpentine link, in the form of a roll-pitch-roll joint, as a part of the

PIRAIA project of the institute (Figure 2.1.5). Multiple links give the mechanism the ability to subtend some very non-snakelike modes of locomotion that incorporate a rolling motion. In one instance, the snake ‘hugs’ a tree and, using the side rolling capability, roll directly up the tree. The joint has another nice feature such that the cables passing through the joint do not twist if the joint is controlled properly. The mechanism, despite being relatively complex, can be realized with standard components. Nilsson’s mechanism is very different from other works and from natural snakes. The two roll motions at each joint give wheel-like effectiveness in locomotion, but also complicate internal mechanics. With this joint design Nilsson achieved experiments on a free climbing gait, which is almost unique to a serpentine robot. [9]

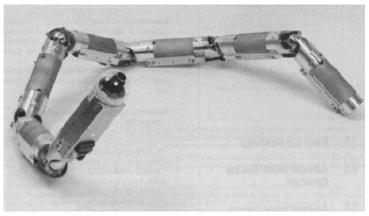


Figure 2.1.4 – Snake robot “Orochi” developed by Ikeda and Takanashi.

Karl Paap and his group at GMD (German National Research Center for Information Technology) in Germany developed a snake-like device to demonstrate concepts and developments for real-time control [40]. The robot is a tensor device that uses short sections with cable winding mechanism to affect curvatures along several segments (Figure 2.1.6). The curvatures are continuous along those sections, but the joining segments, where the drive mechanisms are located, do not bend or move. Some very limited locomotion has been shown in the mechanism and the cable drives have been a design challenge. They also deployed a pan-tilt camera on the head segment of the robot.

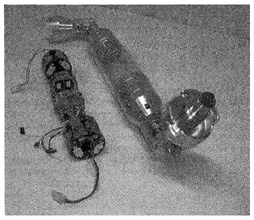


Figure 2.1.5 – The robot segment developed y Nilsson. [9].

IS robotics built a small snake-like machine, named as Kaa, for prehensile grasping and displacement of pipes [45] (Fig. 2.1.7). Being not an effective locomotion device, the robot was initially designed for moving in and through networks of pipes and support structures. It is probably the first completely self-contained snake-like moving robot. Using RC-servos as actuators, the robot propagates a ripple along the body to generate a straight-line motion on a flat surface. The movement is limited and the large box in the middle of the robot, housing power and processor, makes the locomotion problematic.

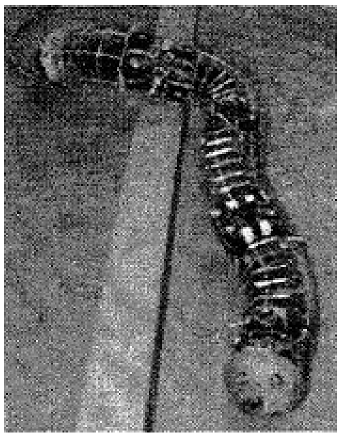


Figure 2.1.6 – GMD Snake 1 developed by Paap and his coworkers [40].

Coupled mobility devices, sometimes called overland trains, are similar to trains of vehicles linked together. Although Hirose's ACM robots resemble a coupled mobility device, all wheels are passive and the robot skates on wheels by means of body movements. Others, such as Odetics' All-Terrain Mobility System (ATMS) (Fig. 2.1.8) are coupled mobility devices with active couplings designed to traverse a variety of terrains. In such devices, both link and wheel motions are explicitly described within a single movement.

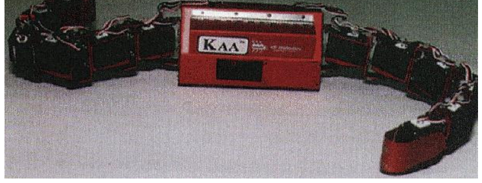


Figure 2.1.7 – Kaa of IS Robotics

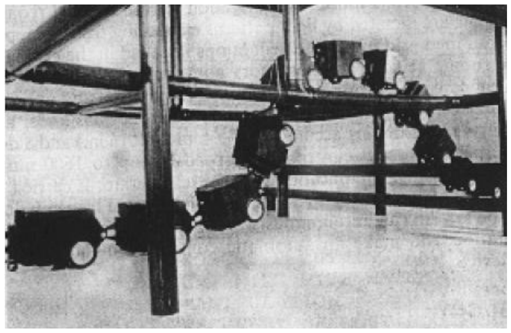


Figure 2.1.8 – Odetics' ATMS can climb and cross over obstacles [41].

Dr. Mark Yim designs prototypes shaping themselves into a wheel to roll over flat grounds shifting their shape into a spider to tackle uneven surfaces and then morphing themselves again into a snake shape that slithers through narrow spaces [35] (Figure 2.1.9). The sensors help the modules read their positions relative to one another. These prototypes are radio-controlled, with a camera on the leading segment. There are two big advantages of these robots. First, they are self-repairing. If one of the identical parts breaks in action, the machine compensates. Second, since all the parts are identical, they can be mass-produced, and they can be cost cheaper than hand-built unique robots. Moreover, since they are adaptable, one robot can perform more than one task.

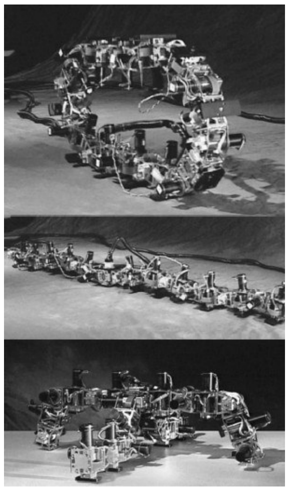


Figure 2.1.9 – Polybot developed by Mark Yim and his team.

Munerato and his colleagues at Univ. of Metz (France) introduce a mobile robot, acting like an earth-worm when both ends are free, or like an elephant trunk, when one end of the robot is fixed [46]. Two platforms with a special 3 degree-of-freedom (DOF) joint with 3 linear actuators connect each segment. As linear actuators, screw-drives and micro motors with gear heads and encoders are used. This articulation has been realized on a scale 2 model that eliminates mechanical problems like radial parasitic effort and axial torque. The robot body consists of 20 similar modules, resulting in 60 DOFs to be controlled.

Kamegawa and his coworkers develop and implement a snake-like robot moving in 3D [15] (See Figure 2.1.10). They propose new mechanical design issues, and several novel locomotion modes. They also develop a genetic algorithm based method for transition of the robot configuration between locomotion modes.

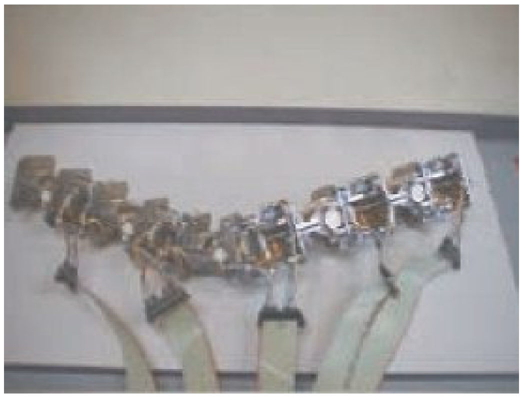


Figure 2.1.10 – Kamegawa et al's snake robot.

Gravagne et. al. design a continuum manipulator inspired from elephant's trunk [42], also stating the equations defining the geometry and kinematics of the structure. Such kind of a mechanism has proven to be highly suitable for applications including grasping (see Figure 2.1.11).

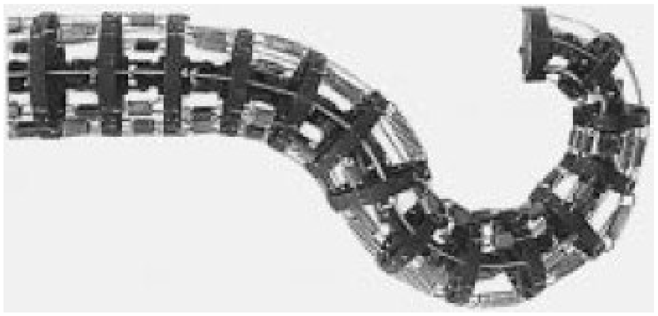


Figure 2.1.11 – The continuum manipulator developed by Gravagne and Walker [42].

Dr. Khosla’s team at CMU (Carnegie Mellon University) develops robot modules named as “Milibot”s and adapts them to serpentine-like structure named as “Milibot train” [34] (See Figure 2.1.12). The individual modules of the robot are able to apply torque at the links, making this robot able to climb stairs, about at a height that is half of the length of the whole train.

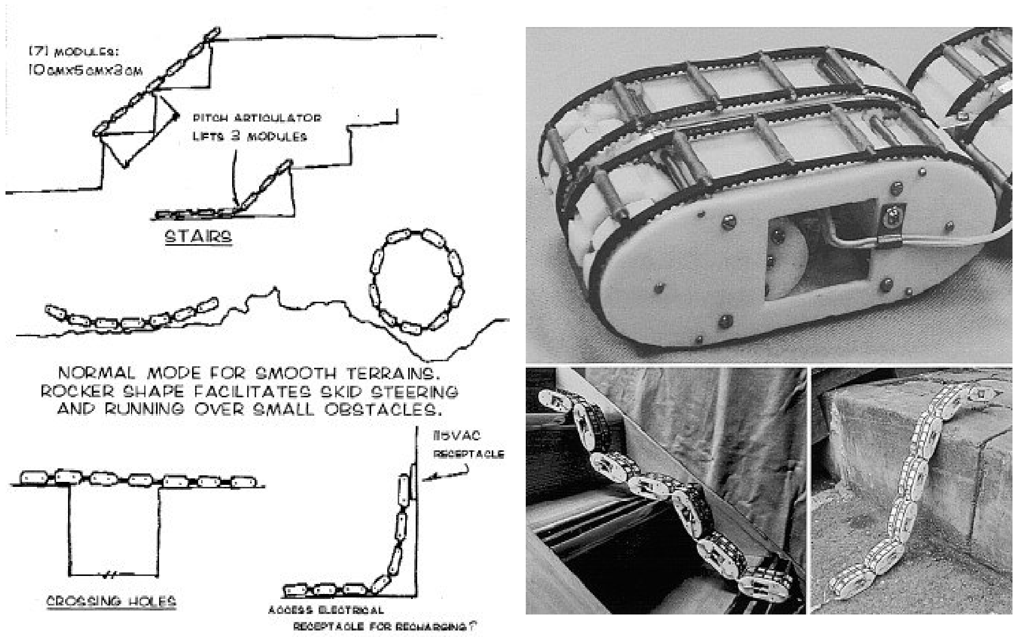

(7) MODULES:
10CMX5CMX3CM
PITCH ARTICULATOR
LIFTS 3 MODULES
STAIRS
NORMAL MODE FOR SMOOTH TERRAINS.
ROCKER SHAPE FACILITATES SKID STEERING
AND RUNNING OVER SMALL OBSTACLES.
115VAC
RECEPTACLE
CROSSING HOLES
ACCESS ELECTRICAL
RECEPTACLE FOR RECHARGING?

Figure 2.1.12 – Milibot train developed by Dr. Khosla’s team at CMU.

Several other, and similar, examples may be found in the literature. Here we just stated examples in order to give an idea about the structures and the abilities of snake-like robots.

2.2 Serpentine Motion Gaits

High degree of freedom makes hyper-redundant robots superior for operation in highly constrained environments; and many conceivable applications require hyper-redundant robot to maneuver, via some form of locomotion, within its environment [5]. Hence, we should redefine the locomotion of hyper-redundant robots, as given in [5].

Definition: Hyper-redundant locomotion is the process of generating net displacements of a hyper-redundant robotic mechanism via internal mechanism deformations; actuation through wheels, tracks, or legs is not necessary.
Definition: A *gait* is a distinct repetitive cycle of mechanism deformation that leads to net robot displacement.

Hence, any sequence of mechanical deformations leading to net displacement of the robot may be defined as being a different gait. Although locomotion of a snake-like robot is adapted from the natural snake gaits, there are also some applied types of robotic locomotion where natural snake gaits cannot be observed. So, the locomotion of a snake-like robot is classified into two types, as *natural snake gaits*, and *unnatural snake gaits*. Extensive definitions and explanations of the gaits found in Dowling's Ph.D. thesis [41] will be overviewed shortly in the coming sub-section.

2.2.1 Natural Snake Gaits

Rectilinear Motion: This type of locomotion is achieved by a wave propagation on the vertical plane along the length of the body [41]. The advantages of this gait are medium energy consumption, and minimum friction with the ground. Its main disadvantage is instability problem that occurs since some parts of the body are lifted above ground, decreasing the number of ground contact points, hence friction. (Fig. 2.2.1)

Lateral Undulation: In this type of gait, a wave is propagated on the horizontal plane [41]. This locomotion provides a sliding motion on the ground , and

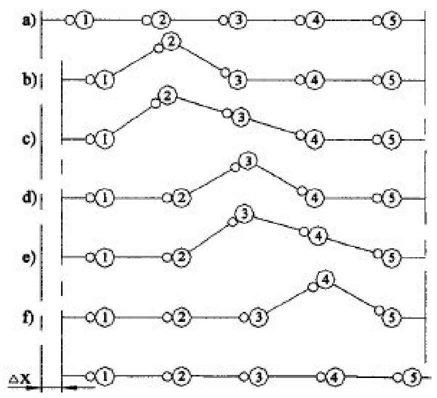


Figure 2.2.1 – A schematic description of rectilinear motion. [43].

slightly lifts some segments with an upward lateral wave. The advantages of this gait type are that no problem of instability occurs, and the energy consumption is reduced. Moreover, this gait may be used in robots with heavy bodies. Main disadvantages of lateral undulation are that rubbing is not negligible that the snake needs wide space, and can not work on smooth surfaces, and that the motion is hindered by a great body mass. (Figure 2.2.2)

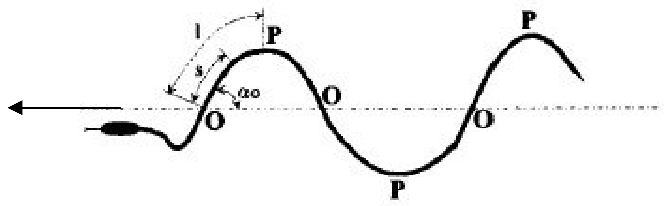


Figure 2.2.2 – A schematic description of lateral undulation motion. [5].

Concertina Progression: In this type of gait, some segments of the robot are folded and unfolded with a low amplitude vertical wave along the body [41]. There is no problem of instability and higher speeds may be achieved; this gait even works in narrow spaces. However, there is a considerable amount of energy consumption, important rubbing and inefficiency. (Fig. 2.2.3)

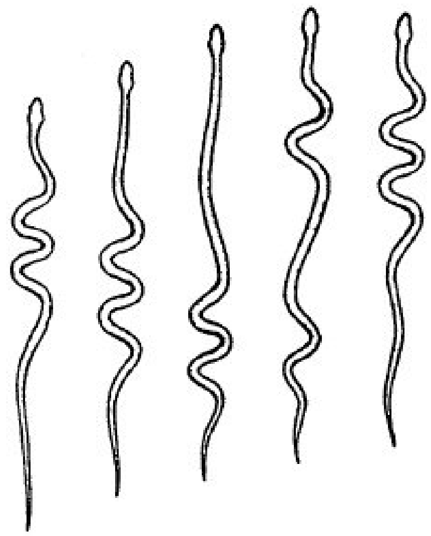


Figure 2.2.3 – A schematic description of concertina progression. [15].

Earthworm Motion: In earthworm motion, some segments of the body shrunk and stretch without producing a wave in the horizontal plane [Malachi]. An elongation / contraction motion is produced to move the segments. There is no problem of instability, energy consumption is reduced and rubbing is negligible. However, elongation and contraction of robot segments lead to difficult mechanical design issues. (Fig. 2.2.4)

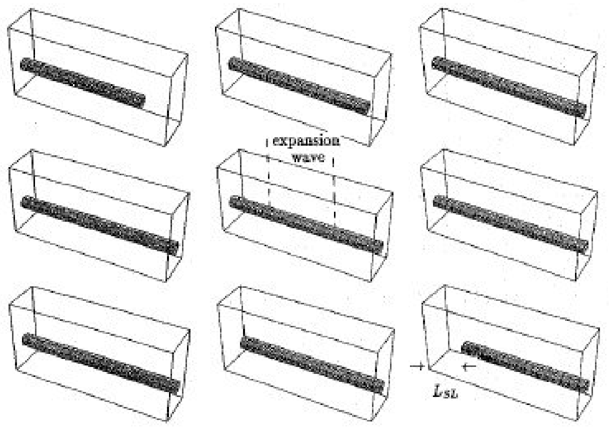


Figure 2.2.4– A schematic description of earthworm motion. [15].

Side-winding: This type of motion is achieved by producing two waves, both in the horizontal direction and the vertical direction. The waves are out of phase, so as to produce a net displacement in the direction perpendicular to both of the wave directions [41]. In side-winding, rubbing is negligible and it may be used on soft grounds. The energy consumption is in a medium level. This type of gait is not suitable in narrow locations. (Fig. 2.2.5)

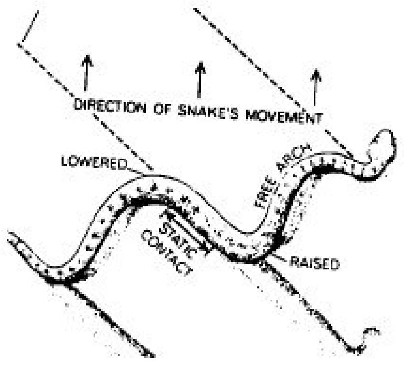


Figure 2.2.5– A schematic description of side-winding motion. [44].

2.2.2 Unnatural Snake Gaits

Flapping Locomotion: The robot moves perpendicular to its body alignment [41]. In flapping motion vertical and horizontal plane waves are in phase, and the robot resembles a bird flapping its wings to move forward. The friction becomes negligible with this type of motion, however the displacement velocity is relatively slow. (Figure 2.2.6)

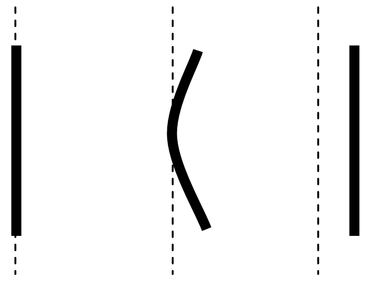


Figure 2.2.6 – A schematic description of flapping motion.

Lateral Rolling: The locomotion of the robot is achieved by producing horizontal and vertical waves, which are in phase [41]. The shape of the body is curled like the letter 'U', then the body rolls giving a net displacement perpendicular to the alignment of the body. With this type of motion energy consumption is reduced, and the stability is increased; however the displacement velocity is slow.

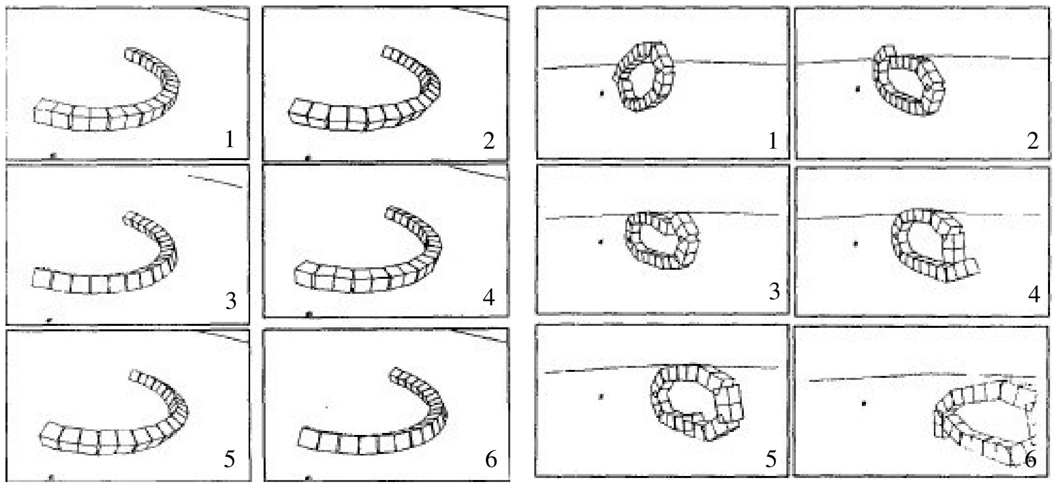


Figure 2.2.7– Schematic description of lateral rolling and wheel motion [41].

Wheel Motion: Head and tail segments of the body are joined together to form an ellipsoid. The motion is achieved by changing the link angles in the vertical direction. (also called "ring mode" in [13]) The resultant motion resembles like a rolling wheel. In this type of motion the velocity is considerably high, but stability is a big problem. (Fig. 2.2.7)

Bridge Mode Locomotion: Robot is configured to stand on its two end segments in a bridge-like shape. Bipedal motion can be implemented in this mode. The basic movement consists of left-right swaying of the center of gravity in synchronism by lifting and forwarding one of the supports [13]. (Fig. 2.2.8)

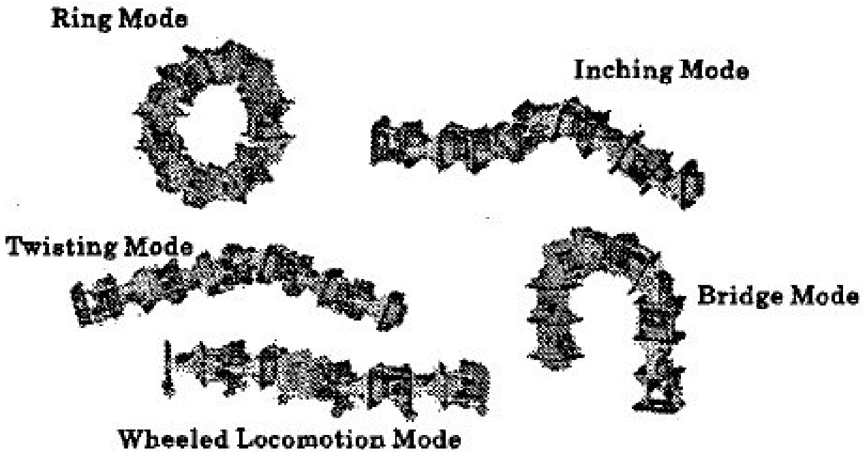


Figure 2.2.8 – Some unnatural snake-like gaits proposed by Kamegawa et. al. [15].

Other types of unnatural snake gaits, like vertical climbing, lean serpentine, sinus-lifting, lift rolling, pedal wave etc. may be added to the definitions. Actually number of possible gaits with snake-like robots is infinite.

Several researchers worked on the analysis and implementation of natural and/or unnatural snake gaits. Hirose formulated the serpentine motion in terms of a parametric curve he called as the “serpenoid curve”. Saito and his coworkers [2] developed a method for optimally efficient 2D serpentine motion. Prautsch and Mita tried to find the best model of snake-like motion in order to minimize the energy needed for motion [14]. Kamegawa and his coworkers proposed several types of motion including ring mode, inching mode, bridge mode, twisting mode, wheeled locomotion mode; they also propsed a genetic algorithm based method to transform robot shape from one mode configuration to another [15]. Chirikjian and Burdick implemented planar grasping with a 30 DOF robot [16]. Ma and his coworkers developed a simulator in order to analyze the creeping motion (usual serpentine motion) of a snake-like robot in 2D [17]. They modeled the robot and environmental dynamics; moreover they tried to point out the problem of optimal creeping motion with respect to the environment. Ma himself by referencing to biological researches analyzed the snake movement forms starting from the characteristics of the muscles, and tried to verify the proposed curves for serpentine motion from biological point of view [18]. Kulali applied the Generalized Approximate Reasoning Based Intelligent Control Architecture (GARIC) to high-level snake-like robot control. In her study, he used a 12-link 3D robot model, she selected some standard gaits and used their linear combination as a movement form for the robot. Gevher [3] modified a traditional path planning algorithm in dynamic environments to snake-like robot high-level control. He used a robot model with 6 links and 10 DOF for demonstration purposes. He also utilized previously defined gait types for the movement of the robot from one position to another.

2.3 Intelligent Control

The key power behind the control talent of human beings is “intelligence”. This concept affected the control engineers starting as the field of “artificial intelligence” (AI) gained popularity among control community. As the result a new era in control engineering has born: “intelligent control”. The aim of intelligent control is designing control systems that exhibit characteristics associated with intelligence in human behavior, such as understanding, learning, reasoning, problem-solving…[6]. Such designs have shown great success in solving problems including high uncertainty and non-linearity. Hence, they are also suitable for application of high-level control of serpentine robots, for which we adapted a learning control methodology. Before introducing the intelligent control techniques inspired in this study, it is necessary to briefly review some approaches to intelligent control.

2.3.1 Artificial Neural Networks – Basics of Artificial Learning.

Artificial neural networks (ANN) are the structures imitating human nervous system. Several artificial neurons are interconnected in this structure mapping inputs to outputs. What makes ANNs distinct as mappings is the fact that an ANN can 'learn' the function it is desired to implement. Using several techniques (usually based on incremental optimization algorithms) 'learning' of any nonlinear function can be achieved by selecting a suitable ANN architecture. This is accomplished by modifying the neuron connection strength values in order to map certain inputs to previously defined outputs. As the result, neural networks are used to make generalizations from given input-output pairs, which is the key idea behind "learning" concept. Applications of ANNs range from image processing to motor control ([20], [21]). In this thesis, artificial neural networks are not used directly. However, the adaptive network structure of the fuzzy controller used is similar to a feed-forward neural network.

2.3.2 Learning Control System Examples

Several controller designs feeding from the generalization capability of neural networks, uncertainty handling and non-linear mapping capability of fuzzy inference systems, and learning from interaction capability of reinforcement learning have been designed and used. In this part of the report, basic theory of reinforcement learning is given and three important examples of such designs are reviewed.

2.3.2.1 Reinforcement Learning

Human beings have also the ability to learn most of his/her behaviors through interaction with the environment. As stated in [22], “when an infant plays, waves its arms, or looks about, it has no explicit teacher, but it has a direct sensorimotor interconnection to its environment. Exercising this connection produces wealth of information about cause and effect, about the consequences of actions, and about what to do in order to achieve goals”. The feedback taken from environments after application of a behavior shapes that behavior in future uses. Reinforcement learning is a computational approach to learn to perform optimally from interaction. Therefore, it is the first kind of example in the learning control structures. The study in this field started in the late 1970s, and research is still in progress.

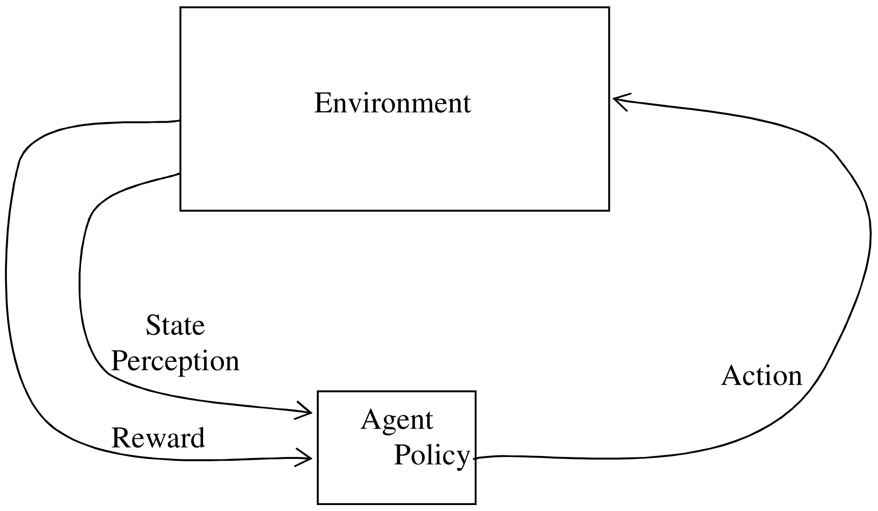


Figure 2.3.1 – Elements of the standard reinforcement learning problem.

In order to give an informal definition, we may use the one in [22] ‘Reinforcement learning is learning what to do –how to map situation to actions- so as to maximize a numerical reward signal’. The learner is not told which actions to take, but instead must discover those that yield the most reward by trying them. A formal definition may be made only after a review of Markov decision processes. A detailed introduction may be found in [22]. Here, only the basics will be given.

A Markovian Decision Process (MDP) is a discrete-time dynamic system defined by the following:

S: Finite discrete state set;
U: Finite discrete action set;
R: The expected primary reinforcements R: $S \text{ x } U \longrightarrow$ R;
P: Transition probabilities P: $S \text{ x } U \text{ x } S \longrightarrow [0,1]$.

At each time step t, the agent observes the current state S_t, and selects an action U_t from the set of possible actions corresponding to that state: $U(S_t)$. When action U_t is triggered in state S_t, the system state at the next time step, $t+1$, is $S_{(t+1)}$ with probability $P_{S_t S_{(t+1)}}(U_t)$. Furthermore, the system emits the reinforcement signal $R(S_t,S_{(t+1)})$, also denoted by r_{t+1} and called the *reward* signal. The action choice in each state is determined by the agent’s policy, which is a mapping from states to actions, and is denoted by $\pi = [\pi(S^1),\pi(S^2),...,\pi(S^{|S|})]$, where $|S|$ is the cardinality of state set S. Following policy π, whenever the agent is in state S_t, it applies action $\pi(S_t)$ (in the case of stationary policies). Each policy has an evaluation function, called the *‘value function’*, V^π qualifying the corresponding agent’s policy, with respect to the reward signal. This evaluation function represents the sum or average of all rewards received during a finite number of time steps (Finite-horizon Model and Average Reward Model respectively). We consider here the most general case, in which the value function represents the expected discounted cumulative rewards received over time when using policy π(Infinite Horizon Model). This value functions are defined for each state $S \in$ S by

$$V^{\pi}(S) = E_{\pi}\left[\sum_{t=0}^{\infty} \gamma^{t} R(S_t, \pi(S_t)) \middle| S_0 = S\right] \qquad (2.1)$$

Where E_{π} is the expected value under the assumption that policy p is always used and S is the starting state (S_0). The discount factor $\gamma\,(\in[0,1])$ is used to weight rewards with respect to time. When $\gamma = 0$, value functions represent solely primary rewards. Otherwise, they denote the expected discounted return over infinite number of time steps (infinite horizon discounted reinforcements). To state in informal way, the value of a state is the expected cumulative reward in the future, where rewards are weighted less as they are away from the current state in time.

Solving an MDP consists of tuning the agent to use an optimal policy π^{*}, one corresponding to the optimal value function, denoted V^{*}. A policy π' is an *improvement* over another policy π if $V^{\pi'}(S) \geq V^{\pi}(S), \forall\, S \in$ S with strict inequality holding for at least one state. A policy is *optimal* in the sense that no policy is an improvement over it. Given a complete and accurate model of the MDP, i.e. (S,*U,R,P*) structures stated above, dynamic programming techniques offer efficient tools for off-line learning of agent's behavior. Value functions of policies are estimated by successive approximations. Let V_n^{π} be the n-step horizon approximation of the evaluation function V^{π}, which is defined iteratively by the equations:

$$V_n^{\pi}(S) = R(S,\pi(S)) + \gamma \sum_{S' \in S} P_{SS'}(\pi(S)) V_{(n-1)}^{\pi}(S') \quad ,\forall\, S \in \text{S}$$

$$(2.2)$$

$$V_1^{\pi}(S) = R(S,\pi(S)) \quad ,\forall\, S \in \text{S}$$

In the case of infinite horizon (infinite step approximation, i.e. the value function itself), value functions must verify the Bellman Equation, which is

$$V^{\pi}(S) = R(S,\pi(S)) + \gamma \sum_{S' \in S} P_{SS'}(\pi(S)) V^{\pi}(S) \quad ,\forall\, S \in \text{S} \qquad (2.3)$$

and the optimal value function is defined by the Bellman Optimality Equation

$$V^{*}(S) = \max_{U}\left\{ R(S,U) + \gamma \sum_{S' \in S} P_{SS'}(U) V^{*}(S') \right\} \quad ,\forall\, S \in \mathrm{S} \qquad (2.4)$$

As seen above, when the MDP model is completely known, the reinforcement learning problem can be solved iteratively. On the other hand, in almost all real-world problems, the MDP model is not completely known. In such cases, methods called Adaptive Dynamic Programming methods are used. They approximate direct Dynamic Programming techniques and constitute two families. For the *model-based* family, a model is approximated via interaction with the system, and Dynamic Programming are then used directly. For the *model-free* family, in which evaluation functions and policies are directly learned during interaction. *Actor-Critic learning* (one model chooses actions, another evaluates the selecting module) and *Q-learning* (both selection and evaluation are performed in the same module) belong to the second family and are also called, by analogy with the control terminology, direct approaches.

An important problem in real-world, concerning the solution of the reinforcement learning problem is the so called 'curse of dimensionality'. As in MDP, the classical Adaptive Dynamic Programming methods deal with discrete input spaces and the state representation often used is a look-up table in which all the states, corresponding values etc. are stored. For example, the best known Actor-Critic Learning method, the Adaptive Heuristic Critic (AHC), is applied in [24] to the cart-pole balancing problem, in which the four-dimensional continuous state-space is discretized. Several examples of such problems can be found in [22], and methods similar to boxing space into discrete regions are used. However, in the case of a large, continuous state space, this representation is intractable. This problem is referred to as *the curse of dimensionality*. Some form of generalization must be incorporated in the state representation. Indeed, we can expect neighboring states to have similar function values. Based on this assumption, various function approximators have been used in conjunction with DP, in spite of the loss of the convergence guarantees that can be found in DP theory. The function approximators often met in Adaptive Dynamic Programming are Cerebellar Model Articulation Controller (CMAC), artificial neural networks, and fuzzy inference systems (FIS) [23].

2.3.2.2 Adaptive Network-based Fuzzy Inference System (ANFIS)

The Adaptive Network-based Fuzzy Inference System (ANFIS) architecture is proposed by Jang in early 1990s [25]. In this architecture, an adaptive multi-layer feed-forward network is used as a fuzzy logic controller, which can adapt its adjustable parameters by using a gradient descent algorithm, named Temporal Back Propagation (TBP). The fuzzy logic controller is embedded into the network by making each layer of the network an implementation of a Takagi-Sugeno type (i.e. with linear defuzzification stage) fuzzy logic controller stage.

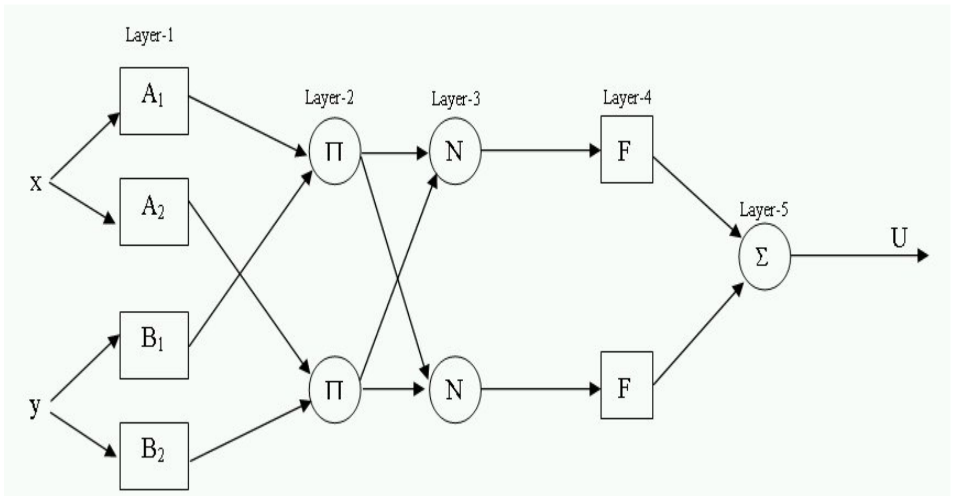


Figure 2.3.2 – Structure of the adaptive multi-layer feed-forward network used for implementation of a Takagi-Sugeno type fuzzy controller, shown on a very simple example with two states, four fuzzy sets, and two fuzzy rules in the rule base.

There are 5 layers in the network, as seen in Figure 2.3.2. The first layer implements the fuzzification step. Here, every input is mapped to the fuzzy membership function, and every node corresponds to a different fuzzy set. Any kind of continuous and partially differentiable membership function can be used. In the second layer, the fuzzified state values are used to evaluate the activation strength of each rule in the fuzzy inference rule base. Hence, there are second layer nodes as there are fuzzy inference system rules. In the third layer, only a normalization procedure is performed and activation strength of the corresponding rule is divided to the total activation strength of all the rules. In the fourth layer, Takagi-Sugeno type defuzzification is performed and every node outputs a control action, which has a weight proportional to the activation strength of the corresponding rule. The fifth node only performs a simple summation and calculates the total control action as the combination of individual control actions proposed by each of the rules.

In the network, nodes shown with a square has adjustable parameters, while nodes shown with a circle perform fixed mathematical operations. In the first layer, the parameters of fuzzy sets are adaptive; and in the fourth layer the defuzzification line coefficients of each rule are adaptive. Therefore, we can infer that ANFIS is a proposal for solution of automatic tuning of fuzzy controller parameters, which is known as a tough problem in fuzzy control community.

Learning (tuning) of fuzzy controller parameters occurs with a mechanism shown schematically in Figure 2.3.3. At every discrete time step, the states at one step before and the inputs to the plant determine the next states as given in equation (2.5) below, for a time-dependent system.

$$\overline{X}_k = f(\overline{X}_{(k-1)}, U_{(k-1)}, (k-1)) \tag{2.5}$$

The desired trajectory of states is assumed to be known *a priori*, and an error measure is calculated using the difference between the actual trajectory and the

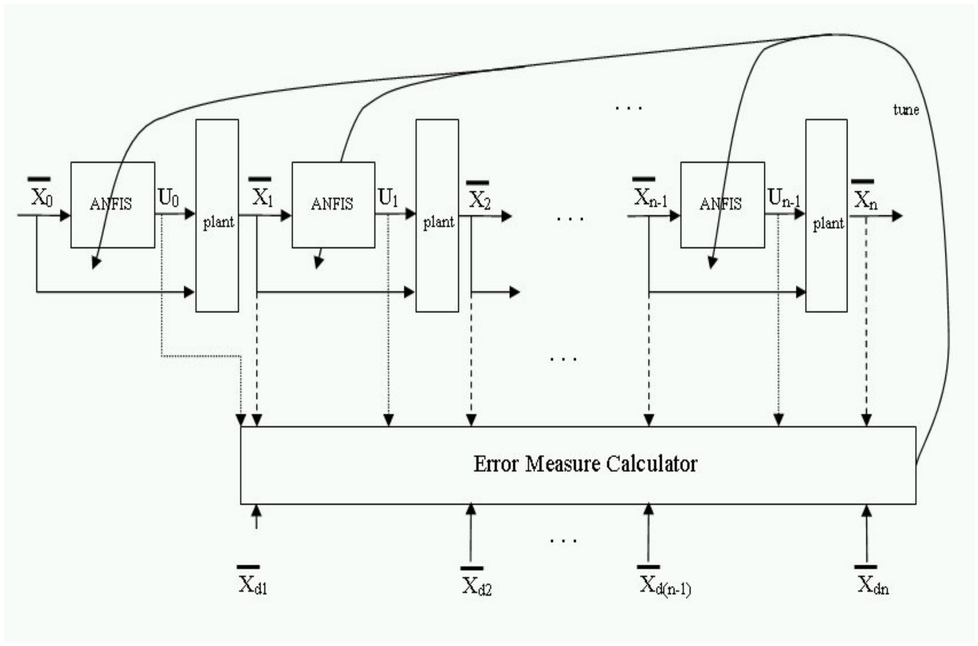


Figure 2.3.3 – Schematic diagram showing the dynamics of a system with ANFIS controller architecture.

desired trajectory as given in equation (2.6). A term including the total control effort is added to this error in order to optimize the control effort versus the state trajectory cumulative error. In other words, if this term is not added, very good results may be taken for the error measure, but with an undesirably high control effort, possibly.

$$E = \sum_{k=1}^{n} \left\| \overline{X}(k) - \overline{X}_d(k) \right\|^2 + \lambda \cdot \sum_{k=0}^{n-1} \left\| U(k) \right\|^2 \qquad (2.6)$$

Finally, the adjustable parameters of the network are updated according to the classical gradient descent rule as in equation (2.7), where an adaptive step size parameter is used for fast and reliable convergence of parameters.

$$\Delta\alpha = -\eta \frac{\partial E}{\partial \alpha} \qquad (2.7)$$

The gradient in the above equation is found using chain rule, by back-propagating the error signal through time (this is the reason for the name Temporal Back Propagation - TBP).

Multi-layer feed-forward neural networks are a special case of multi-layer adaptive networks. Moreover, temporal back propagation is nothing but the usual back-propagation algorithm, which is slightly modified as network layers are distributed in time. This architecture was a wise idea in self-tuning controllers and inspired many researchers. It has found several applications ranging from simple control tasks to bipedal locomotion. Even there is a built-in ANFIS implementation in MATLAB software package. Although the ANFIS approach is very versatile, it is not very suitable for our application. The calculation of error measure includes the desired trajectory of states, which is not completely known in a very complex system.

2.3.2.3 Generalized Approximate-Reasoning Based Intelligent Controller (GARIC)

The ANFIS architecture uses a kind of supervised learning; however in most of the systems the actual desired outcome of an action, which is a needed input for the application of this type of learning, is not known directly. Hence, a more suitable learning method should be employed in such situations. This can be accomplished by using reinforcement learning techniques to adapt the parameters of the fuzzy controller in the system. The Generalized Approximate-Reasoning Based Intelligent Controller (GARIC) architecture is a proposition for such a solution, designed by Berenji and Khedkar in early 1990s [26]. In this architecture two different neural networks are employed. One of these networks is used to evaluate the value function of the current state (i.e., it is the *critic*) and named as the Action Evaluation Network (AEN); and the other network is used to calculate the control action for a given set of state feedbacks (i.e. it is the *actor*) and named as the Action Selection Network (ASN). Since both of these networks are made up of neural networks, they have tunable parameters. These parameters are updated using a reinforcement signal from the plant (which is expected to be only a binary error signal, in general), which is a way of indirect learning. Moreover, the standard back-propagation algorithm of feed-forward neural networks is also employed. As the result, the Action Evaluation Network learns to be a good implementer of the state value function, and the Action Selection Network learns to be a good implementer of the sensorimotor, i.e. state to control action, mapping. In order to search the state-space, and better perturb the system in order to avoid local solutions, a Stochastic Action Modifier (SAM) module is also employed, which changes the action recommended by ASN before applying it to the system. The schematic diagram showing the general structure of the GARIC architecture is given in Figure 2.3.4.

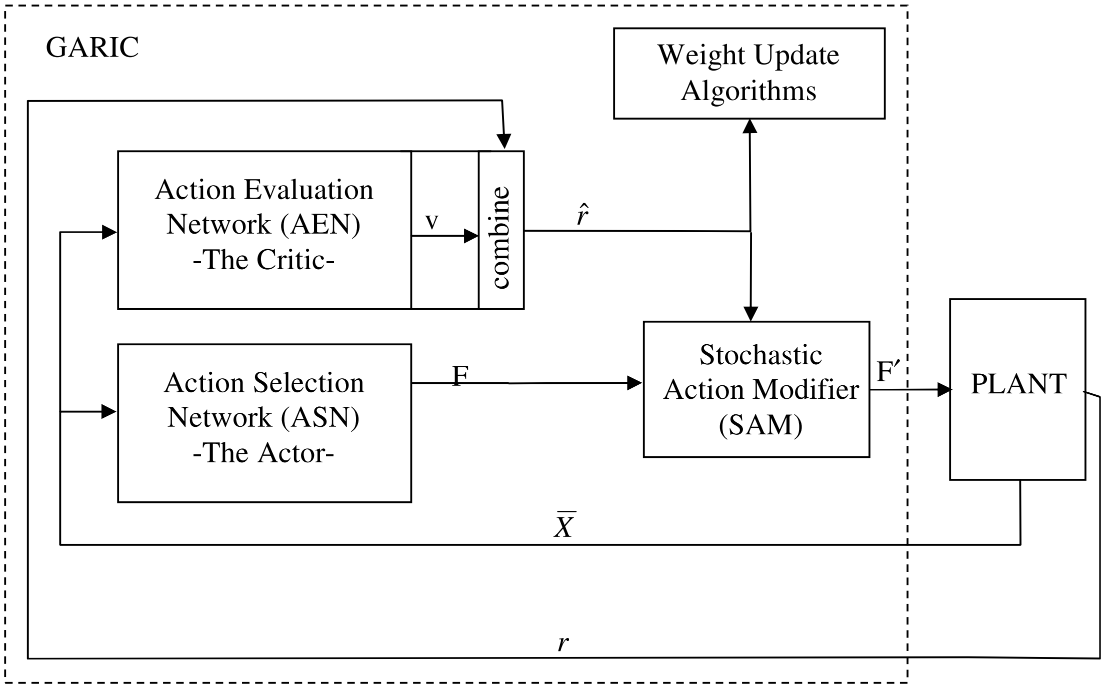

GARIC
Weight Update Algorithms
Action Evaluation Network (AEN) -The Critic-
v
combine
$\hat{r}$
Action Selection Network (ASN) -The Actor-
F
Stochastic Action Modifier (SAM)
F′
PLANT
$\overline{X}$
r

Figure 2.3.4 – Schematic diagram showing the GARIC architecture.

A detailed schematic of AEN module is given in Figure 2.3.5. As seen in this figure, AEN is a usual feed-forward neural network, that implements the state to value mapping. This mapping is not known initially, as a common part of the reinforcement learning problem, and AEN is trained on-line in order to improve its correctness at each time step.

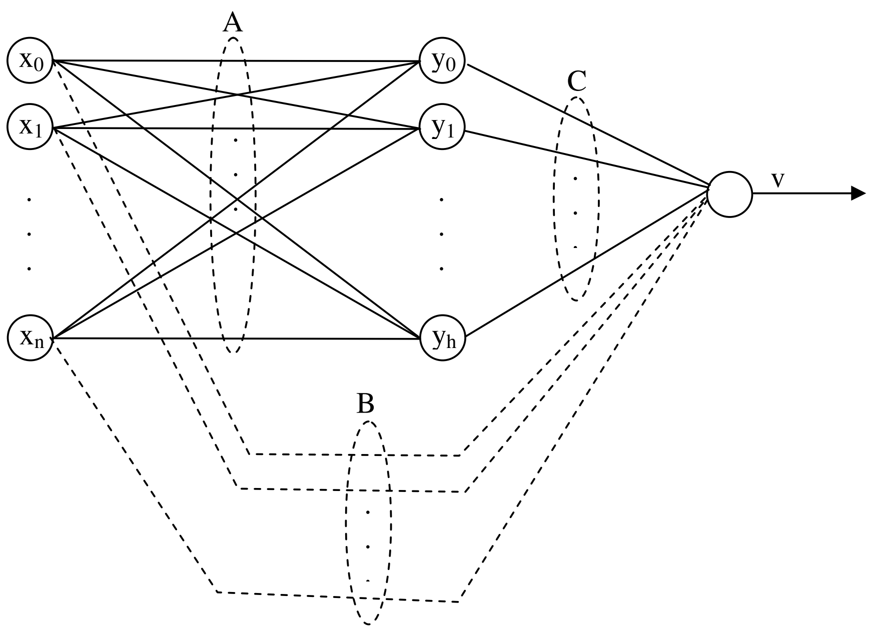


Figure 2.3.5 – Schematic diagram showing the AEN module.

In [26] AEN is proposed as a standard two layer feed-forward neural network. However, any suitable neural network architecture may be used as an action evaluator. Going over the network structure proposed in [26], the input layer consists of the states (and probably bias terms, which are not shown explicitly in the figure). The hidden layer outputs are calculated using a sigmoid activation function at each time step as

$$y_i(t,t+1) = g\left(\sum_{j=1}^{n} a_{ij}(t) \cdot x_j(t+1)\right) \qquad (2.8)$$

where, g(x) is the sigmoid function given by

$$g(x) = \frac{1}{1+e^{-x}} \qquad (2.9)$$

The hidden layer outputs are calculated using the current states at time t+1, and the A coefficients, which are not updated yet. The neuron at the output layer gives the value function estimate of the current state by combining its inputs as in equation (2.10) below

$$v(t,t+1) = \sum_{i=1}^{n} b_i(t) \cdot x_i(t+1) + \sum_{i=1}^{h} c_i(t) \cdot y_i(t,t+1) \qquad (2.10)$$

using this value estimate an internal reinforcement signal, $\hat{r}$, is calculated as

$$\hat{r} = \begin{cases} 0 & \text{,if start state} \\ r(t+1) - v(t,t) & \text{,if failure state} \\ r(t+1) + \gamma v(t,t+1) - v(t,t) & \text{,otherwise} \end{cases} \qquad (2.11)$$

which is a standard update procedure in temporal difference learning, with value of failure state is defined to be 0, without any calculation. After calculating internal reinforcement signal, update of B and C vectors are performed using a standard reward/punishment scheme; whereas update of vector A is performed using a modified back propagation algorithm. As the result, the coefficients are updated at each time step, in order to give low values to *bad states* and high values to *good states* in the steady state.

The ASN structure is similar to the adaptive network structure used in ANFIS. However, this time, the network is defined as a feed-forward neural network. At layer 1, the inputs are gathered into the controller and fed to the related fuzzy set membership functions, implemented as neurons of the second layer. In the second layer, neurons that are similar to a radial basis function implementation are used to perform the fuzzy set matching of the inputs [26]. A function of three parameters (similar to a triangular function) is employed.

$$\mu_{C_V, S_{VL}, S_{VR}}(x) \qquad (2.12)$$

Here, V represents the fuzzy set and C_V, S_{VL}, S_{VR} parameters are used to represent the center, left spread and right spread values respectively. In layer 3, fuzzy rules are implemented and strength of each rule is calculated using a softmin operator so as to implement the conjunction of all the antecedent conditions in a rule, as follows

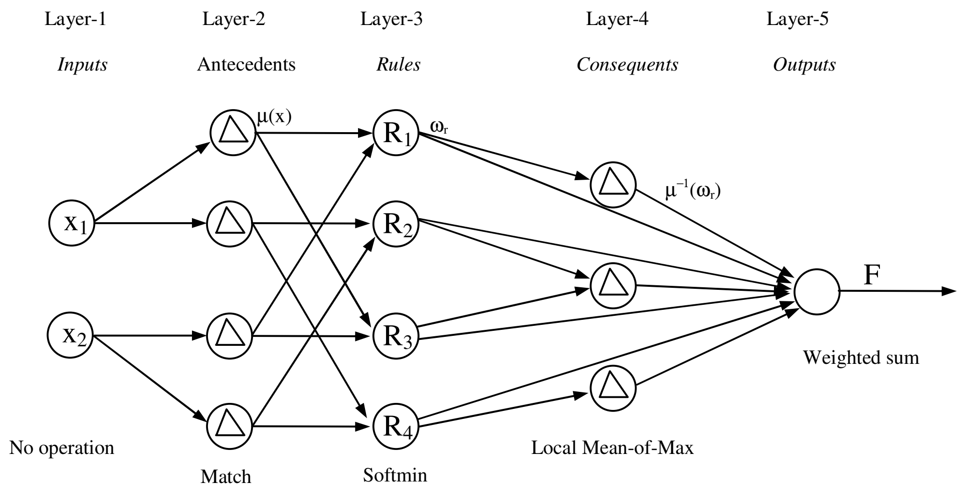


Figure 2.3.6 – Schematic diagram showing the ASN module for a controller with 2 state feedbacks, 4 antecedent fuzzy sets, 4 rules and 3 consequent fuzzy sets.

$$O_R = \omega_r = \frac{\sum_i \mu_i e^{-k\mu_i}}{\sum_i e^{-k\mu_i}} \qquad (2.13)$$

Where μ_i is the degree of match between a fuzzy label occurring as one of the antecedents of rule *r* and the corresponding input variable. *k* is the parameter controlling the hardness of the softmin operator. After processing of layer 3, all rules' firing strengths are determined. In the fourth layer, consequent parts of these rules are resolved, using a local mean-of-max (LMOM) operation on the consequent fuzzy sets, and a local defuzzification is performed for each of the output fuzzy sets. Hence, we may represent the operation of this layer as

$$\mu^{-1}{}_{C_V, S_{VL}, S_{VR}}(x) \qquad (2.14)$$

where V represents a specific consequent label and C_V, S_{VL}, S_{VR} parameters are used to represent the center, left spread and right spread values of the fuzzy membership function of this set. In layer 5, the defuzzified output values of each output fuzzy set are combined to generate a scalar output. This is accomplished by getting the firing strengths of the rules from the layer 3, and using these as weights of a weighted average. As the result, ASN outputs the following value for the output

$$F = \frac{\sum_r \omega_r \mu^{-1}(\omega_r)}{\sum_r \omega_r} \qquad (2.15)$$

This output is not directly applied to the plant. It is modified in the stochastic action modifier (SAM) first. This is accomplished by choosing the final control action F′ randomly from a Gaussian random variable, whose mean is F and whose standard deviation increases with decreasing internal reinforcement. This operation is performed, in order to be able to conquer the state space evenly, and escape from states when the internal reinforcement is small.

The adaptive parameters in ASN are the fuzzy set membership function parameters of layers 2 and 4. Tuning of these parameters is performed using the relationship between the ASN output and the value function estimate, *v*, of AEN. If we represent the tunable parameters of ASN in a vector, *p*, then ASN represents a mapping between the states and the control action, tunable over this vector, i.e. $F = F_p(X)$. The intent of computing F is to maximize *v*, so that the system ends up in a good state and avoids failure. Hence, we should maximize *v* as a function of *p*. Hence we may use the classical gradient descent algorithm so as to minimize –*v*, which results in the classical back-propagation algorithm of neural networks, giving the update of vector *p*, at each time as

$$\Delta p = \eta \frac{\partial v}{\partial p} = \eta \frac{\partial v}{\partial F} \cdot \frac{\partial F}{\partial p} \qquad (2.16)$$

The gradient of *v* with respect to *F* may be calculated using some approximations, and the gradient of *F* with respect to *p* using the chain rule. The details of these calculations may be found in [26].

In summary GARIC architecture is proposed in order to tune a fuzzy logic controller using reinforcement learning techniques, in the existence of only a binary failure feedback from the plant. In this architecture, both the fuzzy logic controller and the state value function evaluator are embedded into neural networks. Parameters of these networks are updated at each time step according to the calculated internal reinforcement. This architecture has found implementations in several problems. Even, an implementation of this architecture for gait selection of snake-like robots was studied [19], in which the output of ASN is the vector joint torques. Although this architecture is successful in several respects, it needs several modifications in order to perform an effective implementation. Moreover, this architecture is rather old, and new architectures are continuously being proposed by several researchers, one of which is explained in detail in the next sub-section.

2.3.2.4 Fuzzy Actor-Critic Learning (FACL) Controller

Fuzzy Actor Critic Learning (FACL) architecture is proposed by Jouffe in late 1990s. The idea behind it is essentially the same for other similar architectures: Tuning fuzzy controller parameters using learning techniques. This architecture is implemented and used as the controllers of various jobs in this thesis; hence it is explained in detail in the following paragraphs.

The fuzzy controller architecture used in FACL is similar to ones used in other architectures; however, it has some predefined specifications. First of all, the fuzzy clustering of the input variables must be *strong*. Secondly, for the defuzzification part, output fuzzy sets are all crisp. Thirdly, the input fuzzy sets are assumed to be known *a priori*, and only the output parameters are tuned during learning. Finally, the fuzzy rules are combinations of all input fuzzy sets “AND”ed.

As seen in the structure of the fuzzy inference system used in FACL (see Figure 2.3.7), the controller module uses N rules of the form

$$R_i\text{: IF } S_1 \text{ is } L_1^i \text{ AND ... AND } S_{N_I} \text{ is } L_{N_I}^i \text{ THEN } Y_1 \text{ is } O_1^{\mathrm{i}} \text{ AND ... AND } Y_{N_{\mathrm{o}}}^{\mathrm{i}} \text{ is } O_{N_{\mathrm{o}}}^{\mathrm{i}} \qquad (2.17)$$

where

R_i	i th rule of the rule base
S	input vector
$\mathrm{S} = \mathrm{S}_1\mathrm{x}\mathrm{S}_2\mathrm{x}\ldots\mathrm{S}_{\mathrm{N}_\mathrm{I}}$	universe of discourse for input variables
L_{j}^i	linguistic term (fuzzy label) of input variable S_j in rule R_i; its membership function is denoted by $\mu_{L_j^{\mathrm{i}}}$
$(Y_m)_{m=1,\ldots,N_O}$	N_O output variables.
L_j	linguistic term of output variable Y_j in rule R_i.

As usual in the similar structures discussed so far, the input layer is used to gather input values, from the sensors. These values are fuzzified using the related fuzzy sets in the second layer. In the third layer, the combination of the input variables is performed and rule truth values are evaluated. Finally, in the fourth layer, rule outputs are combined to generate quantitative output values.

In FACL the fuzzy sets for the N_I input variables are assumed to be exactly known a priori. Moreover, since these inputs are states of the system to be controlled, the fuzzy sets should be describing all useful states without ambiguity. The types of

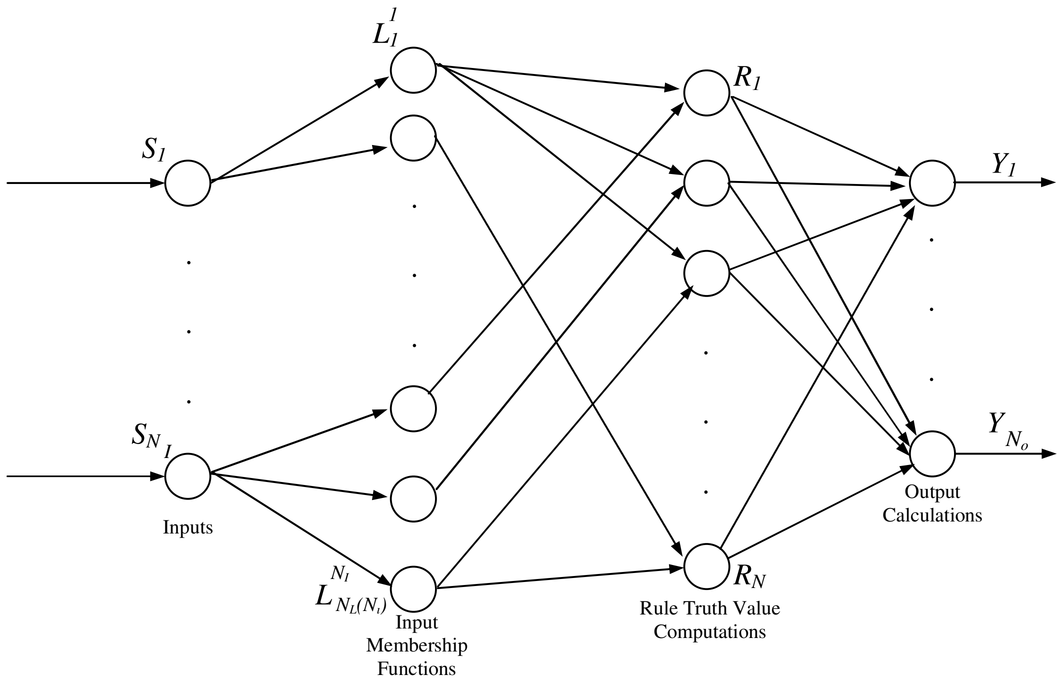


Figure 2.3.7 – Structure of the fuzzy inference system used in FACL.

fuzzy sets used are trapezoidal and triangular. Moreover, strong fuzzy partitioning is employed, which means

$$\sum_{j=1}^{N_L(n)} \mu_{L_n^j}(S_n) = 1 \qquad ,\forall\ S_n \in \mathrm{S_n} \tag{2.18}$$

where $N_L(n)$ denoted the number of fuzzy sets used to represent the universe of discourse of input variable S_n. Such a partitioning implies that there are no more than two fuzzy sets activated (with $\mu > 0$) for an input value, and any value of an input activates at least one fuzzy set. Hence the number of activated fuzzy sets for an input variable is at least one, and at most two. The user must define all the input fuzzy membership functions accordingly and fix their parameters. Such a partitioning is shown in Figure 2.3.8. As seen in this figure, the membership degree of an input value S_n for a label $L_n(v_l,v_r,s_l,s_r)$ is defined by the following expression

$$\hat{r} = \begin{cases} \max\left(0.0, 1.0 - \dfrac{(S_n - v_r)}{s_r}\right) & , S_n > v_r \\ \max\left(0.0, 1.0 - \dfrac{(v_l - S_n)}{s_l}\right) & , S_n < v_l \\ 1.0 & , \text{otherwise} \end{cases} \qquad (2.19)$$

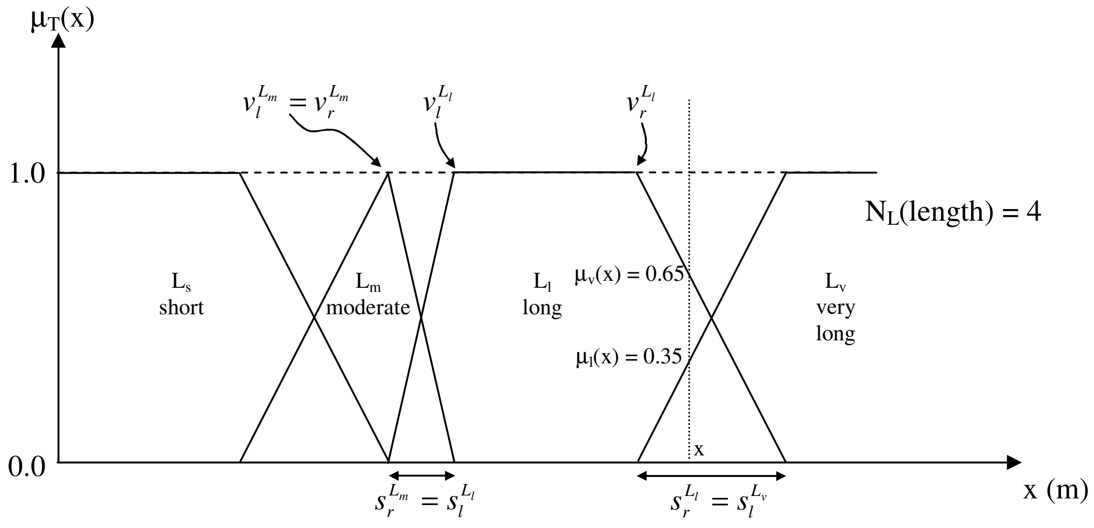


Figure 2.3.8 – Example of a strong fuzzy partitioning using an example fuzzy variable "distance".

For the rules part, all "AND"ed combinations of the input fuzzy variables are used. Hence, the number of the rules, N, is determined by the number of total linguistic input variables, N_I, and the number of fuzzy sets defined for each input, N_L, as follows

$$N = \prod_{i=1}^{N_I} N_L(i) \qquad (2.20)$$

Each of these rules has N_O corresponding conclusions, $(O_m^i)_{m=1,\dots,N_O}^{i=1,\dots,N}$. All fuzzy sets for these outputs are defined as crisp sets. In mathematical terms, we may state

$$\mu_{O_m^i}(Y_m) = \begin{cases} 1.0 & , Y_m = o_m^i \\ 0.0 & , \text{otherwise} \end{cases} \qquad (2.21)$$

After computing the membership function values for all inputs, then we should find the truth values of the rules. Since we use “AND”ing of all combinations of input variables, for a given rule R_i the truth value is computed using a T-norm operation. If we implement the T-norm operation using usual product operation we get

$$\alpha_{R_i}(S) = \prod_{j=1}^{N_i} \mu_{L_j^i}(S_j) \qquad (2.22)$$

Hence we compute the truth value of each rule having a non-zero precondition part, and denote this as the activated rule set *A*. Note that the maximum number of rules in *A* at a given instant is 2^{N_I} and the minimum number of activated rules is 1, due to the strong partitioning used. Given the above rule truth values, the FIS outputs are calculated by

$$Y_m(S) = \sum_{R_i \in A} \alpha_{R_i}(S) o_m^i \quad , m = 1,...,N_O \qquad (2.23)$$

In other words, the outputs offered by each rule are weighted by rule truth values and summed up.

As stated before, in FACL learning occurs in only the conclusion part of the FIS. The number and positions of the input fuzzy labels are assumed to be set *a priori*. The learning in FACL is a modified version of Sutton’s Adaptive Heuristic Critic (AHC) algorithm [24], which is a reinforcement learning method. The learner in FACL is a fuzzy inference system, whereas it is a look-up table in AHC. In actor-critic learning there are two basic components, the actor and the critic (similar to GARIC, and usual in modified AHC algorithms). The critic represents the state value function and adapts itself at each learning time step. Whereas, the actor implements the fuzzy inference system described above.

The state is represented by a vector Φ, which is directly related to the truth values of the rules, due to the rule generation scheme employed. The critic is modeled by the state related value vector *v*. The actor has discrete action sets for each state (hence for each rule). Each discrete action in such a set has a corresponding weight that determines the probability of choosing the action in that rule. The continuous action performed by the actor is then the weighted sum of the actions elected in the rules describing the current state.

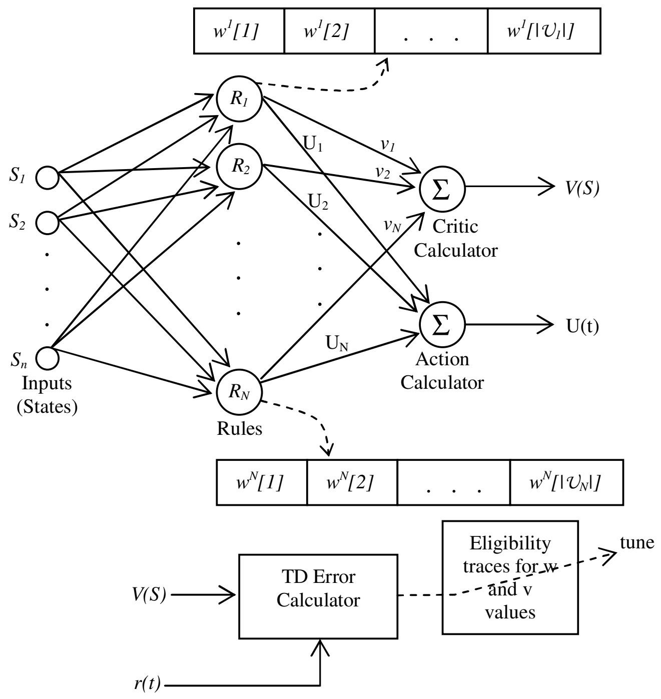


Figure 2.3.9 – The FACL architecture.

The structure of the FACL controller is given in Figure 2.3.9. The critic calculates the estimated value of an input state, as a linear combination of the rule strengths, at time step *t*, as

$$V_t(S_t) = \sum_{R_i \in A_t} v_t^i \cdot \alpha_{R_i} = v_t \bullet \Phi_t^T \qquad (2.24)$$

Using reinforcement learning approximation, the error calculated at time step *t+1* (the difference between the actual value of the optimal state value function and the estimation at time step t) is, then

$$\varepsilon_{t+1} = V^*(S_t) - V_t(S_t) \qquad (2.25)$$

However, actual value of this error is not known (otherwise, we would know the value of V^*, estimation of which is the basic problem in reinforcement learning). On the other hand, this error term may be estimated using the quantities available at time step *t+1*. This new error term is called the TD (temporal difference) error [22] and calculated as

$$\tilde{\varepsilon}_{t+1} = r_{t+1} + \gamma \cdot V_t(S_{t+1}) - V_t(S_t) \qquad (2.26)$$

Remembering equation (2.4) given before

$$V^*(S) = \max_U \left\{ R(S,U) + \gamma \sum_{S' \in S} P_{SS'}(U) V^*(S') \right\} \quad ,\forall\, S \in \mathrm{S}$$

and denoting the action satisfying this equation as U^*, we get

$$V^*(S) = \left\{ R(S,U^*) + \gamma \sum_{S' \in S} P_{SS'}(U^*) V^*(S') \right\} \quad ,\forall\, S \in \mathrm{S} \qquad (2.27)$$

Hence, we are approximating the term $R(S,U^*)$ using r_{t+1} (instant reward feedback), and the summation term (delayed rewards) by $\gamma.V_t(S_{t+1})$. After calculating $\tilde{\varepsilon}$, we may use the TD learning rule [22] to tune the vector used for the value estimate as follows
where β is the critic learning rate.

$$v_{t+1} = v_t + \beta \cdot \tilde{\varepsilon}_{t+1} \cdot \Phi_t \qquad (2.28)$$

The actor part is used to code the policies. There are several rules (total number given by equation (2.20) above) and each rule has discrete action sets, whose elements are defined as fuzzy sets as in equation (2.21). Each rule R_i uses a weight vector, entries of which are associated with the elements of the discrete action set (this is schematically shown in Figure 2.3.9). Every rule in A_t proposes a local action from the discrete action set in order to be used as an element of the global action, and these weights participate in the selection of this action. In each activated rule a competition between the discrete actions is hold using an ε-greedy function defined as

$$\varepsilon - Greedy_{\mathcal{U}}(w) = \arg\max_{U \in \mathcal{U}} (w(U) + \eta(U) + \rho(U)) \qquad (2.29)$$

where $\mathcal{U}$ is the set of discrete actions in the rule being considered. In this ε-greedy function $w(U)$ term is used for exploitation, $\eta(U)$ is used for undirected exploration and $\rho(U)$ is used for directed exploration. Undirected strategies are related to random walk (similar to SAM in GARIC), and directed strategies memorize exploration-specific knowledge and tend to explore previously unexplored states. The exploitation term is the sole term used in steady-state, after the learning is complete and V^* is approximated. With this ε-greedy function, the resultant global action applied to the system is given by

$$U_t(S_t) = \sum_{R_t \in A_t} \varepsilon - Greedy_{\mathcal{U}(i)}(w_t^i) \cdot \alpha_{R_i} \qquad (2.30)$$

In words, the global action applied by the actor element is the weighted sum of the individual discrete actions proposed by the active rules. The weights used are the activation strengths of the rules. Hence, a global action continuous in the state space is applied to the system.

For the calculation of the exploratory terms the following method is proposed in [23].

For the undirected exploratory term, use a vector of random values Ψ each term of which is associated with a discrete action, and selected from the same exponential distribution; then scale this vector so as to normalize with respect to the range of *w* values in use by

$$s_f = \begin{cases} 1 & \text{, if } \max(w) = \min(w) \\ \dfrac{s_p(\max(w) - \min(w))}{\max(\Psi)} & \text{, otherwise} \end{cases} \qquad (2.31)$$

$$\eta = s_f \cdot \Psi$$

where s_p is referred to as the "noise size", with respect to the range of qualities, and s_f is the corresponding scaling factor used for normalization.

The directed exploratory term should increase the probability of selecting actions that have rarely been elected; this is achieved by using

$$\rho(U) = \frac{\theta}{e^{n_t(S_t,U)}} \qquad (2.32)$$

where θ represents a positive factor to weight the directed exploratory term, and $n_t(S_t,U)$ is the number of previous time steps in which action U is selected. Since, several rules may use same action, and with different weights; this number is approximated by

$$n_t(S_t,U) = \sum_{R_i \in A_t} n_t(U^i) \cdot \alpha_{R_i} \qquad (2.33)$$

where $n_t(U^i)$ is is the number of previous applications, at time step t, of action U in rule R_i. This term may be calculated recursively, and values may be stored in a look-up table.

Learning in the actor occurs as an update of the weights of exploitation terms using the TD error, which is performed as

$$w_{t+1}^i(U_t^i) = w_t^i(U_t^i) + \tilde{\varepsilon}_{t+1} \cdot \alpha_{R_i} \qquad (2.34)$$

Hence, weight of an action participating in a better state is increased by a factor proportional to the strength of the related rules in the global action.

During the learning process of the parameters *v* and *w*, two key concepts are employed. One of these concepts is the eligibility traces used for solving the temporal credit assignment problem, and the other one is the meta learning rule, which is used for adapting learning rates so as to avoid instabilities in learning.

The parameter update formulas given in equations (2.28) and (2.34) correspond to *TD(0)* [22]. In this algorithm no delayed rewards are considered, and only the parameters corresponding to the rules activated at time *t* are updated. However, the states visited and the actions taken in the previous time steps are also effective in ending with the current conditions. Hence they must also be altered according to their effectiveness in the result, which is determined by the distance in time to the current time, *t*. A method for solving this problem is using data structures called *"eligibility traces"*. In the eligibility traces for every state, and action an *eligibility* value is employed, these values are stored in a vector, say $\overline{\Phi}_t$, which is updated by using the accumulating eligibility trace rule, for the critic, we get:

$$\overline{\Phi}_t = \sum_{n=0}^{t} (\gamma \cdot \lambda)^{(t-n)} \Phi_n = \Phi_t + \gamma \cdot \lambda \cdot \sum_{n=0}^{t-1} (\gamma\lambda)^{((t-1)-n)} \cdot \Phi_n = \Phi_t + \gamma \cdot \lambda \cdot \overline{\Phi}_{t-1} \quad (2.35)$$

where the recency factor λ (in [0,1]) is known as the *eligibility rate.* Hence, the critic eligibility traces are accumulating traces, which are increased as the related rule is activated (without any limit), and decreased as time passes.

For the actor eligibility trace, for each of the *U* values, an eligibility value must be stored. Let $e_t^i(U^i)$ be the trace associated with discrete action U_i of rule R_i at time step *t*, we have

$$e_t^i(U^i) = \begin{cases} \lambda' \cdot e_{(t-1)}^i(U^i) + \Phi_t^i & \text{, if } U^i = U_t^i \\ \lambda' \cdot e_{(t-1)}^i(U^i) & \text{, otherwise} \end{cases} \quad (2.36)$$

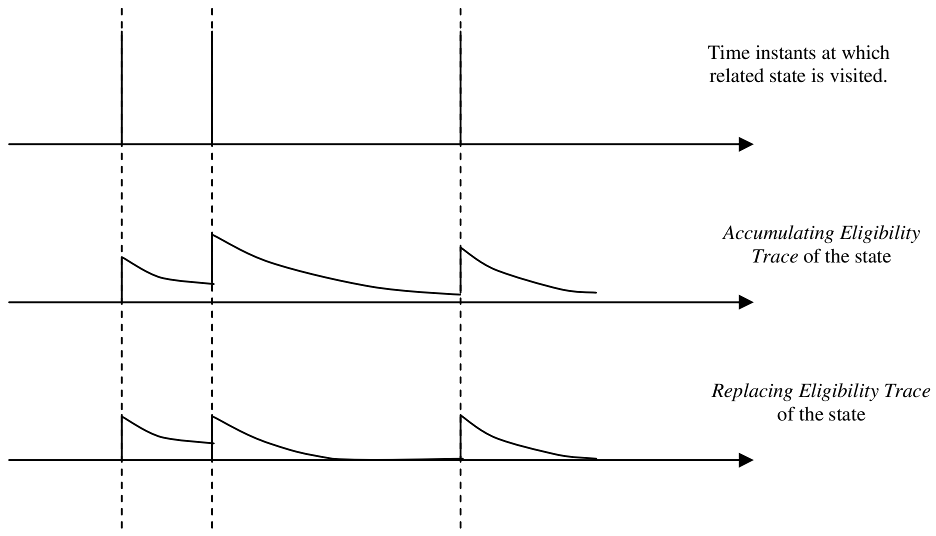

Time instants at which related state is visited.
Accumulating Eligibility Trace of the state
Replacing Eligibility Trace of the state

Figure 2.3.10 – Visualization of *accumulating eligibility trace* vs *replacing eligibility trace* [23].

Where λ' represents the actor recency factor. Similar to the critic recency factor, as an action participates in the global action, its eligibility trace is increased by a factor proportional to the activation strength of the related rule.

As a result; the updates for the *v* and *w* terms given in equations (2.28) and (2.34) respectively, are modified as follows

$$\begin{aligned} v_{t+1} &= v_t + \beta \cdot \tilde{\varepsilon}_{t+1} \cdot \overline{\Phi}_t \\ w_{t+1} &= w_t + \tilde{\varepsilon}_{t+1} \cdot e_t \end{aligned} \qquad (2.37)$$

In practical implementation, eligibility traces are stored for only a finite number of recent states and actions, by using a threshold for the traces; and ignoring the traces less than this threshold.

As we stated, the second key issue is about the learning rate updating. In order to prevent oscillations in estimating the value function *V*, the learning rates are updated by using four heuristics. First of all every parameter should have its own learning rate. Secondly, every learning rate should be allowed to vary over time. Thirdly, when the derivative of a parameter possesses the same sign for several consecutive time steps, its learning rate should be increased. Finally, when the parameter sign alternates for several consecutive time steps, its learning rate should be decreased. In order to implement these heuristics, the delta-bar-delta rule [27] is employed. In this rule, the change in learning rate at each time step is given by

$$\Delta\beta_t^i = \begin{cases} \kappa & \text{,if } \bar{\delta}_{t-1}^i \cdot \delta_t^i > 0 \\ -\sigma \cdot \beta_t^i & \text{,if } \bar{\delta}_{t-1}^i \cdot \delta_t^i < 0 \\ 0 & \text{,otherwise} \end{cases} \qquad (2.38)$$

Where β_t^i is the critic learning rate of rule R_i at time step t; $\delta_t^i = \tilde{\varepsilon}_{t+1} \cdot \overline{\Phi}_t^i$ is a term used to integrate the eligibility trace; and $\bar{\delta}_t^i = (1-\varphi)\delta_t^i + \varphi\bar{\delta}_{t-1}^i$ represents the geometric average. Hence, the critic update rule given in equation (2.37) may be further modified to show that the learning rate is time-dependent

$$v_{t+1} = v_t + \beta_t \cdot \tilde{\varepsilon}_{t+1} \cdot \overline{\Phi}_t \qquad (2.39)$$

In summary Fuzzy Actor-Critic Learning (FACL) architecture is proposed by Jouffe in late 1990s; it is a modified form of Sutton's AHC algorithm; a fuzzy inference system (FIS) is employed to approximate the value function; linear combination of discrete actions are used to generate actions which are continuous in state space; temporal difference (*TD(λ)*) methods are employed in tuning the actor and critic parameters; eligibility traces and meta learning rule are used for solving temporal credit assignment and adaptive learning rate problems. In action selection a function which is a combination of exploitory, directed exploratory, and undirected exploratory terms is employed. This architecture is shown to be more successful than its counterparts, similar to GARIC and ANFIS (see [23]). Six algorithmic steps for the implementation of FACL is given in [23], and FACLs are implemented as MATLAB M-files in this thesis work.

2.3.3 Optimization via Genetic Algorithms

An interesting problem in snake-like robot study is the generation of gaits, which is defined as the sequence of joint motions causing net displacements of the whole body. In Dowling's study [41] this is achieved by directly searching the motion space for a desired displacement of the body. A similar approach based on genetic algorithms is used in this work. Hence, a basic introduction of this topic is also given in this part, for the sake of completeness of the report.

"Genetic Algorithms" (GA), first introduced by John Holland of University of Michigan in the mid 1970s, is a branch of *evolutionary computing*. This topic is an inspiration from Darwin's theory of evolution. Genetic algorithms uses concepts from the genetics of living creatures, and optimization problems are tried to be solved using a similar approach. So let's first introduce some concepts from biology.

All living organisms consist of *cells*. In each cell there is the same set of *chromosomes*. Chromosomes are strings of *DNA* and serve as a model for the whole organism. A chromosome consists of *genes*, blocks of DNA. Each gene encodes a particular protein. Basically, it can be said that each gene encodes a *trait*, for example color of eyes. Possible settings for a trait (e.g. blue, brown) are called *alleles*. Each gene has its own position in the chromosome. This position is called *locus*. Complete set of genetic material (all chromosomes) is called *genome*. Particular set of genes in genome is called *genotype*. The genotype is with later development after birth base for the organism's *phenotype*, its physical and mental characteristics, such as eye color, intelligence etc. During *reproduction*, *recombination* (or *crossover*) first occurs. Genes from parents combine to form a whole new chromosome. The newly created offspring can then be mutated. *Mutation* means that the elements of DNA are a bit changed. This changes are mainly caused by errors in copying genes from parents. The *fitness* of an organism is measured by success of the organism in its life (survival).

If we are solving a problem, we are usually looking for some solution that will be the best among others. The space of all feasible solutions (the set of solutions among which the desired solution resides) is called *search space* (also *state space*). Each point in the search space represents one possible solution. Each possible solution can be "marked" by its value (or *fitness*) for the problem. With GA we look for the best solution among a number of possible solutions - represented by one point in the search space. Looking for a solution is then equal to looking for some extreme value (minimum or maximum) in the search space. At times the search space may be well defined, but usually we know only a few points in the search space. In the process of using GA, the process of finding solutions generates other points (possible solutions) as evolution proceeds. The problem is that the search can be very complicated. One may not know where to look for a solution or where to start. There are many methods one can use for finding a suitable solution, but these methods do not necessarily provide the best solution. Some of these methods are hill climbing, tabu search, simulated annealing and the genetic algorithm. The solutions found by these methods are often considered as good solutions, because it is not often possible to prove what the optimum is.

Genetic algorithm begins with a set of solutions (represented by chromosomes) called *(initial) population*. Solutions from one population are taken and used to form a new population. This is motivated by a hope, that the new population will be better than the old one. Solutions which are then used to form new solutions (*offspring*) are selected according to their *fitness* - the more suitable they are the more chances they have to reproduce.

This is repeated until some condition (for example number of populations or improvement of the best solution) is satisfied.

Outline of the Basic Genetic Algorithm can be described with the following steps [47]

1. [Start] Generate random population of *n* chromosomes (suitable solutions for the problem)

2. [Fitness] Evaluate the fitness *f(x)* of each chromosome *x* in the population

3. [New population] Create a new population by repeating following steps until the new population is complete

    1. [Selection] Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected)

    2. [Crossover] With a crossover probability cross over the parents to form new offspring (children). If no crossover was performed, offspring is the exact copy of parents.

    3. [Mutation] With a mutation probability mutate new offspring at each locus (position in chromosome).

    4. [Accepting] Place new offspring in the new population

4. [Replace] Use new generated population for a further run of the algorithm

5. [Test] If the end condition is satisfied, stop, and return the best solution in current population

6. [Loop] Go to step 2.

Since genetic algorithms is not a basic element of this thesis; but rather used for implementing a lower-level controller for demonstration purposes; details of it are not given in this study. However, several works concerning this topic may be found in the artificial intelligence literature.

For a thorough introduction to the topic of genetic algorithms the reader may refer to [47].

CHAPTER 3

PROPOSED HIGH-LEVEL CONTROLLER ARCHITECTURE

In this part of the thesis the proposed intelligent controller architecture for the high-level control of snake-like robot motion is explained. In the following sections, first of all the general snake-like robot structure, considered in the design phase is given. Then the design criteria for the developed controller architecture are stated. Finally, the developed controller architecture satisfying the considered design criteria is explained in detail.

3.1 The General Snake-like Robot Structure

It is worth noting that the proposed high-level controller architecture may be implemented using any type of snake-like robot. Hence, the mechanical model used for demonstrative applications of our approach will be given in the simulations sections of the next chapter. However, in order to be able to describe our controller, we need to use a representative snake-like robot structure that will be introduced here. we have a robot structure as in Figure 3.1.1 consisting of dynamically identical links, with interchangeable extreme links defined as the "head link" and the "tail link", and any link in between is referred to as an "intermediate link". Although the given figure seems to be planar, the robot is able to move in 3D space.

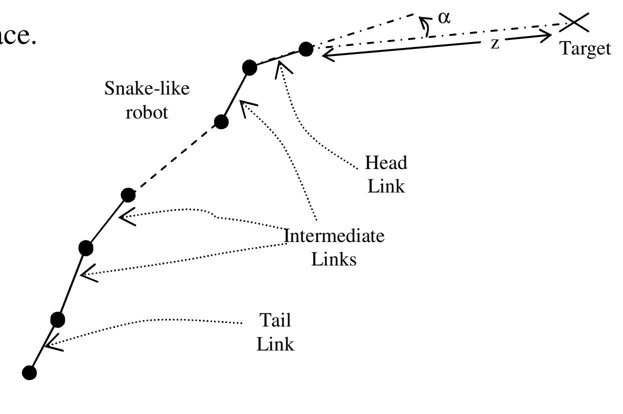


Figure 3.1.1 - The representative snake-like robot mechanism.

The serpentine robot is assumed to be equipped with proximity sensors detecting surrounding obstacles on each side of the links; the head link is further equipped with additional proximity sensors for sensing obstacles in front of the robot. The sensors are grouped in sensor suits (like [29]), each giving the distance of the closest obstacle for the side sensors; distance and angular position of the closest obstacle for the frontal sensors. There are two sensor suits for side sensing and two sensor suits for frontal sensing, as illustrated in Fig. 3.1.2. Sensor suits 1 and 2 are present on all segments, while sensor suits 3 and 4 are only present on the head segment.

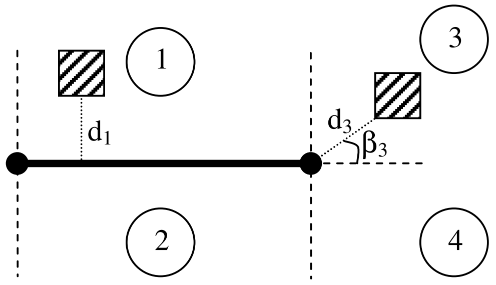


Figure 3.1.2 – Representation of ultra-sound sensor suits of the head segment. Measurements of sensor suits 1 and 3 are shown as distances d_1,d_3, and obstacle sight angle β_3. Obstacles are shown as striped rectangles.

Moreover, the robot is assumed to know the location of the target to be reached implicitly, by measuring α and z variables shown in Figure 3.1.1. This assumption is necessary for error feedback to the controller and was also present in several similar studies like [3] and [19].

3.2 Design Criteria for the High-level Controller

As stated in [28], serpentine SAR devices possess the disadvantage of complexity and uncertainty handling in motion gait planning and control due to high degrees of freedom. In designing a control system for such a complex structure there are several issues to be taken into account, and several problems for which solutions must be developed. In this study, we focus on task adaptability on variable terrain by changing the serpentine gaits for target searching despite obstacles, and also task shapability where snake redundancy is partially distributed to lasso-type grasping and dragging or lifting tasks; while the remaining portion undergoes serpentine gaits. Control complexity and uncertainty in gait planning and execution are overcome by a distributed fuzzy actor critic learning (FACL) controller architecture, described in detail in section 2.3.2.3 above, that we modify for our serpentine robot. We couple our controller architecture to an online optimization technique modified from Dowling's studies [30] for dynamic gait configuration changes. Moreover, in order to fully take the advantage of the hyper-redundant mechanism, we added a lasso-type grasping scheme to the abilities of our robot, making it superior than wheeled and legged robots in SAR applications. Following is a summary of criteria that have to be met in the design of the proposed controller architecture.

i. Dealing with the uncertainties, the non-linearity of the complex hyper-redundant structure, and the dynamic behavior of the environment through learning control.

ii. Being segment-modular, such that every link having the same mechanical structure is controlled individually in the highest control level, making the whole structure a robot network.

iii. Being control-modular, such that different duties of the controller are carried out by different processing units rendering the control architecture, a control network, thus yielding property iv.

iv. Allowing distributed control and parallel processing in the implementation; such that the computational burden of the complex algorithms is divided between the controllers of individual links, overlaying the robot network with the control network.

3.3 The Proposed High-Level Controller Architecture

In order to achieve the aforementioned design criteria, we divide the basic control aims first into the two classical behaviors of "target reaching" and "obstacle avoidance" [29]; then, extending them by the "object grasping" behavior for grasping desired objects during serpentine motion, when necessary. Having three control objectives for the robot, we further distribute these objectives throughout the robot network's individual links as separate controllers. Our approach yields the general controller architecture block diagram as shown in Fig. 3.3.1.

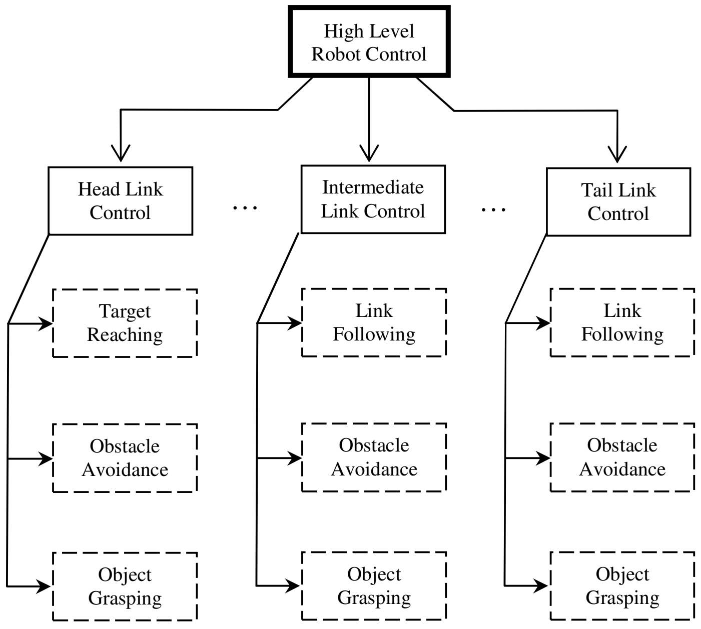


Figure 3.3.1 - The block diagram of the high-level control architecture.

As the result, we have a segment modular and control modular high-level architecture, i.e. a robot network overlaid by a control network, where each segment applies an individual behavior for a given state.

When target-reaching behavior is active, the head link tries to reach the target; as if it is an individual robot, and other links try to follow the immediate preceding link. In the existence of nearby obstacles, a link may switch to obstacle avoidance behavior individually. Moreover, when object grasping is needed, target-reaching behavior is temporarily switched to object grasping behavior; such that target reaching becomes a secondary objective and the robot tries to grasp the desired object without colliding into surrounding obstacles, until grasping is successfully completed, or canceled. The implementation of this high level architecture is realized in our work using Fuzzy Actor Critic Learning (FACL) in the individual controllers, where uncertainties such as vagueness, imprecision, and interval valuedness are modeled as fuzzy learning inference with fuzzy rules.

3.3.1 Controller Structures in the Individual Links

As stated before, the control objectives of our robot is distributed between the links. The head link is the leading part; it decides on its heading velocity and angular speed at a given time, like an individual robot, which completely defines its next state. It only uses the local measurements of target position for target reaching, and sensor suit measurements for the obstacle avoidance behavior. The block diagram of head controller is shown in Figure 3.3.2.

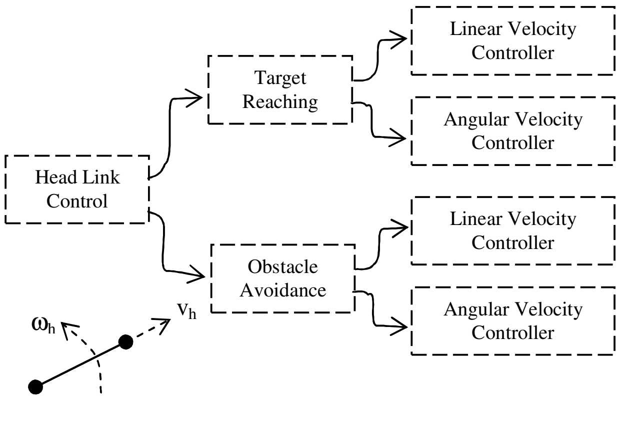

Head Link Control
Target Reaching
Obstacle Avoidance
Linear Velocity Controller
Angular Velocity Controller
Linear Velocity Controller
Angular Velocity Controller
ω_h
v_h

Figure 3.3.2 - The block diagram of the high level head link control architecture for target reaching and obstacle avoidance behaviors.

The other links (intermediate links and the tail link) are all defined as a single group of "follower" links. Controllers of these links are identical and trained at once, which may be further divided into groups for the case of excessive number of links in order to avoid collisions between the links of the robot. A link controller controls the connected joint angular velocity, $\omega_{(j-1)}$, as shown in Figure 3.3.3.

After processing of each controller, the robot decides on its next states, each state decided by individual controllers. (The head link controller determines the spatial positioning of the robot, and the other link controllers determine the internal mechanism structure, i.e. joint angles).

Each controller of the robot is made of individual FACL controllers. For the head link, the target reaching behavior controller uses (α,z) as the state feedback. The obstacle avoidance controller uses the values of d_j for j=1,...,4 and the distance sight angle β_j for j=3,4.

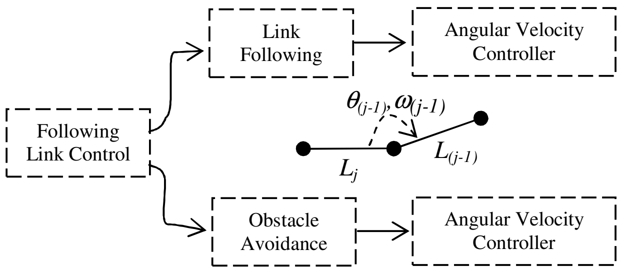

Link Following
Angular Velocity Controller
$\theta_{(j-1)}, \omega_{(j-1)}$
Following Link Control
L_j
$L_{(j-1)}$
Obstacle Avoidance
Angular Velocity Controller

Figure 3.3.3 - The block diagram of the high level following link control architecture for target reaching and obstacle avoidance behaviors.

For the "follower" links, each behavior controls the joint angular speed between two consecutive links. For the "link following" behavior, the inputs to the controller are the current angular speed of the related angle ($\omega_{(j-1)}$,which may be estimated from optical encoder readings on a possible hardware implementation) and the current value of the related angle $(\theta_{(j-1)})$. For the "obstacle avoidance" behavior d_1 and d_2 measurements of the sensor suits and the angular velocity of the related angle, $\omega_{(j-1)}$, are used. The block diagram of these controllers is shown in Fig. 3.3.3.

3.3.2 Incorporating the Object Grasping Behavior

One of the applications of snake-like robots which may be very useful in SAR applications is lasso-type grasping [7]. This ability of the robot may be used to carry parts in ruins of collapsed buildings and to clean the way of the robot. Moreover, a victim may be grasped and carried, or can be freed of debris pressing onto him in future applications. As our major contribution, one of our main goals was to implement a grasping scheme during serpentine locomotion for our snake-like robot dedicated to SAR applications.

While our robot is moving in an environment, it tries to reach a previously defined target point, and avoid obstacles on the way. In a SAR application, a camera for image acquisition is employed and the user can see this image using RF techniques (or by direct cabling), outside of the robot's workspace. When an object to be grasped is seen in the camera, the robot is instructed to grasp the object by the outside rescue team. After getting the grasping command, the robot switches from "target reaching" behavior to "object grasping behavior"; while "obstacle avoidance" behavior is still active. Hence, "object grasping" is an alternative to "target reaching". When "object grasping" becomes active, "target reaching" becomes a secondary objective and is postponed until the object is grasped successfully or the grasping operation is canceled by the user.

Our "object grasping" behavior implementation consists of three stages. The first stage is getting closer to the object to be grasped. In this stage, the head link moves towards the object to be grasped, while the "follower" links still apply the "link following" behavior. This stage is achieved using the "target reaching" behavior with target set to the object. When the head link reaches the object boundary, the second stage is initiated. In this stage, the head link aims at going around the object, beginning to enwrap the object. For the "follower" links, a link sensing the object switches from "link following" behavior to "object grasping" behavior. The aim of the link becomes the same as the head link in the second stage: going around the object and enwrapping it. In the third stage, the head finishes going around the object (turning 360 degrees in plane), while the "follower" links are still going around the object. At this point the object is surrounded by robot links, the robot decides on switching back to "target reaching" phase, if it decides that it can carry the object, otherwise more links go around the object in order to apply higher pulling force (power grasp) to the object. After successful completion of object grasping, the links around the object become "inactive" from the control point of view, hence their angles are forced to be constant, i.e. lasso links stiffen; while the remaining links continue normal serpentine operation in the "target reaching" behavior instead of "object grasping" behavior.

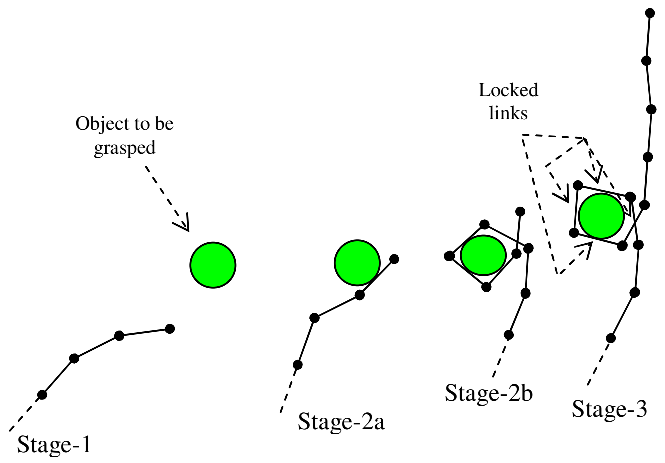


Figure 3.3.4 - Visualization of the planar grasping behavior. Different stages are shown. First the robot is directed towards the object (Stage-1), then head switches to "object grasping" behavior (Stage-2). When head completes enwrapping, the following object sensing links continue enwrapping operation, while head switches to "target reaching" behavior (Stage-2b). Finally, when object becomes to a position that power grasping is possible, related links are locked and other links perform "target reaching" behavior.

Object grasping behavior is also realized with FACL controllers. For the head link, at the first stage the “target-reaching” controller is used as the target is set to the object to be grasped. In the second stage, distance to the object (sensed by obstacle sensors) and the total angular position change of the head (measured from the α value of Figure 3.1.1) are used as the input to the controller. Outputs are still the heading and the angular velocities. Similarly, for the “follower” links, first the “link following” behavior is active, when the object is sensed via obstacle sensors “object grasping” behavior is switched on. In this behavioral stage, the distance to the object, controlled angle angular speed and angle value are used as inputs, while the output is still the angular velocity of the angle between the link and the preceding link. When head completes a single turn around the obstacle, it switches back to “target reaching” behavior with target is set to the original position. A “follower” link which completes the turn also switches back to “link following” behavior. When the robot decides that the power grasp may be succeeded the grasping links are locked, and their link angles are fixed, while all the other links continue in “link following” (“target reaching” for the head link) behavior. A visualization of this approach is given in Fig. 3.3.4.

3.3.3 Dynamic Serpentine Gait Selection

Up to this point of the thesis, dynamics are somehow ignored, there are several studies on snake dynamics and implementation of snake-like motion in robotics such as [2],[31],[17], and [7]. In similar control studies, [3] and [17], previously memorized gaits are used for motion. However, these approaches are not generic and their origin comes from trying to mimic the locomotion of real snakes by link motions. In our study, we use an on-line optimization approach similar to the one in [30]. A matrix of individual entries corresponding to the link angles at different time steps is used. This is an $N \, x \, (t/T_s)$ matrix, where N is the number of links, t is the time length of the movement, T_s is the execution time step. A genetic algorithm implementation is, then, used to evaluate the best matrix, by employing sinusoidal modal functions, resulting in a best-fit solution to the desired position in state-space at each time length of the movement. Our approach concerning the details of the genetic algorithm used during simulationa is given in section 4.3.2. As the result, the active FACL controllers produce the state of the robot for the next time, and the genetic algorithm is used to search the necessary angular movements in time, to achieve the desired position dynamically. With this approach, we incorporate the environmental conditions into gait programming. The drawback is that, the friction coefficients must be known (or estimated) in order to correctly evaluate the resulting motion in space due to changing link angles. Note also that, low-level control of link actuators are not taken into account and assumed to be achieved ideally.

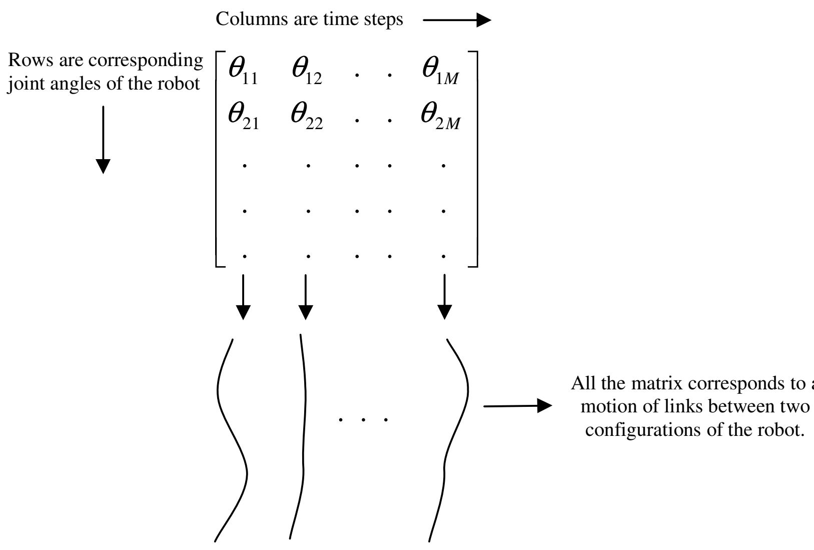

Columns are time steps
Rows are corresponding joint angles of the robot
θ_{11} θ_{12} . . θ_{1M}
θ_{21} θ_{22} . . θ_{2M}
All the matrix corresponds to a motion of links between two configurations of the robot.

Figure 3.3.5 – Representation of the motion generation via genetic algorithms and motion matrix [30]. Although a 2D schematic is shown here, if there are angle entries for 3D motion, then the resultant motion is in space rather than plane.

After incorporating the dynamic motion generation to the system, our basic algorithm may be represented as in Figure 3.3.6.

At this point, it is worth noting that the main focus of this study is the higher level control architecture, which is used for target reaching, obstacle avoidance and object grasping purposes. The lower-level genetic algorithm control implementation is not original to this study and used because it is in coherence with the genericity of the developed high-level control algorithm.

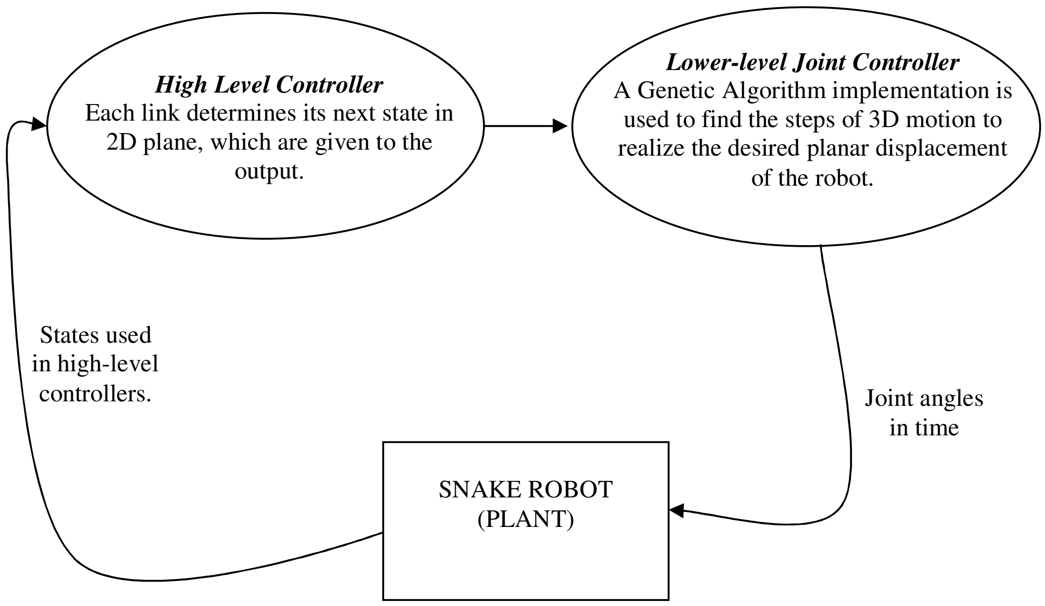

High Level Controller
Each link determines its next state in 2D plane, which are given to the output.
Lower-level Joint Controller
A Genetic Algorithm implementation is used to find the steps of 3D motion to realize the desired planar displacement of the robot.
States used in high-level controllers.
Joint angles in time
SNAKE ROBOT (PLANT)

Figure 3.3.6 – Block diagram showing the feedback architecture of the system.

3.4 Parameters Used for the High Level Controllers

Every high level controller developed in this study is a different form of FACL controller with different parameters. At this part of the report, we give numerical values of these parameters that are used in the simulations.

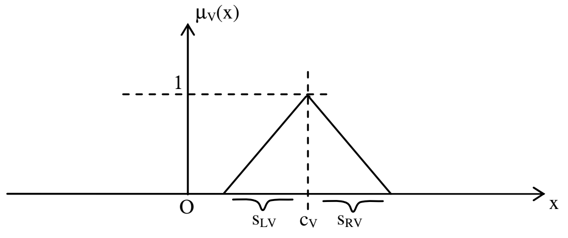


Figure 3.4.1 – Parameters of a typical triangular fuzzy set defined for the inputs.

As given in Chapter 2, trapezoidal membership functions are used for the input fuzzy sets in FACL. We further simplify this assumption by setting triangular fuzzy sets for the inputs. In Figure 3.4.1 typical parameters of a triangular fuzzy set is shown. It is also worth noting that the output fuzzy sets are single valued, i.e. they are crisp. Crisp fuzzy sets may be represented as triangular fuzzy sets with both spreads set to 0, while the center value is the single point at which the fuzzy membership takes the value 1. The parameters of the fuzzy sets used in the controllers are shown in Table 3.4.1

There are also several properties of controllers directly defined by the inputs and outputs. These are number of rules and the number of action-weights used for that controller. Moreover, the final values of reinforcement learning parameters such as eligibility traces' recency factors (λ), reinforcement learning discount factor (γ) and value function initial learning rate (β) are all adjusted by trial and error. All these values are given in Table 3.4.2.

Reward generation mechanism is also specific to a controller. This issue is very critical, since determining the reward generation mechanism determines exactly what should the controller "control" and how to do that. This is similar to a reference set point source for a classical controller. The table showing the reward generation mechanism for each of the controllers used is given as Table 3.4.3.

VARIABLE	FUZZY SET	S_{LV}	C_V	S_{RV}
Object, Joint and Target Angles (α_o,θ,α)	NVS	0	$-\pi$	$\pi/3$
	NS	$\pi/3$	$-2\pi/3$	$\pi/3$
	N	$\pi/3$	$-\pi/3$	$\pi/3$
	Z	$\pi/3$	0	$\pi/3$
	P	$\pi/3$	$\pi/3$	$\pi/3$
	PB	$\pi/3$	$2\pi/3$	$\pi/3$
	PVB	$\pi/3$	π	0
Sensory Distances (d_1, d_2 ,d_3 ,d_4)	S	0	0	0.5
	M	0.5	0.5	0.25
	B	0.25	0.75	∞
Sensory Angles (β_3,β_4)	S	0	0	$\pi/4$
	M	$\pi/4$	$\pi/4$	$\pi/4$
	B	0	$\pi/2$	$\pi/4$
Angular speeds of all links (ω)	NB	0	$-\pi/4$	0
	NS	0	$-\pi/8$	0
	Z	0	0	0
	PS	0	$\pi/8$	0
	PB	0	$\pi/4$	0
Head tip o target point distance (z)	VN	∞	2.5	5
	N	5	7.5	5
	F	5	12.5	5
	VF	5	17.5	∞
Directional speed of head (v)	NB	0	−2	0
	NS	0	−1	0
	Z	0	0	0
	PS	0	1	0
	PB	0	2	0

Table 3.4.1 – Fuzzy sets used for the input and output variables of the controllers.

Controller	***Inputs***	***Outputs***	***Number of Rules***	***Number of Weights***	***R.L. Parameter Vector*** $[\gamma \lambda\, \lambda_a\, \beta_0]$
Head T.R. / Angle	(z, α)	ω	4x7 = 28	28x5 = 140	[0.9 0.1 0.1 0.0001]
Head T.R./Linear Speed	(z, α)	v	4x7 = 28	28x5 = 140	[0.9 0.5 0.1 0.0001]
Head O.A.	$(d_1, d_2, d_3, d_4, \beta_3, \beta_4)$	(ω,v)	3x3x3x3x3x3 = 729	729x5x5 = 18225	[0.9 0.5 0.5 0.0001]
Head O.G.	$(d_1, d_2, d_3, d_4, \beta_3, \beta_4, \alpha_o)$	(ω,v)	3x3x3x3x3x3x7 = 5103	5103x5x5 = 127525	[0.9 0.9 0.9 0.0001]
Follower Link / T.R.	(θ,ω)	ω	7x7 = 49	49x5 = 245	[0.9 0.9 0.5 0.0001]
Follower Link / O.A.	(d_1, d_2, ω)	ω	3x3x5 = 45	45x5 = 225	[0.9 0.5 0.5 0.0001]
Follower Link / O.G.	$(d_1, d_2, \omega, \alpha_o)$	ω	3x3x5x7 = 315	315x5 = 1575	[0.9 0.9 0.9 0.0001]

Table 3.4.2 – High level controller parameters. T.R.: Target Reaching; O.A.: Obstacle Avoidance; O.G.: Object Grasping.

The final point that should be noted regarding the high level controllers is the combination of actions of different controllers which are active at the same time instant. This is achieved by a linear combination of different actions offered by different controllers. The weights used in these linear combinations are functions of the situation. Such situations occur when target reaching and obstacle avoidance behaviors are active for the same link at the same time instant. The resultant output of the high level controller is given as

$$Y_{OUT} = k_{TR} \times Y_{TR} + k_{OA} \times Y_{OA}$$

Where $k_{OA} = 1 - \min(d_{i(t)})$, and $k_{TR} = (1 - k_{OA})$

There are also parameters κ, σ and φ used in FACL structure. These values are set as follows:

$$\kappa = 0.01 \cdot \beta_o$$
$$\sigma = 0.01$$
$$\varphi = 0.2$$

Controller	*Parameters*	*Parameter Values*	*Success Condition*	*Failure Condition*	*Reward Function*
Head T.R. / Angle	-	-	$\lvert\alpha\rvert<\pi/18$	$\lvert\alpha\rvert>3\pi/4$	$r_t = -1$ if failure $r_t = 1$ if success $r_t = 0$ else
Head T.R./Linear Speed	k: reward gain $\Delta z = z_t - z_{(t-1)}$	$k = -0.05$	$z < 0.5$	$z > 10$	$r_t = -1$ if failure $r_t = 1$ if success $r_t = k \times (\Delta z)$,else
Head O.A.	k = obstacle avoidance gain $\Delta d_i = \min(d_{i(t+1)}) - \min(d_{i(t)})$, i=1..4	$k = 1-\min(d_{i(t)})$	-	Obstacle hit	$r_t = -10$ if failure $r_t = k \times (\Delta d_i)$,else
Head O.G.	k_α = object angle gain k_d = object distance gain $\Delta d_i = \min(d_{i(t+1)}) - \min(d_{i(t)})$, i=1..4 $\Delta\alpha_o=\alpha_{o(t+1)}-\alpha_{o(t)}$	$k_\alpha = 0.05$ $k_d = 0.05$	$\alpha_o>35\pi/36$	$\alpha_o< -\pi/36$ OR Object Hit OR $d_o>0.3$	$r_t = -10$ if failure $r_t = 10$ if success $r_t = k_d \times \Delta d_i + k_\alpha \times \Delta\alpha_o$ else
Follower Link / T.R.	k = joint angle gain $\Delta\theta_j = \lvert\theta_{j(t+1)}\rvert - \lvert\theta_{j(t)}\rvert$	$k = 0.05$	Target reached with $\lvert\theta\rvert<2\pi/18$	$\lvert\theta\rvert>5\pi/18$	$r_t = -10$ if failure $r_t = 10$ if success $r_t = k \times \Delta\theta_j$ else
Follower Link / O.A.	k = obstacle avoidance gain $\Delta d_i = \min(d_{i(t+1)}) - \min(d_{i(t)})$, i=1,2	$k = 1-\min(d_{i(t)})$	-	Obstacle hit	$r_t = -10$ if failure $r_t = k \times (\Delta d_i)$,else
Follower Link / O.G.	k_α = object angle gain k_d = object distance gain $\Delta d_i = \min(d_{i(t+1)}) - \min(d_{i(t)})$, i=1..4 $\Delta\alpha_o=\alpha_{o(t+1)}-\alpha_{o(t)}$	$k_\alpha = 0.05$ $k_d = 0.05$	$\alpha_o>35\pi/36$	$\alpha_o< -\pi/36$ OR Object Hit OR $d_o>0.3$	$r_t = -10$ if failure $r_t = 10$ if success $r_t = k_d \times \Delta d_i + k_\alpha \times \Delta\alpha_o$ else

Table 3.4.3 – High level controller parameters used for reward calculations. T.R.: Target Reaching; O.A.: Obstacle Avoidance; O.G.: Object Grasping.

It should also be noted that a sampling time of 0.1 seconds is used for the high level simulations, and 0.002 seconds sampling time is used for dynamical simulations within a sample period.

CHAPTER 4

SIMULATIONS AND RESULTS

4.1 Simulation Environment

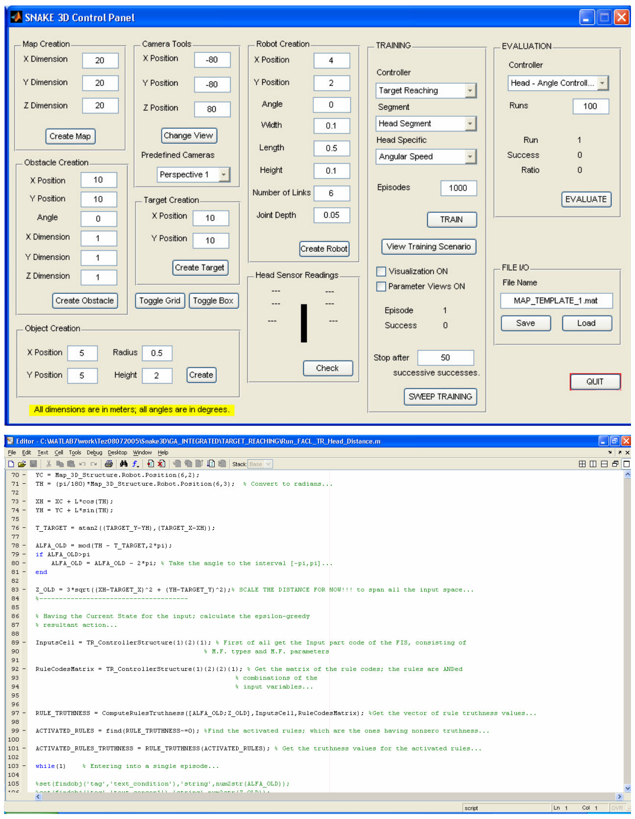
For the verification and the evaluation of the developed high-level controller architecture, a simulation program is developed in the MATLAB environment. The program consists of different modules. These modules are the graphical user interface (GUI), the controllers' source codes (written as M-files), genetic algorithm module (implemented using Genetic Algorithms Toolbox of MATLAB), mechanical simulation module (implemented using Simmechanics blockset of MATLAB-Simulink) and the visualization module. Snapshots from these modules are shown in Figures 4.1.1, 4.1.2 and 4.1.3.

For easy access to the written source code, the GUI shown in Figure 4.1.1 is prepared. Within this GUI, the user can perform the following operations:

- Create map with desired x,y and z dimensions,
- Create obstacles at any desired position in the map, with adjustable dimensions and angular orientation in z axis,
- Create target point at any desired position in the map,
- Create cylindrical object at any point in the map by specifying its radius and height,
- Create snake robot by specifying segment dimensions, joint dimensions, number of links, initial position, and initial alignment,
- Change the camera parameters for viewing from different positions,
- Train any FACL controller during specified number of episodes.

- Evaluate the performance of any FACL controller with incorporation of dynamics as an option,
- Perform sweep training, which runs training of selected controller for different values of FACL parameters,
- Save and load previously created maps,

Upon selection of each operation, the callback functions of the used button controls performs batch call to related M-file source code and Simulink simulations when necessary.



Map Creation: X Dimension 20; Y Dimension 20; Z Dimension 20

Camera Tools: X Position -80; Y Position -80; Z Position 80; Predefined Cameras: Perspective 1

Robot Creation: X Position 4; Y Position 2; Angle 0; Width 0.1; Length 0.5; Height 0.1; Number of Links 6; Joint Depth 0.05

TRAINING: Controller: Target Reaching; Segment: Head Segment; Head Specific: Angular Speed; Episodes 1000

EVALUATION: Controller: Head - Angle Controll...; Runs 100; Run 1; Success 0; Ratio 0

Obstacle Creation: X Position 10; Y Position 10; Angle 0; X Dimension 1; Y Dimension 1; Z Dimension 1

Target Creation: X Position 10; Y Position 10

Head Sensor Readings: --- --- --- --- --- ---

Visualization ON; Parameter Views ON; Episode 1; Success 0

FILE I/O: File Name MAP_TEMPLATE_1.mat

Object Creation: X Position 5; Y Position 5; Radius 0.5; Height 2

Stop after 50 successive successes.

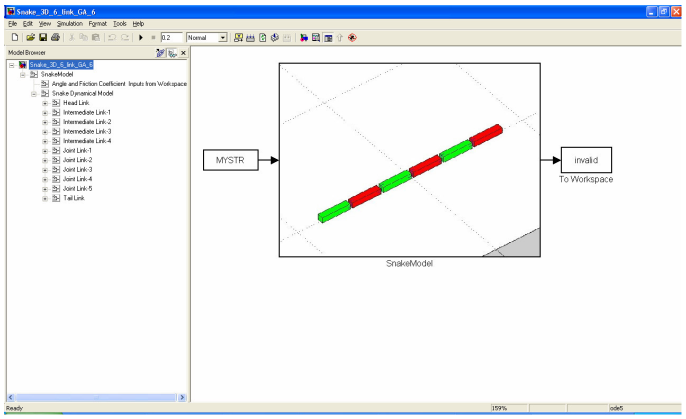
All dimensions are in meters; all angles are in degrees.

Editor - C:\MATLAB7\work\Tez08072005\Snake3D\GA_INTEGRATED\TARGET_REACHING\Run_FACL_TR_Head_Distance.m

```
YC = Map_3D_Structure.Robot.Position(6,2);
TH = (pi/180)*Map_3D_Structure.Robot.Position(6,3);  % Convert to radians...

XH = XC + L*cos(TH);
YH = YC + L*sin(TH);

T_TARGET = atan2((TARGET_Y-YH),(TARGET_X-XH));

ALFA_OLD = mod(TH - T_TARGET,2*pi);
if ALFA_OLD>pi
    ALFA_OLD = ALFA_OLD - 2*pi; % Take the angle to the interval [-pi,pi]...
end

Z_OLD = 3*sqrt((XH-TARGET_X)^2 + (YH-TARGET_Y)^2);% SCALE THE DISTANCE FOR NOW!!! to span all the input space...
%---------------------------------------

% Having the Current State for the input; calculate the epsilon-greedy
% resultant action...

InputsCell = TR_ControllerStructure{1}{2}{1}; % First of all get the Input part code of the FIS, consisting of
                                            % M.F. types and M.F. parameters

RuleCodesMatrix = TR_ControllerStructure{1}{2}{2}{1}; % Get the matrix of the rule codes; the rules are ANDed
                                                    % combinations of the
                                                    % input variables...


RULE_TRUTHNESS = ComputeRulesTruthness([ALFA_OLD;Z_OLD],InputsCell,RuleCodesMatrix); %Get the vector of rule truthness values...

ACTIVATED_RULES = find(RULE_TRUTHNESS~=0); %Find the activated rules; which are the ones having nonzero truthness...

ACTIVATED_RULES_TRUTHNESS = RULE_TRUTHNESS(ACTIVATED_RULES); % Get the truthness values for the activated rules...

while(1)    % Entering into a single episode...

%set(findobj('tag','text_condition'),'string',num2str(ALFA_OLD));
```

script Ln 1 Col 1 OVR

Figure 4.1.1 - A MATLAB GUI and several M-files codes have been prepared for the simulations.



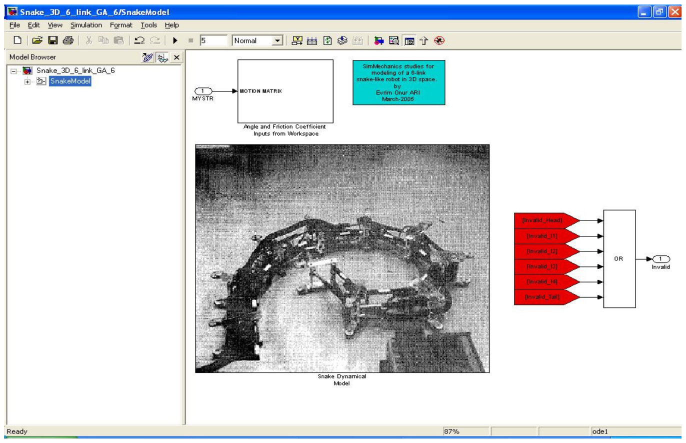

Snake_3D_6_link_GA_6/SnakeModel
MYSTR
MOTION MATRIX
Angle and Friction Coefficient Inputs from Workspace
SimMechanics studies for modeling of a 6-link snake-like robot in 3D space. by Evrim Onur ARI March-2006
Snake Dynamical Model
OR
Invalid

Figure 4.1.2 - Simulink and its SimMechanics Blockset are employed for performing the dynamical simulations of the robot and environment interaction.

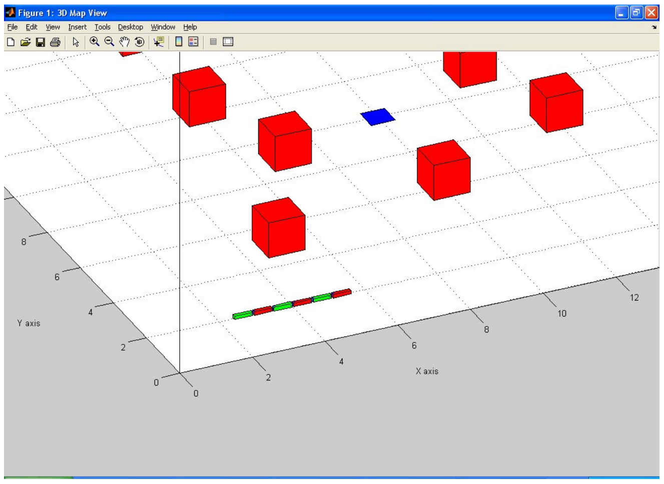


Figure 4.1.3 - A snapshot from the visualization componenent of the simulation program.

Simulations performed using these modules consist of two different phases. The first phase is training of the high-level controllers and the second phase is the performance evaluation with genetic algorithm and the dynamical model added to the controllers. Explanation of these phases is given in the following two sub-chapters.

4.2 High-level Controller Training Simulations

Since the controller architecture developed in this thesis is a modular one, the learning processes of different controllers are achieved in different simulation scenarios. The steps for the learning of the high-level controllers may be given in stages as follows:

1- Training the head's angular speed controller: At this stage, only the head link is considered, and it learns aligning through the target to be reached in an environment without any obstacles.

2- Train the head's heading speed controller: At this stage, the head link "target reaching" controller is further trained for reaching the target by controlling its heading speed. Since the controller has already learnt aligning towards the target at the first stage; after the second stage, head is able to reach the target point in an environment without any obstacles.

3- Train the following segments' link following controller: At this stage, the following links are trained to follow each other. The link following the head learns to align with the head; the link after it learns to align with this link and so on.

4- Train the head's obstacle avoidance controller: After the steps 1 and 2 we have a head link that can reach a given target by adjusting its angular and heading speeds. At this stage, the head link's "obstacle avoidance controller" is trained to avoid the obstacles, while the target reaching controller trained in steps 1 and 2 is employed for moving through the target point.

5- Train the following segments' obstacle avoidance controller: The following segments are able to follow each other after the training at step 3. At this fifth step, the "obstacle avoidance controller" of the following links is trained for avoiding the obstacles, while the link following controller trained in step 3 is used directly.

6- Train the head's object grasping controller: After step 5 we have a robot which is able to avoid obstacles while reaching a previously given target point in an environment with obstacles. Since we also desire to have an object grasping behavior associated with our robot, we continue with the training of this controller. At this stage, the head segment is learnt to enwrap around a given object that is to be grasped by the snake-like robot.

7- Train the following segments' object grasping controllers: Following segments learn to follow the head segment in enwrapping the object; the segments must go around the object without colliding it, but also without going away from the object so as to enwrap it and apply a pulling force when needed.

It is worth noting that during all these stages of the simulations dynamics are not taken into account. Genetic algorithm tool is used to incorporate the dynamic effects into the simulation in the performance evaluation stage.

The detailed explanation of the simulation of these stages is given in the following paragraphs.

4.2.1 Training of the Head Target Reaching Controller

As stated above, the learning process starts with the target reaching behavior of the head link. This process is achieved in two successive steps. First of these steps is the training of the head for aligning through the target point in the environment. An FACL controller is used for achieving this desired behavior. The inputs to the controller are the distance between the tip point of the head segment and the target (z in Figure 3.1.1), and the heading angle error of the head segment; which is given as the angle α in Figure 3.1.1. Upon getting the current values of these inputs, the head angle controller outputs an angular speed for the head segment. Using the sampling time of the simulation and the orientation of the head link at the previous time step, the new orientation of the head link is found and the segment is redrawn for the calculation of the new desired angular speed.

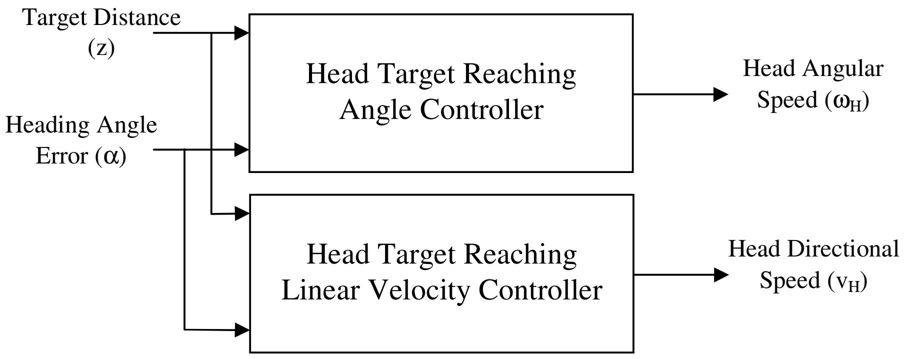

Target Distance (z)
Heading Angle Error (α)
Head Target Reaching Angle Controller
Head Target Reaching Linear Velocity Controller
Head Angular Speed (ω_H)
Head Directional Speed (v_H)

Figure 4.2.1 - Inputs and outputs of the head angle and linear velocity controllers used for the target reaching behavior.

The flowchart of the FACL controller learning algorithm for the head target reaching behavior angle controller is shown in Figure 4.2.3. Actually this flowchart may be generalized to all FACL controllers by replacing the input/output variables with the general terms.

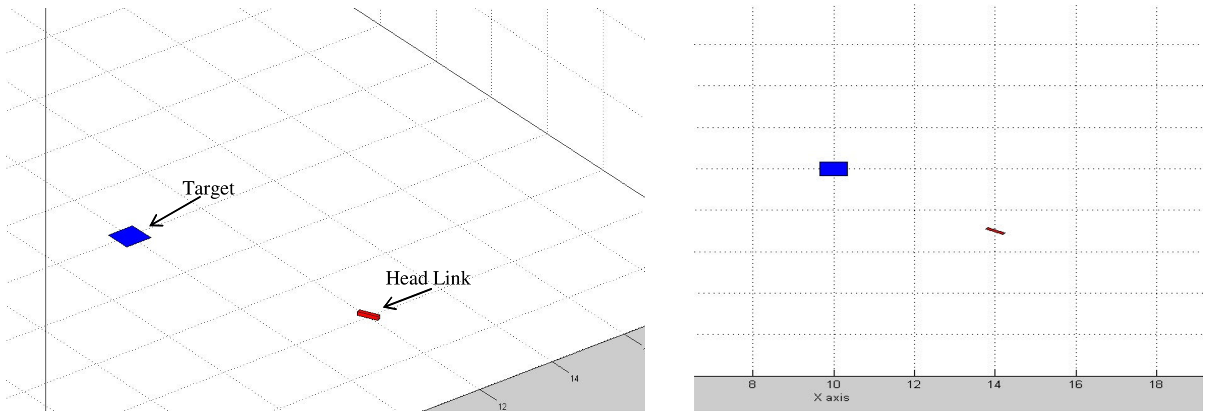


Figure 4.2.2 – Two views from the angular speed learning simulations of the head link. On the left a 3D scene is shown with the perspective view, while on the right a planar scene is shown.

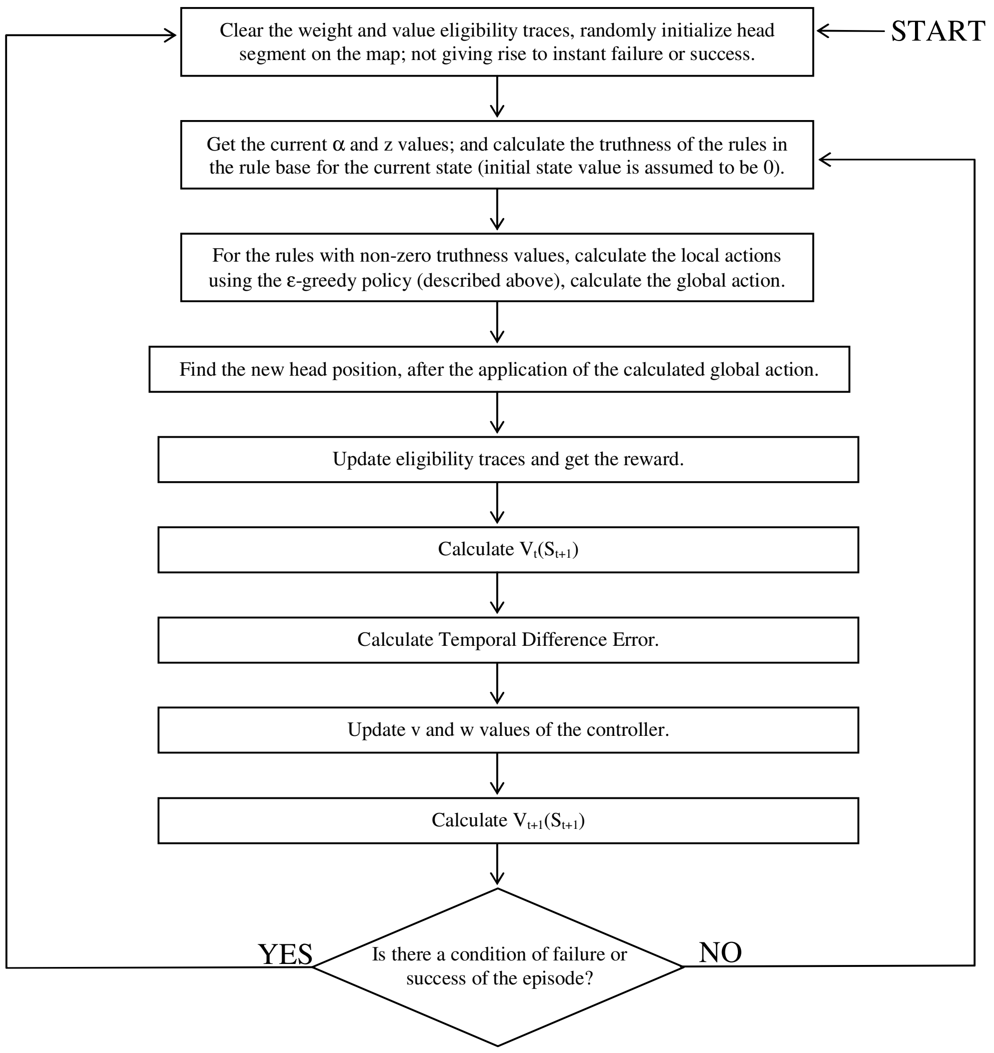

Clear the weight and value eligibility traces, randomly initialize head segment on the map; not giving rise to instant failure or success.
START
Get the current α and z values; and calculate the truthness of the rules in the rule base for the current state (initial state value is assumed to be 0).
For the rules with non-zero truthness values, calculate the local actions using the ε-greedy policy (described above), calculate the global action.
Find the new head position, after the application of the calculated global action.
Update eligibility traces and get the reward.
Calculate $V_t(S_{t+1})$
Calculate Temporal Difference Error.
Update v and w values of the controller.
Calculate $V_{t+1}(S_{t+1})$
Is there a condition of failure or success of the episode?
YES
NO

Figure 4.2.3 – The flowchart showing the implementation steps of the FACL learning for the head link angular speed controller for a single episode.

The second step in head target reaching behavior learning is the training of the head link for adjusting its linear speed in an environment with no obstacles. For example, when the head link points 180 degrees from the target, its directional speed should be small. Hence, we give the distance to the target and the heading angle error as inputs to the controller, and make the directional speed as the output whose pattern should be learnt for different values of the inputs. After training of the head for learning its directional speed behavior, this segment is able to reach a target point in an environment with no obstacles. This is illustrated in Figure 4.2.4.

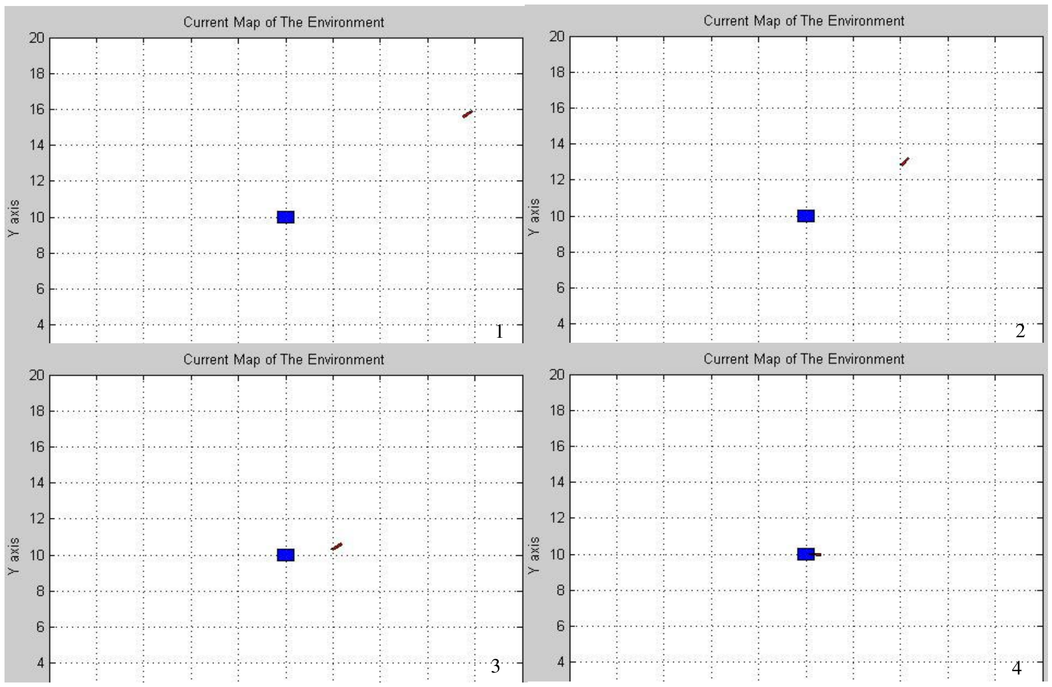


Figure 4.2.4 – Snapshots from the head target reaching behavior learning simulation. Here, an episode is finished with success by reaching to the target at the end of the episode.

4.2.2 Training of the Link Following Controller

After making the head link to reach the target, the next step is making the following links to learn how to follow the link just in front of them. This is achieved by first making the head following link to learn how to follow the head, then replicating this trained controller for the other follower links.

For the link following controller, the current value of the controlled joint angle and the angular speed of the previous link are the inputs. The controller learns to adjust the angular speed of the previous joint such as to minimize the previous joint angle. Hence, a positive reward is given as this angle approaches to 0, and a negative reward is given as this angle deflects from zero. The episode ends with a failure (big negative reward), if angular deflection exceeds a predefined threshold.

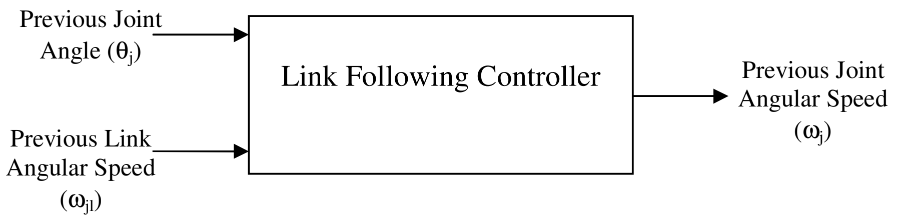

Previous Joint Angle (θ_j)
Previous Link Angular Speed (ω_{jl})
Link Following Controller
Previous Joint Angular Speed (ω_j)

Figure 4.2.5 - Inputs and outputs of the link following controller used for the target reaching behavior.

In the simulations of the training of the link following controller, previously trained head link target reaching controller is operated and the link behind the head is used as the training environment for the controller. After the controller is trained successfully, it is replicated for the other following links. Snapshots from the training stage are given in Figure 4.2.6 and the result of replication of the controller to all follower links is shown in Figure 4.2.7.

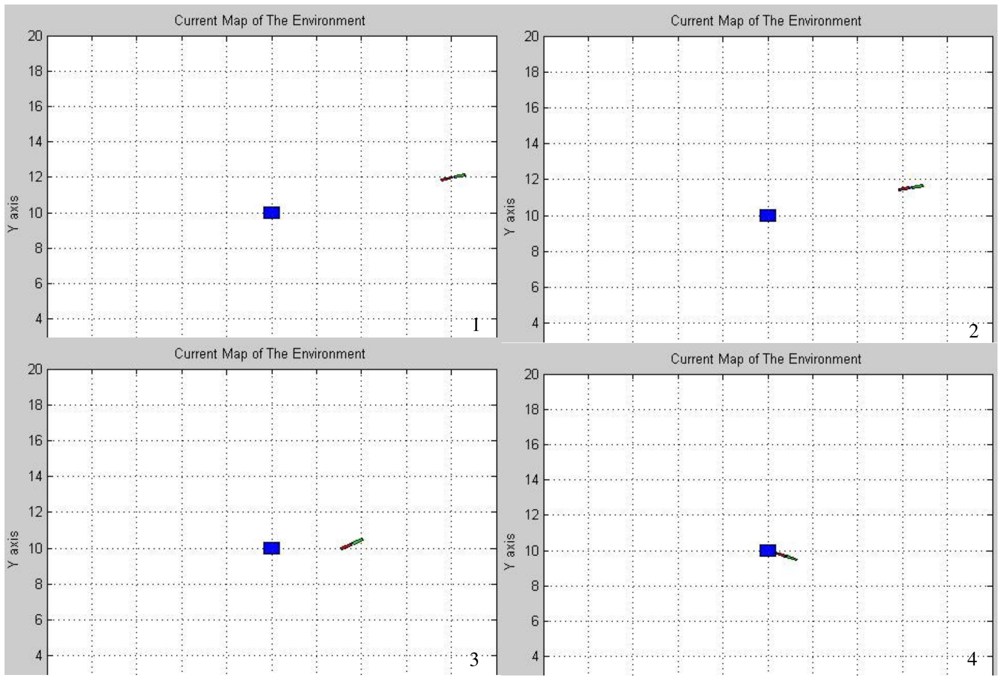


Figure 4.2.6 – Snapshots from a successful episode of link following controller training.

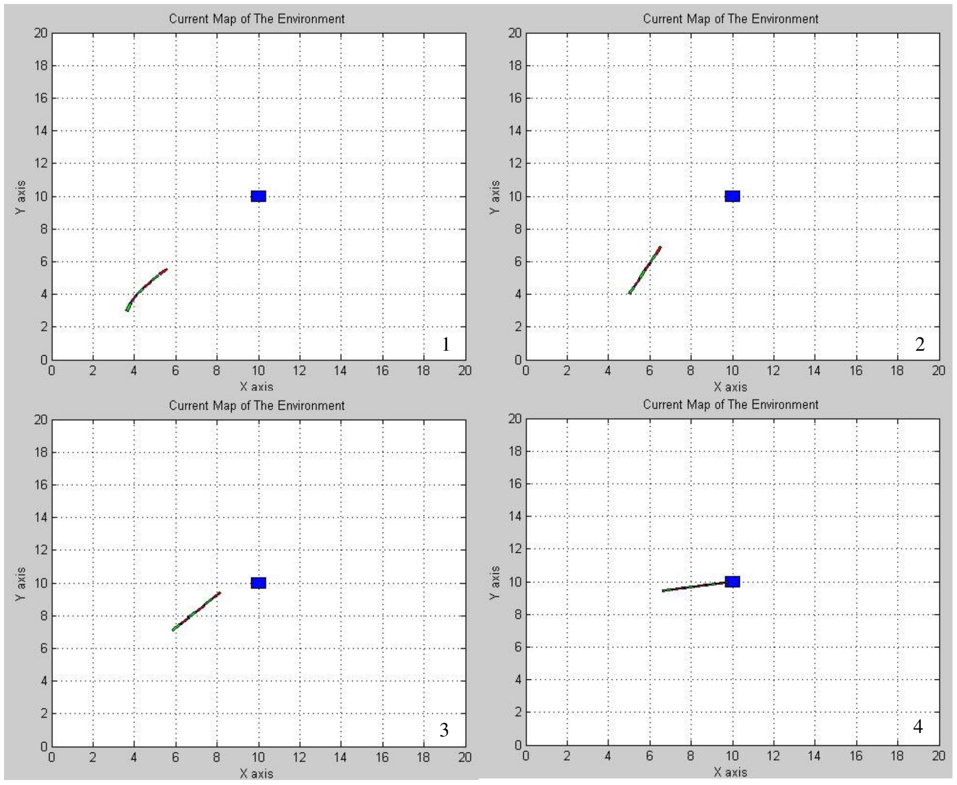


Figure 4.2.7 – Snapshots from a run of trained head target reaching controller and replicated link following controllers.

4.2.3 Training of the Head Obstacle Avoidance Controller

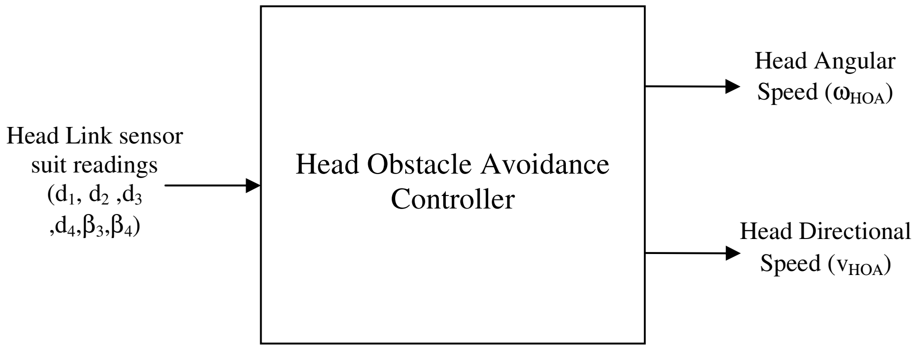

Head Link sensor suit readings (d_1, d_2 ,d_3 ,d_4,β_3,β_4)
Head Obstacle Avoidance Controller
Head Angular Speed (ω_{HOA})
Head Directional Speed (v_{HOA})

Figure 4.2.8 - Inputs and outputs of the head link obstacle avoidance behavior controller.

After training of the controllers related with the target reaching behavior, obstacle avoidance behavior training is performed. At the first step of this training, the head link is taught to avoid obstacles. During the simulations of this stage, the head link is operated in an environment with obstacles and a target point. While the previously trained target reaching behavior is run for this link, obstacle avoidance behavior is activated whenever an obstacle is come across. Outputs of both the target reaching and the obstacle avoidance sensors are the linear and angular velocities of the head link, hence a linear combination of these outputs is used to determine the resultant outputs. The combination weights of different controller outputs are determined by the obstacle closure measure, which is defined as the distance to the nearest obstacle. In Figure 4.2.9 an episode of obstacle avoidance behavior training ending with a failure is shown. After the completion of this training stage, the head link is able to reach a target point without hitting any obstacles in the environment.

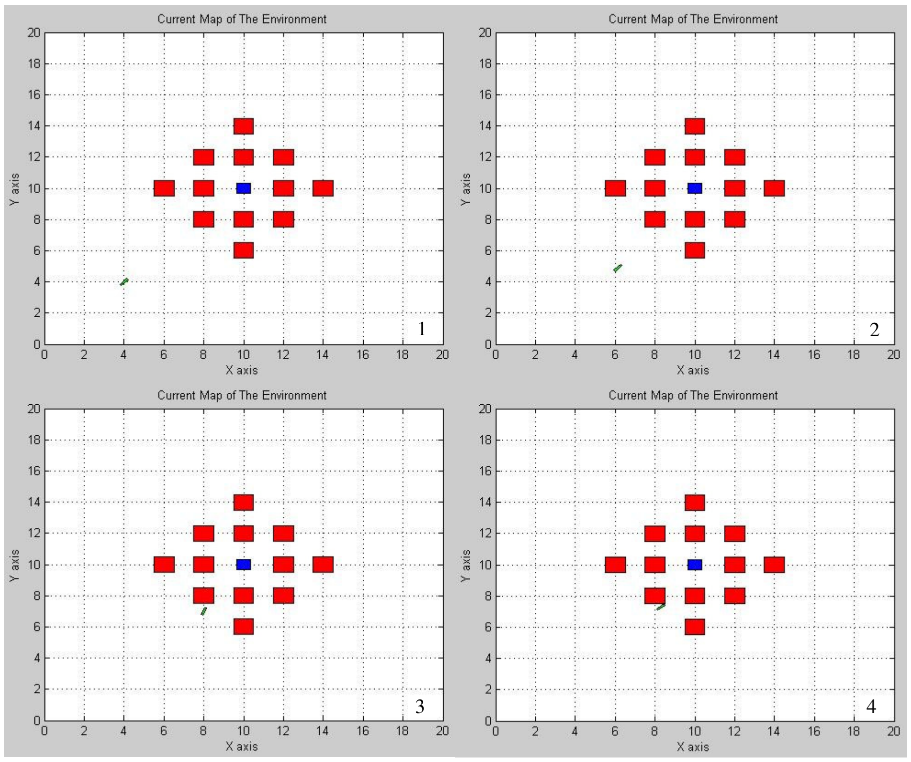


Figure 4.2.9 – Snapshots from an unsuccessful episode of head link obstacle avoidance controller training simulation.

4.2.4 Training of the Follower Link Obstacle Avoidance Controller

The second stage in making the robot learn to avoid obstacles while reaching to the target is training of the obstacle avoidance controllers for the follower link. This is achieved by training the controller for the link following the head, and replicating this controller for the other follower links.

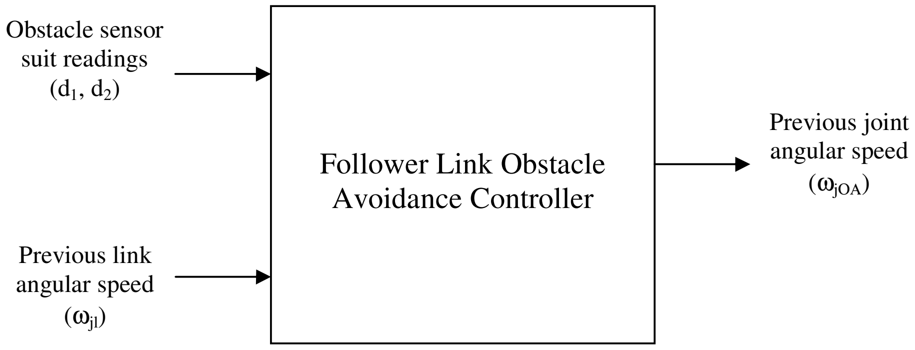

Obstacle sensor suit readings (d_1, d_2)
Previous link angular speed (ω_{jl})
Follower Link Obstacle Avoidance Controller
Previous joint angular speed (ω_{jOA})

Figure 4.2.10 - Inputs and outputs of the follower link obstacle avoidance behavior controller.

In training of this controller, the head link is operated in both target reaching and obstacle avoidance behaviors, while the follower link's link following behavior is also active. Similar to the head link, for the follower link the resultant joint angular speed is determined by a linear combination of the link following and obstacle avoidance behaviors, and the combination weight of each behavior is determined by the obstacle closure measure. A successful episode of the training phase is shown in Figure 4.2.11. After the completion of this stage, replicating the trained controller for the remaining follower links yields a control structure which is able to make the robot reach a target without colliding any obstacles. A sample simulation of the controller with the follower link obstacle avoidance behavior replicated is shown in Figure 4.2.13.

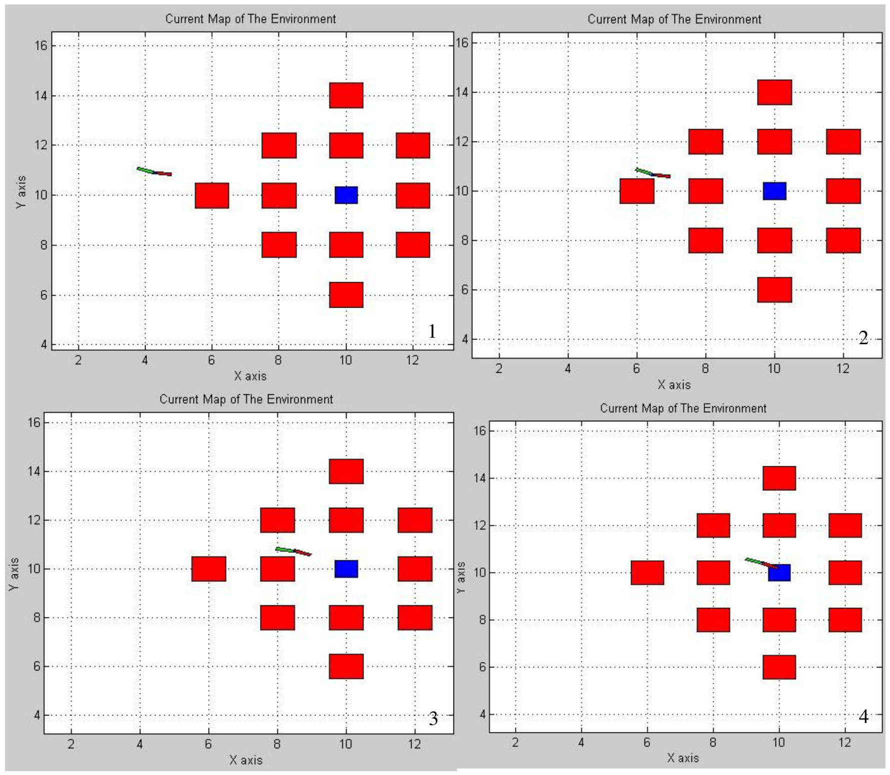


Figure 4.2.11 – Snapshots from a successful episode of follower link obstacle avoidance controller training simulation.

4.2.5 Training of the Head Object Grasping Controller

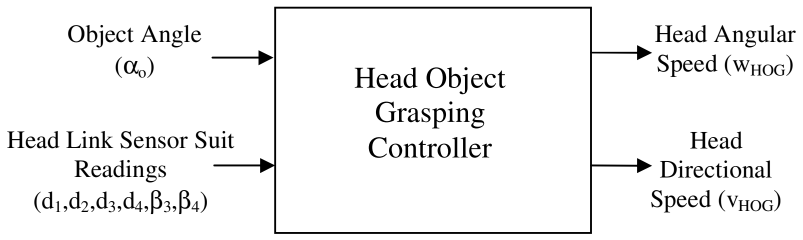


Figure 4.2.12 - Inputs and outputs of the head link object grasping behavior controller.

As we stated before, as an indication of extendibility of our control architecture, we added a grasping scheme to our serpentine robot, and in simulations, we used grasping of cylindrical objects as a case study. After completion of all other training stages, the robot should be trained for grasping objects. As a first step the head link is trained for grasping an object. This is achieved in an environment without obstacles, with a target and an object. Each episode of training starts when the head reaches to the object and switches to the object grasping behavior. An episode ends with failure if head can't enwrap the object in a given amount of time steps, or if it collides with the object. Sample snapshots from the simulations of this training are shown in Figure 4.2.14.

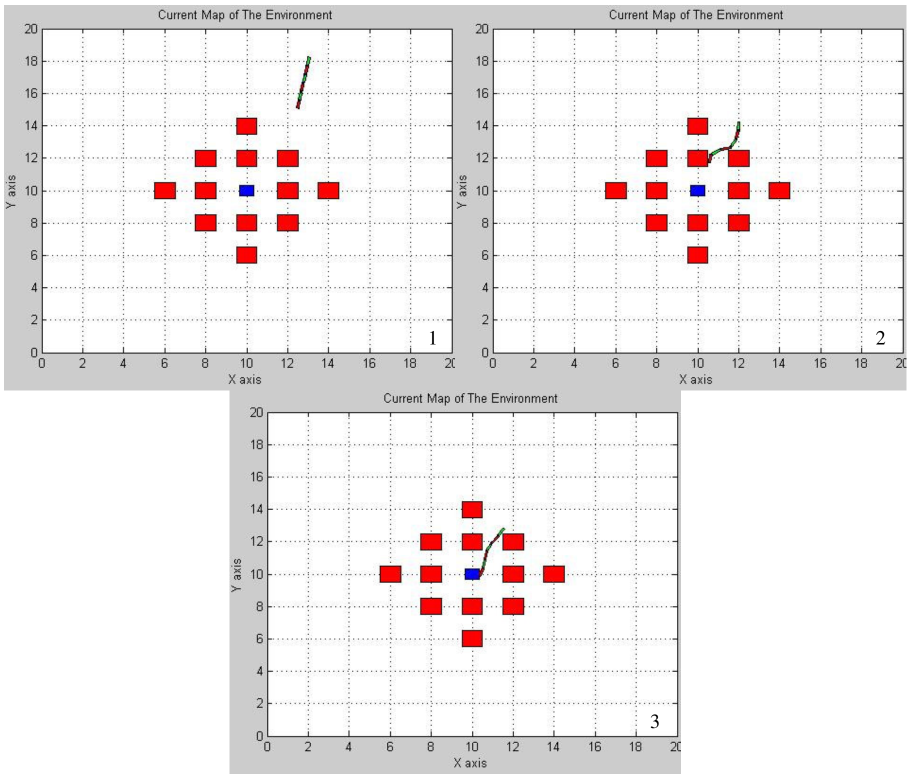


Figure 4.2.13 – Snapshots from a run of target reaching and obstacle avoidance behaviors operated simultaneously.

After completion of head object grasping controller training, next step is training of the following link object grasping controllers.

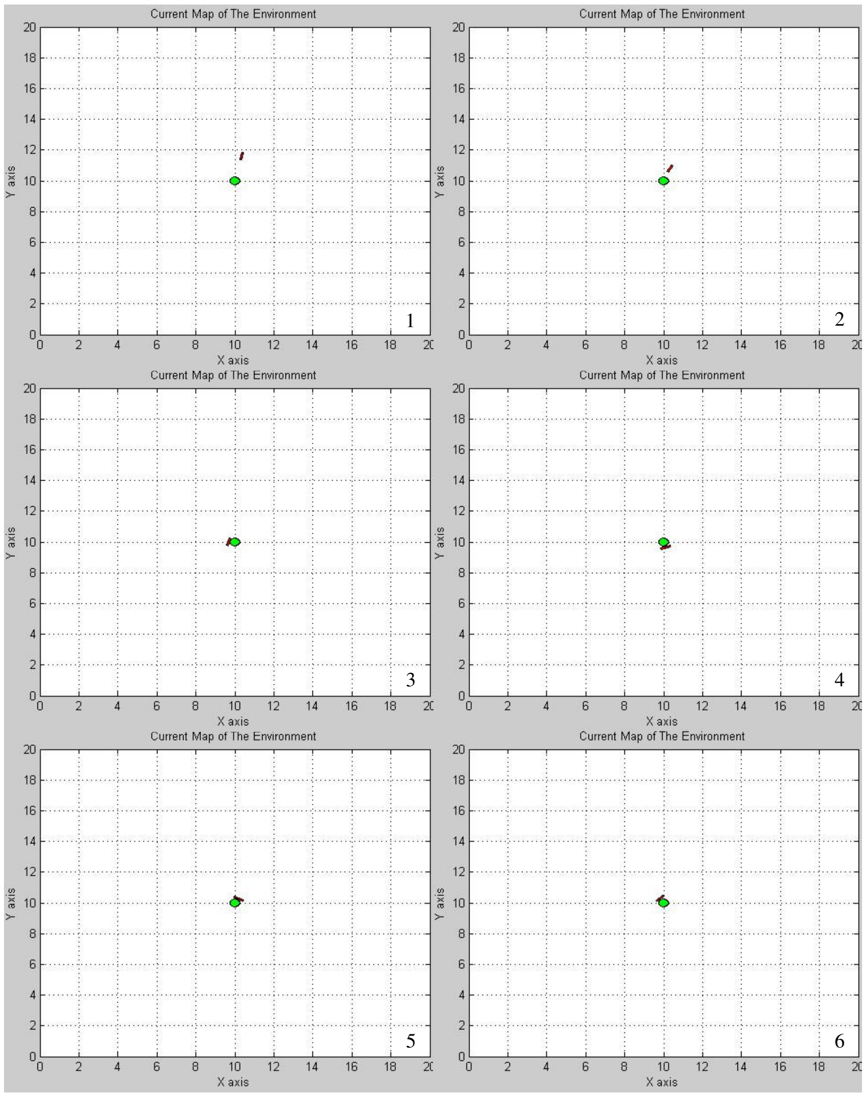


Figure 4.2.14 – Snapshots from a successful episode of head link object grasping controller training.

4.2.6 Training of the Follower Link Object Grasping Controller

Similar to the head object grasping controller, these controllers are also trained in an environment without obstacles. In the training phase, only the link following the head is used, and the trained controller is replicated in the other links.

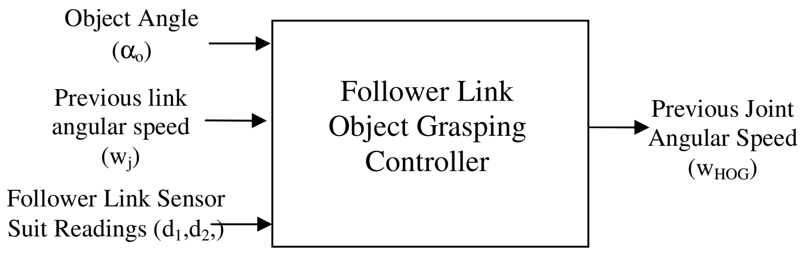

Object Angle (α_o)
Previous link angular speed (w_j)
Follower Link Sensor Suit Readings (d_1,d_2,)
Follower Link Object Grasping Controller
Previous Joint Angular Speed (w_{HOG})

Figure 4.2.15 - Inputs and outputs of the follower link object grasping controller.

Again, until reaching to the object to be grasped, the follower link's link following controller is active. When the object to be grasped is sensed by the obstacle sensors, the following link switches to training of the object grasping behavior (i.e., an episode starts at this time step). The episode ends with a success if the object angle sweeps all the range from the starting point without colliding with the object while the distance of the follower link to the object doesn't exceed an acceptable limit. A run from a successful episode is shown in Figure 4.2.17.

After the completion of this training stage, the robot is able to move in an environment with obstacles, and reach a target without colliding any obstacles. Moreover, it is able to grasp a desired cylindrical object. In other words, training of the high level controller is completed. The discussion on the performance of the learning phase is given in the following paragraphs. The parameters of the high-level controllers used are tabulated in section 3.4 above.

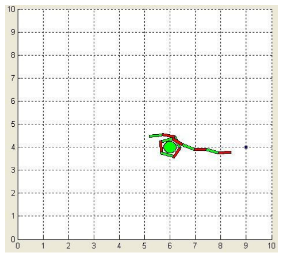


Figure 4.2.16 – After replication of the grasping controller to the other following links, the robot is able to grasp an object.

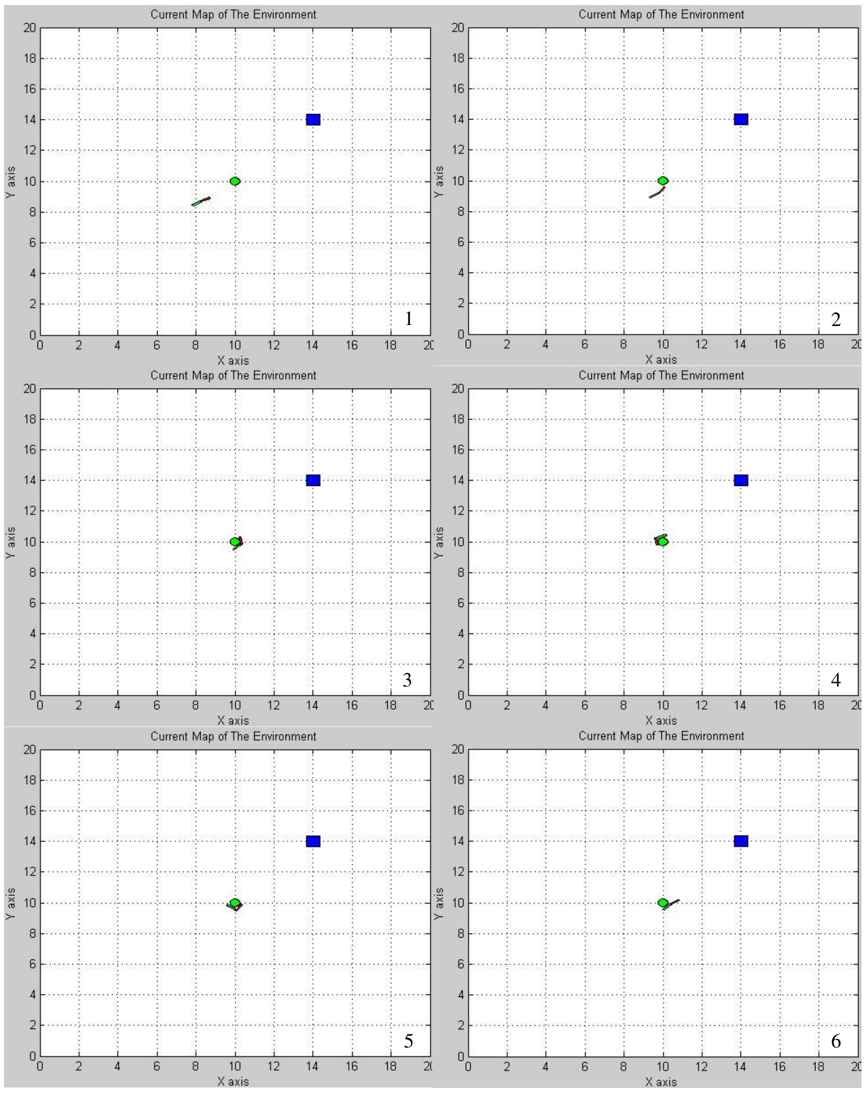


Figure 4.2.17 – Snapshots from a successful episode of follower link object grasping controller training.

4.2.7 Training Simulation Results

In order to evaluate the learning performance of the controllers with different FACL parameter (*[γ λ λ_a β_0]*) values, we follow a way similar to one in [23]. We first define the *learning speed* parameter as the number of training episodes needed before a 50 successive episodes ending in success. If number of training episodes needed exceeds 10000, then we claim that the FACL parameters used are not applicable for correct learning. Note that, since the learning speed is measured using the number of episodes, ast he number of episodes needed increases the learning gets slower. Hence, a big number of learning speed means a slower learning. The obtained values of learning speeds are tabulated in Table 4.2.1 to Table 4.2.7 for different FACL controllers used.

FACL Parameters vector *$[\gamma \lambda \lambda_a \beta_0]$*	*Number of Episodes Needed for Learning*
[0.1 0.1 0.1 0.0001]	N.A.
[0.5 0.1 0.5 0.0005]	N.A.
[0.9 0.1 0.9 0.001]	1562
[0.1 0.5 0.1 0.0001]	N.A.
[0.5 0.5 0.5 0.005]	2628
[0.9 0.5 0.9 0.001]	50
[0.1 0.9 0.1 0.0001]	N.A.
[0.5 0.9 0.5 0.0005]	321
[0.9 0.9 0.9 0.001]	123
[0.9 0.9 0.9 0.0001]	50

Table 4.2.1 – Learning speed for different FACL parameters, given for the head target reaching angle controller.

FACL Parameters vector *$[\gamma \lambda \lambda_a \beta_0]$*	*Number of Episodes Needed for Learning*
[0.1 0.1 0.1 0.0001]	N.A.
[0.5 0.1 0.5 0.0005]	N.A.
[0.9 0.1 0.9 0.001]	1382
[0.1 0.5 0.1 0.0001]	N.A.
[0.5 0.5 0.5 0.005]	3181
[0.9 0.5 0.9 0.001]	123
[0.1 0.9 0.1 0.0001]	N.A.
[0.5 0.9 0.5 0.0005]	413
[0.9 0.9 0.9 0.001]	188
[0.9 0.9 0.9 0.0001]	50

Table 4.2.2. – Learning speed for different FACL parameters, given for the head target reaching velocity controller.

FACL Parameters vector ***$[\gamma\ \lambda\ \lambda_a\ \beta_0]$***	***Number of Episodes Needed for Learning***
[0.1 0.1 0.1 0.0001]	N.A.
[0.5 0.1 0.5 0.0005]	N.A.
[0.9 0.1 0.9 0.001]	1134
[0.1 0.5 0.1 0.0001]	N.A.
[0.5 0.5 0.5 0.005]	N.A.
[0.9 0.5 0.9 0.001]	625
[0.1 0.9 0.1 0.0001]	N.A.
[0.5 0.9 0.5 0.0005]	1242
[0.9 0.9 0.9 0.001]	321
[0.9 0.9 0.9 0.0001]	163

Table 4.2.3 – Learning speed for different FACL parameters, given for the follower links' link following controller.

FACL Parameters vector ***$[\gamma\ \lambda\ \lambda_a\ \beta_0]$***	***Number of Episodes Needed for Learning***
[0.1 0.1 0.1 0.0001]	N.A.
[0.5 0.1 0.5 0.0005]	N.A.
[0.9 0.1 0.9 0.001]	4387
[0.1 0.5 0.1 0.0001]	N.A.
[0.5 0.5 0.5 0.005]	N.A.
[0.9 0.5 0.9 0.001]	2114
[0.1 0.9 0.1 0.0001]	N.A.
[0.5 0.9 0.5 0.0005]	5318
[0.9 0.9 0.9 0.001]	3154
[0.9 0.9 0.9 0.0001]	1352

Table 4.2.4 – Learning speed for different FACL parameters, given for the head obstacle avoidance controller.

FACL Parameters vector ***$[\gamma\ \lambda\ \lambda_a\ \beta_0]$***	***Number of Episodes Needed for Learning***
[0.1 0.1 0.1 0.0001]	N.A.
[0.5 0.1 0.5 0.0005]	N.A.
[0.9 0.1 0.9 0.001]	3118
[0.1 0.5 0.1 0.0001]	N.A.
[0.5 0.5 0.5 0.005]	N.A.
[0.9 0.5 0.9 0.001]	1755
[0.1 0.9 0.1 0.0001]	N.A.
[0.5 0.9 0.5 0.0005]	3741
[0.9 0.9 0.9 0.001]	2188
[0.9 0.9 0.9 0.0001]	998

Table 4.2.5 – Learning speed for different FACL parameters, given for the follower link obstacle avoidance controller.

FACL Parameters vector ***[$\gamma \lambda \lambda_a \beta_0$]***	***Number of Episodes Needed for Learning***
[0.1 0.1 0.1 0.0001]	N.A
[0.5 0.1 0.5 0.0005]	N.A
[0.9 0.1 0.9 0.001]	8622
[0.1 0.5 0.1 0.0001]	N.A
[0.5 0.5 0.5 0.005]	N.A
[0.9 0.5 0.9 0.001]	7643
[0.1 0.9 0.1 0.0001]	N.A.
[0.5 0.9 0.5 0.0005]	N.A
[0.9 0.9 0.9 0.001]	6429
[0.9 0.9 0.9 0.0001]	5622

Table 4.2.6 – Learning speed for different FACL parameters, given for the head object grasping controller.

FACL Parameters vector ***[$\gamma \lambda \lambda_a \beta_0$]***	***Number of Episodes Needed for Learning***
[0.1 0.1 0.1 0.0001]	N.A.
[0.5 0.1 0.5 0.0005]	N.A.
[0.9 0.1 0.9 0.001]	1633
[0.1 0.5 0.1 0.0001]	N.A.
[0.5 0.5 0.5 0.005]	N.A.
[0.9 0.5 0.9 0.001]	1461
[0.1 0.9 0.1 0.0001]	N.A.
[0.5 0.9 0.5 0.0005]	N.A.
[0.9 0.9 0.9 0.001]	1294
[0.9 0.9 0.9 0.0001]	963

Table 4.2.7 – Learning speed for different FACL parameters, given for the follower link object grasping controller.

We can infer from the above tables that the reinforcement learning discount factor, γ, should be close to 1 for all of our controllers; and the initial learning rate, β_0, should be in the order of 0.0001 for faster learning. Use of eligibility trace recency factors , λ and λ', close to 1 increases the learning speed for all the cases, however for more complicated controllers (complexity increases in the order: target reaching, obstacle avoidance, object grasping), their effect is more sensible since as the state space gets larger, using memory terms gains meaning.

4.3 Performance Evaluation Simulations

After the completion of the training of the controllers, the next step in the simulations is the evaluation of the performance of the complete controller architecture. This is achieved by running robot simulations in different environmental maps. These maps are arranged according to different criteria such as complexity of the obstacle placement, and object grasping situation. For the runs including only target reaching and obstacle avoidance controllers a 6-link robot simulation is used. For the runs with object grasping included, a 12-link robot simulation is used.

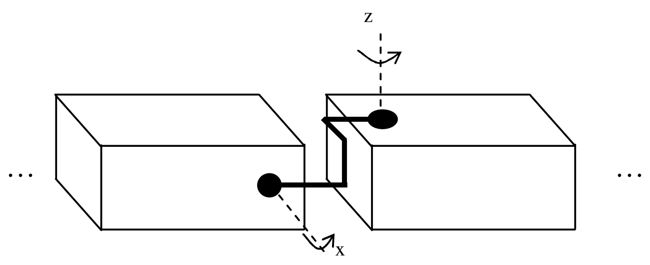


Figure 4.3.1– The structure of the robot links and joints.

It is also worth noting that the dynamics is also added to the evaluation simulations. The genetic algorithm is incorporated as the low-level part of the algorithm to implement the dynamical control. In the evaluation simulations we use links that are 50 cm in length and 5 cm in width, with a mass of 1 kg each, with a corresponding 0.0208 kg.m^2 moment of inertia with respect to center of mass. The Coulomb friction model is used with tangential and normal friction coefficients of 0.1 and 0.5 respectively. In these simulations, obstacles in the range of 50 cm from the links were assumed to be sensed (i.e., farther obstacles are not sensed). Dynamical modeling is not performed by hand-written equations, but SimMechanics software of MATLAB is used.

The robot structure used in simulations is as shown in Figure 4.3.1: Two successive links are connected to each other with a 2-DOF joint that's able to move in pitch and yaw angles. In the simulations, it is assumed that these angles are active variables (i.e., in a physical implementation there are actuators able to change these angles as desired.) and they cause the total motion of the robot. Genetic algorithm searches for suitable angle functions in time, that cause the desired displacement, as explained in Chapter 3.

Another point to note for the dynamical simulations is that for the high-level learning simulations, the controller architecture is on a planar snake robot and for its grasping the collisions between the grasping links are ignored, which in fact is unavoidable on a plane. However, for a robot that can move in 3D, precautions may be taken to avoid such collisions. This is achieved by raising the colliding link within an acceptable safety margin above the ground. If a head following link is colliding with its consecutive links, then the immediate consecutive links raise in a triangular bridge over their predecessor links to allow a passage for that preceding link. A "height envelope" around the forehand link that prevents collision in space with its succeeding links is used for this purpose. This idea is illustrated schematically in Figure 4.3.2.

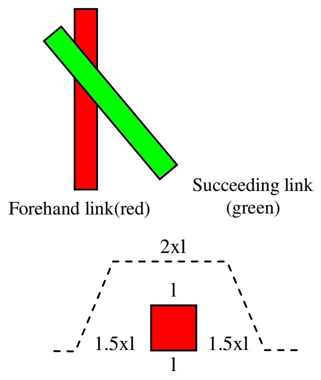


Figure 4.3.2 – A representation of two links colliding in plane, and the envelope function representation for the forehand link.

4.3.1 High-level Controller Performance Evaluation

Three different maps are used in the evaluation simulations of the classical target reaching and obstacle avoidance behaviors. These maps are shown in Figure 4.3.3. For the first map we may use the term "dense small obstacles", for the second one "rare big obstacles" and for the third one "rare long obstacles". The 6-link robot is operated in each of these environments for 100 trials. The trials in which the 6-link robot reaches to the target point without colliding with any obstacles are assumed to be successful.

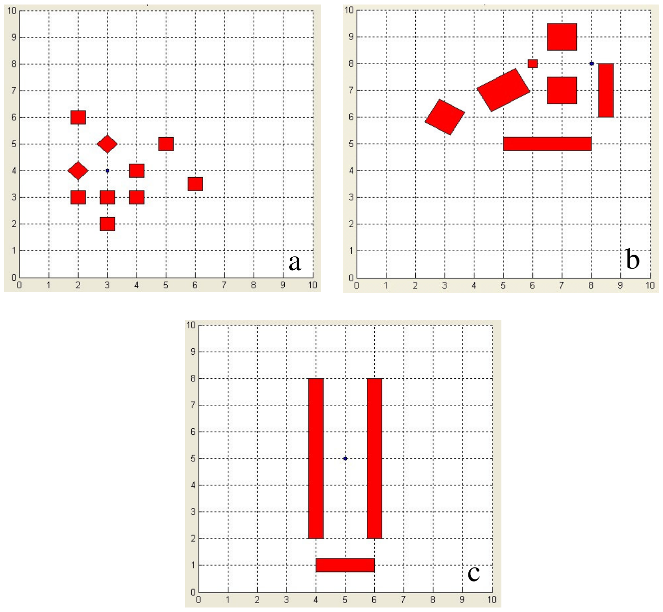


Figure 4.3.3 – Three maps used for the evaluation of the high level controller.

The results of the evaluation simulations are given in Table 4.3.1. As shown in this table, success rates are below 65% for the first two maps and above 80% for the 3rd map. This is due to the obstacle structure and the lack of training generalization. In order to improve generalization, the controllers are further trained in these environments for 100 episodes and the results are shown in the 3rd column of Table 4.3.1 as the further training (F.T.) success.

MAP	INITIAL SUCCESS	F.T. SUCCESS
a	60%	94%
b	64%	96%
c	85%	100%

Table 4.3.1 – Success percentages for the maps given in Figure 4.3.3.

Hence, our controller architecture is able to adapt itself to environments by performing further training when necessary.

For the performance evaluation of the grasping controller we use a 12-link serpentine robot simulation in an environment with obstacles, and also with an object of radius 0.5 m. The expectation of the controller is going through the object and reaching to the target after grasping it. This task is achieved as shown in the snapshots given in Figure 4.3.5. The distance of the tip of the robot's head link to the center of the object to be grasped is given in Figure 4.3.4. As shown in this figure, the robot gets closer to the object and when the object is "close enough" (after time step 121) it starts to enwrap the object, during which the object to head tip distance is almost constant.

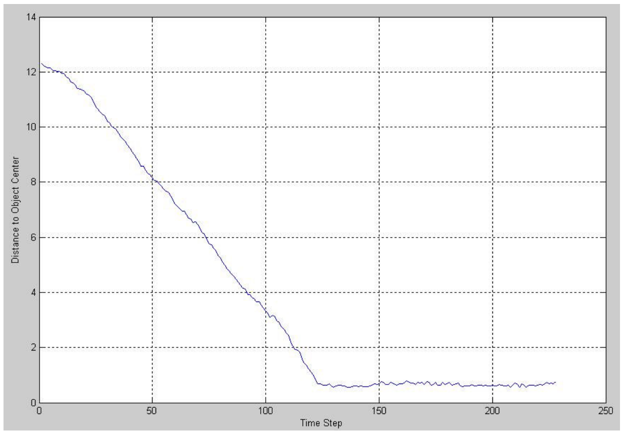


Figure 4.3.4 – Graph showing the distance of the head segment's tip point planar projection to the center of the object to be grasped during object grasping behavior.

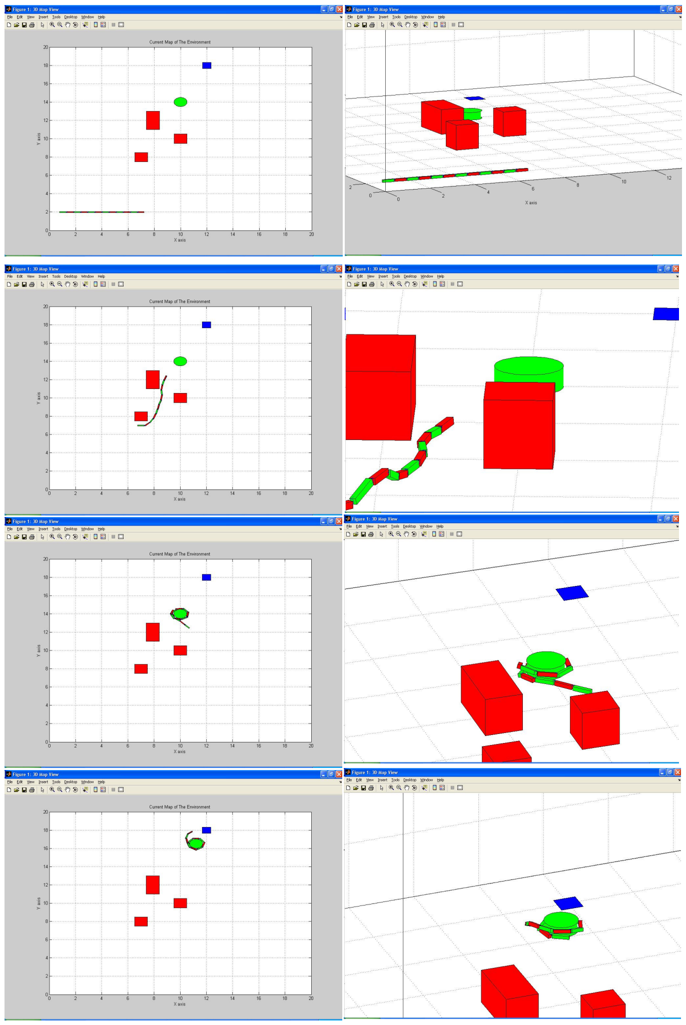


Figure 4.3.5 – Visualizations from a run of the developed high level controller, performing target reaching, obstacle avoidance and object grasping behaviors. Top views are on the left, 3-D views on the right.

Another measure that gives insight about the performance may be the α plot versus time, for the head segment. This is shown in Figure 4.3.6 for the simulation in Figure 4.3.5. As seen in this figure, the head angle controller first regulates the angle to 0; near time 60 the first obstacle is come across and the obstacle avoidance controller dominates, and the regulation is disturbed. A similar situation is seen near time step 90, where the second obstacle is come across.

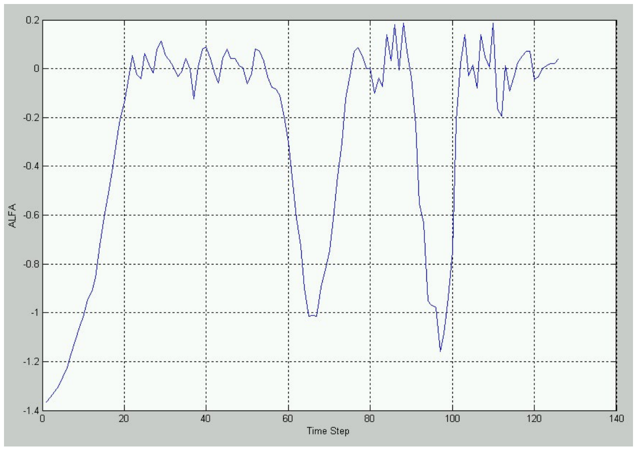


Figure 4.3.6 – Graph showing the angular deviation between the head segment and the target line (α) versus time during object grasping simulation, until the object grasping behavior is activated for the head segment.

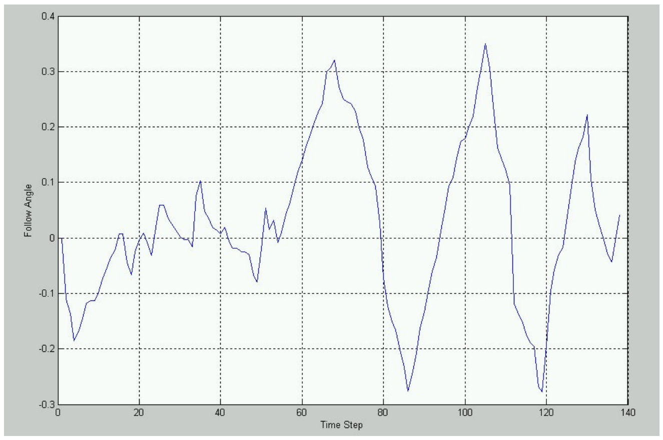


Figure 4.3.7 – Graph showing the angular deviation between the head and the following segment versus time step.

Performance of the intermediate link controllers may be evaluated using the alignment error between the head segment and the following segment. A plot of this measure is given in Figure 4.3.7 for the simulation run given in Figure 4.3.5. The first negative deviation in the angle results from the fact that the head tries to align through the target object. The big oscillations result from the obstacles on the way of the robot. When the head tries to avoid the obstacles angle goes positive, and when the follower link tries to avoid obstacles angle goes negative.

4.3.2 Genetic Algorithm Performance Evaluation

As explained in Chapter 3.3.3, a genetic algorithm search engine is employed in order to find the desired joint angle movements for creating a net displacement close enough to the displacement desired by the high-level controller. A combination of the average of position error (defined by the absolute sum of the position errors of 4 non-coplanar points in each robot link) and the energy consumption (calculated using the integral of torque – angular speed product for each active joint) is employed as the fitness function. Which may be formulated mathematically as follows:

$$\sum_{i=1}^{Nx4} E_i + 10 \cdot \sum_{j=1}^{(N-1)x2} \int_{T_s} \tau_j(t) \cdot \omega_j(t) \cdot dt$$

In this equation, E_i stands for the position error of i^{th} point, measured in centimeters. τ_j (N.m) and ω_j (rad/s) stands for torque applied at j^{th} actuator, and related angles angular speed respectively. The factor of 10 included in front of the energy some is for making the position error in centimeters comparable to the energy consumption in Joules. Critical parameters of the genetic algorithm are shown in Table 4.3.2.

Parameter	***Value Used***
Population Size	20
Crossover	Intermediate / Ratio: 0.5
Fitness Function	Position Error & Energy Consumption
Mutation	Uniform / Rate : 0.05
Elite Count	2
Selection Rule	Tournament

Table 4.3.2 – Critical parameters of the genetic algorithm search module.

The performance evaluation of the genetic algorithm is performed for the 6-link and 12-link robots separately. For the 6-link robot, runs from the 3 maps shown in Figure 4.3.3 are considered. For the 12-link robot, the run shown in Figure 4.3.5 is considered.

MAP	***NUMBER OF STEPS***	***AVERAGE NUMBER OF GENERATIONS***
a	143	6.35
b	155	6.12
c	203	6.36

Table 4.3.3 – Performance of the genetic algorithm search module for the 6-link robot.

Number of steps	Average Number of Generations
226	18.37

Table 4.3.4 – Performance of the genetic algorithm search module for the 12-link robot.

The results are shown in Table 4.3.3 , for the 6-link robot, and in Table 4.3.4 for the 12-link robot. The genetic algorithm uses about *6.25* generations at each time step. For the 12-link robot, however, the average number of generations increases to 18.37.

In the genetic algorithm implementation, we assume that each of the joint angles consists of 10 sinusoids in time, and the frequency and amplitude component of each angle are equal, but the phases of these angles are different in general. Hence, i^{th} joint angle may be written in time (during a single sample period) as

$$\theta_i(t) = \sum_{j=1}^{10} A_j \cdot \sin(w_j t + \rho_{ij})$$

This means that or a robot having *N* links, we have *10+10+(N-1)x10x2* parameters as the input vector of the fitness function (chromosomes) (*10* for amplitude, 10 for frequency coding, *(N-1)x10x2* for phase coding, refer to Chapter 3.3.3.). This means, number of parameters increases as the number of links of the robot increases. Hence, genetic algorithm may limit the number of segments of the robot used.

Another point to stress for the high level FACL controllers and lower level genetic algorithm interaction is the error between the final robot configuration desired by the FACL controllers and the configuration reached by the genetic algorithm. In our approach, we use a fitness function which is a linear combination of the cumulative position error and the total energy consumption. Hence, zeroing of position error should not be ideal. The high level controller desires a next position, but genetic algorithm could not exactly reach it; rather it approaches that position. The cumulative position error (sum of absolute values of errors in position of the 4 vertices of a prismatic link for all robot links) average for the demo map shown in Figure 4.2.5 is *46.3* cm. Hence for a single vertex, this error is *(46.3 / 12)/4 = 0,963 cm*.

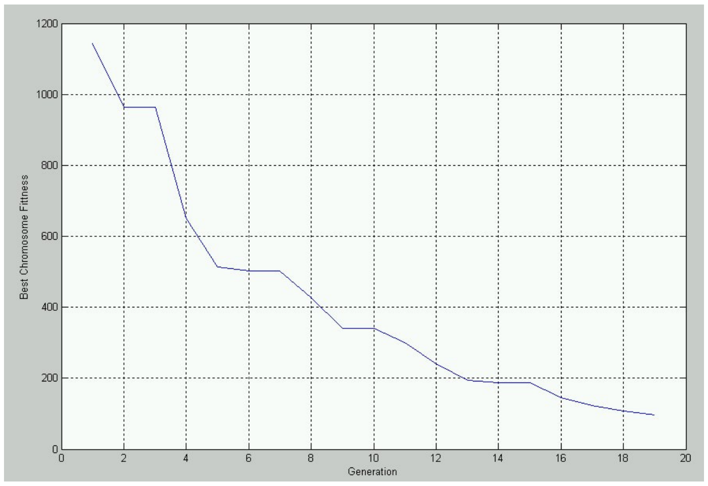


Figure 4.3.8 – An example of fitness function versus number of generations for a single genetic algorithm run during the case shown in Figure 4.3.5.

A standard evaluation for genetic algorithms is the fitness function change in between generations. This change is shown in Figure 4.3.8 for a single run of genetic algorithm during the simulation shown in Figure 4.3.5. The fitness function threshold for stopping is set at 100. In the figure, it is seen that the fitness function starts from 1143 at generation 1 and drops down to 96.11 at generation 19 when it stops.

4.4 Sensitivity Analysis

In most of the studies in the literature ([20], [23], [24],[26],[29]]), the parameters of fuzzy logic controllers, neural networks or reinforcement learning algorithms are chosen by trial and error. We also follow the same way in developing our controller. However, a simplified mathematical analysis of the sensitivity of controller outputs to the modifiable parameters may help in gaining insight of how to select these parameters. In this part of the report, we give the results of the sensitivity analysis for the head target reaching angle controller, since it is the simplest controller used.

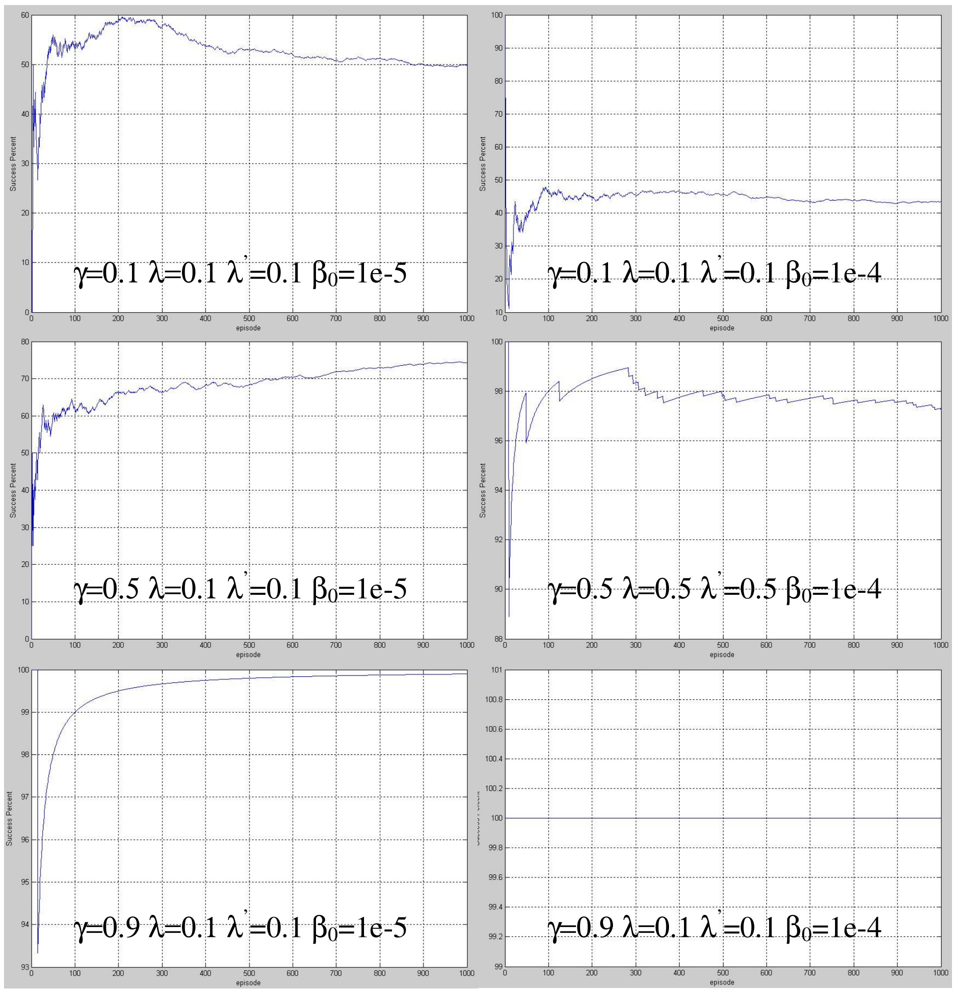


Figure 4.4.1 - Head Angle Controller learning performance for different parameter selections of reinforcement learning.

In Figure 4.4.1, the success percentage of head angle controller is plotted with respect to number of episodes. The reinforcement learning parameters used for each run is shown on the corresponding figure. This figure demonstrates how critical the parameters selection is effective in the speed of learning, even in the ability of learning of the controller.

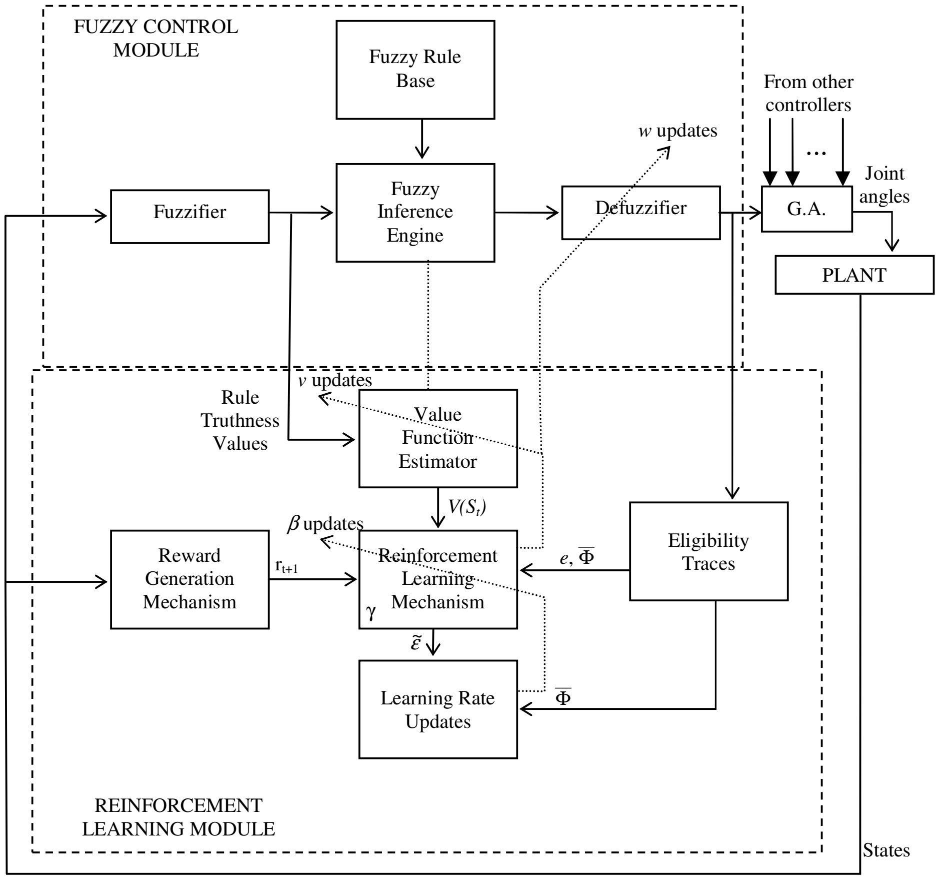


Figure 4.4.2 – Schematic diagram showing the relations between the controller components.

Before passing to the details of the sensitivity analysis we should give the simplified architecture of our controller as shown in Figure 4.4.2. Here, it is evident to see that the controller consists of two components. One of these components is named as the “Fuzzy Control Module”. The remaining parts are named as "Reinforcement Learning Module". In our sensitivity analysis, we perform the analysis of these two units separately.

4.4.1 Fuzzy Control Module Sensitivity Analysis

For the fuzzy controller sensitivity analysis, we investigate the relation between a parameter of the fuzzy control module and the output of the fuzzy control module. The derivatives give us an insight of sensitivity. For the output part of the fuzzy control module, rewriting equation (2.23) given before

$$Y_m(S) = \sum_{R_i \in A} \alpha_{R_i}(S) o_m^i \qquad , \mathrm{m} = 1,...,\mathrm{N_O} \qquad (2.23)$$

Hence, sensitivity of the output of the fuzzy controller to the activation value of rule R_i, is measured by using:

$$\frac{\partial Y_m(S)}{\partial \alpha_{R_i}(S)} = o_m^i \qquad (4.1)$$

The sensitivity of the fuzzy controller output to the discrete action offered by rule R_i, is measured using:

$$\frac{\partial Y_m(S)}{\partial o_m^i} = \alpha_{R_i}(S) \qquad (4.2)$$

In equations (4.1) and (4.2), it is seen that the output of the FIS module is sensitive to the truthness value of a given rule in proportion to the local action offered by that rule. Similarly, FIS module output is more sensitive to the local action offered by a rule as the truthness increases.

Remembering the equation giving the activation value of rule R_i (equation (2.22)):

$$\alpha_{R_i}(S) = \prod_{j=1}^{N_i} \mu_{L_j^i}(S_j) \qquad (2.22)$$

The sensitivity of activation of rule R$_i$ to the related jth input fuzzy set membership function value is

$$\frac{\partial \alpha_{R_i}(S)}{\partial \mu_{L_j^i}(S_j)} = \prod_{k=1,k\neq j}^{N_i} \mu_{L_k^i}(S_k) \qquad (4.3)$$

Hence, the sensitivity of rule truthness to an individual input set membership value in the antecedents of that rule increases as the membership values of the other antecedent variable fuzzy sets increase.

Remembering the general trapezoidal fuzzy membership function equation (equation (2.19)):

$$\mu_{L_j^i}(S_j) = \begin{cases} \max\left(0.0, 1.0 - \dfrac{(S_j - v_r^{L_j^i})}{s_r^{L_j^i}}\right) & , S_j > v_r^{L_j^i} \\ \max\left(0.0, 1.0 - \dfrac{(v_l^{L_j^i} - S_j)}{s_l^{L_j^i}}\right) & , S_j < v_l^{L_j^i} \\ 1.0 & ,\text{otherwise} \end{cases} \qquad (2.19)$$

Hence, a measure of sensitivity of the membership function value to the right spread, right vertex, left spread and left vertex sensitivities are as follows, respectively:

$$\frac{\partial \mu_{L_j^i}(S_j)}{\partial s_r^{L_j^i}} = \begin{cases} \dfrac{(S_j - v_r^{L_j^i})}{(s_r^{L_j^i})^2} & , \text{if } v_r^{L_j^i} < S_j < v_r^{L_j^i} + s_r^{L_j^i} \\ 0 & , \text{otherwise} \end{cases} \qquad (4.4)$$

$$\frac{\partial \mu_{L_j^i}(S_j)}{\partial v_r^{L_j^i}} = \begin{cases} \dfrac{1}{s_r^{L_j^i}} & , \text{if } v_r^{L_j^i} < S_j < v_r^{L_j^i} + s_r^{L_j^i} \\ 0 & , \text{otherwise} \end{cases} \qquad (4.5)$$

$$\frac{\partial \mu_{L_j^i}(S_j)}{\partial s_l^{L_j^i}} = \begin{cases} \dfrac{(v_l^{L_j^i} - S_j)}{(s_l^{L_j^i})^2} & \text{, if } v_l^{L_j^i} - s_l^{L_j^i} < S_j < v_l^{L_j^i} \\ 0 & \text{, otherwise} \end{cases} \tag{4.6}$$

$$\frac{\partial \mu_{L_j^i}(S_j)}{\partial v_l^{L_j^i}} = \begin{cases} -\dfrac{1}{s_l^{L_j^i}} & \text{, if } v_l^{L_j^i} - s_l^{L_j^i} < S_j < v_l^{L_j^i} \\ 0 & \text{, otherwise} \end{cases} \tag{4.7}$$

From equations (4.4) to (4.7), one can infer that the sensitivity of trapezoidal fuzzy set membership function value is affected by variables related to “that side” of the trapezoid. Moreover, the sensitivity to vertex values are inversely proportional with the spread values. Note also that, when trapezoidal membership function value is “1” it is not sensitive to differential changes in the parameters of the trapezoid.

The analysis described above may be performed for any controller and any parameter of the selected controller. In this study we perform it for the head angle controller of the target reaching behavior, since this controller has a simple structure making it suitable for demonstration.

α \ z	**0**	**2.22**	**4.44**	**6.66**	**8.88**	**11.11**	**13.33**	**15.55**	**17.77**	**20**
-π	1	1	[1;2]	[1;2]	[2;3]	[2;3]	[3;4]	[3;4]	4	4
-7π/9	[1;5]	[1;5]	[1;2;5;6]	[1;2;5;6]	[2;3;6;7]	[2;3;6;7]	[3;4;7;8]	[3;4;7;8]	[4;8]	[4;8]
-5π/3	[5;9]	[5;9]	[5;6;9;10]	[5;6;9;10]	[6;7;10;11]	[6;7;10;11]	[7;8;11;12]	[7;8;11;12]	[8;12]	[8;12]
-3π/9	[5;9]	[5;9]	[5;6;9;10]	[5;6;9;10]	[6;7;10;11]	[6;7;10;11]	[7;8;11;12]	[7;8;11;12]	[8;12]	[8;12]
-π/9	[9;13]	[9;13]	[9;10;13;14]	[9;10;13;14]	[10;11;14;15]	[10;11;14;15]	[11;12;15;16]	[11;12;15;16]	[12;16]	[12;16]
π/9	[13;17]	[13;17]	[13;14;17;18]	[13;14;17;18]	[14;15;18;19]	[14;15;18;19]	[15;16;19;20]	[15;16;19;20]	[16;20]	[16;20]
3π/9	[13;17]	[13;17]	[13;14;17;18]	[13;14;17;18]	[14;15;18;19]	[14;15;18;19]	[15;16;19;20]	[15;16;19;20]	[16;20]	[16;20]
5π/9	[17;21]	[17;21]	[17;18;21;22]	[17;18;21;22]	[18;19;22;23]	[18;19;22;23]	[19;20;23;24]	[19;20;23;24]	[20;24]	[20;24]
7π/9	[21;25]	[21;25]	[21;22;25;26]	[21;22;25;26]	[22;23;26;27]	[22;23;26;27]	[23;24;27;28]	[23;24;27;28]	[24;28]	[24;28]
π	25	25	[25;26]	[25;26]	[26;27]	[26;27]	[27;28]	[27;28]	28	28

Table 4.4.1 – Table of activated rules, for 100 sample states from the (α,z) state space for the target reaching head angle controller.

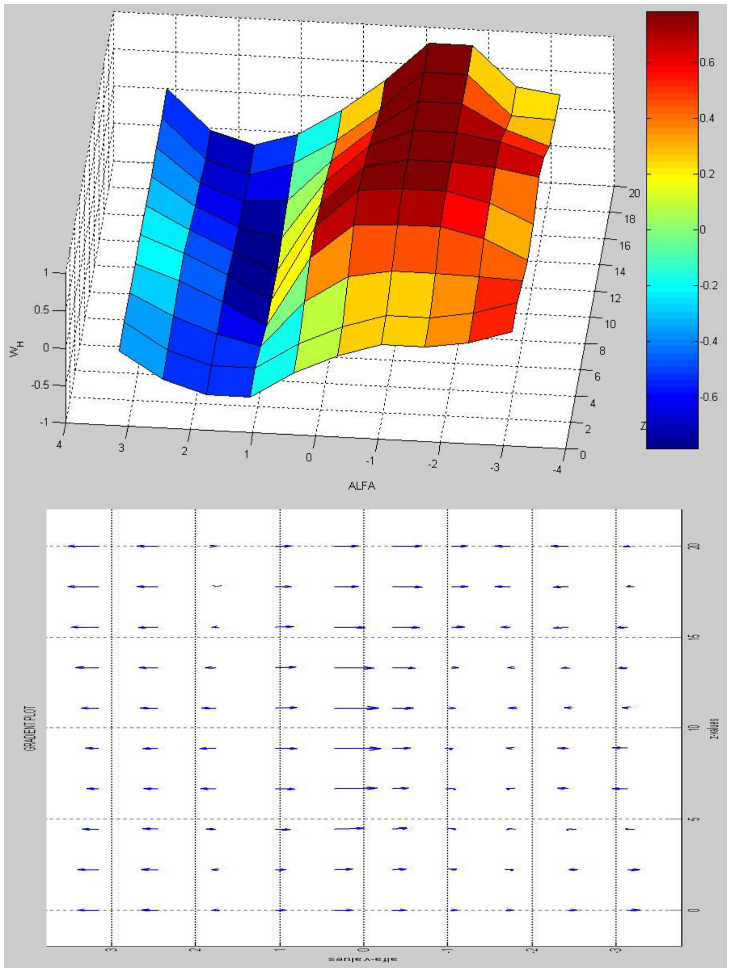


Figure 4.4.3 – Input-Output mapping and its gradient plot for the target reaching head angle fuzzy controller.

In Figure 4.4.3, the Input-Output mapping of the fuzzy inference system is plotted for 100 samples from the (α,z) space. Since the output is the angular velocity of the head segment, for negative values of α it should be positive to increase α through 0. Similarly, for positive values of α, it should be negative to decrease α through 0. Note in Figure 4.4.3 that the output is positive for negative values of α and negative for positive values of α, and changes in z do not affect the form of the output severely, as expected. This figure visually represents equation (23).

In Table 4.4.1, the indices of the activated rules are given for the same 100 samples of (α,z) values. This table represents A_t for 100 different values of S_t.

For simplified, representative sensitivity analysis, we consider 4 different points in the (α,z) space:

- For (-7π/9, 2.22): $A_t = \{R_1,R_5\}$

$\alpha_{R_1} = 0.333$, $o^1 = 0.5236$, $\mu_{L_1^1} = 0.333, \mu_{L_2^1} = 1$

$s_l^{L_1^1} = 0, v_l^{L_1^1} = -3.1416, s_r^{L_1^1} = 1.0472, v_r^{L_1^1} = -3.1416$

$s_l^{L_2^1} = 0, v_l^{L_2^1} = 0, s_r^{L_2^1} = 5, v_r^{L_2^1} = 2.5$

$\alpha_{R_5} = 0.667$, $o^5 = 0.2618$, $\mu_{L_1^5} = 0.667, \mu_{L_2^5} = 1$

$s_l^{L_1^5} = 1.0472, v_l^{L_1^5} = -2.0944, s_r^{L_1^5} = 1.0472, v_r^{L_1^5} = -2.0944$

$s_l^{L_2^5} = 0, v_l^{L_2^5} = 0, s_r^{L_2^5} = 5, v_r^{L_2^5} = 2.5$

Using these numerical values in equations (4.1) and (4.7), we get:

$$\frac{\partial Y(S)}{\partial \alpha_{R_1}(S)} = 0.5236, \frac{\partial Y(S)}{\partial \alpha_{R_5}(S)} = 0.2618$$

$$\frac{\partial Y(S)}{\partial o^1(S)} = 0.333, \frac{\partial Y(S)}{\partial o^5(S)} = 0.667$$

From above values, it is seen that the output is more sensitive to the truthness value of rule-1 than rule-5. This is because the action offered by rule-1 is greater (in value) than the action offered by rule-5. However, note that the output is less sensitive to the local action offered by rule-1 than the local action offered by rule-5. Again, this is expected since the truthness value of rule-1 is less than truthness value of rule-5 for the given state value.

For the sensitivity of the rule truthness values, we have the following numerical values:

$$\frac{\partial \alpha_{R_1}(S)}{\partial \mu_{L_1^1}(S)} = 1, \ \frac{\partial \alpha_{R_1}(S)}{\partial \mu_{L_2^1}(S)} = 0.333; \frac{\partial \alpha_{R_5}(S)}{\partial \mu_{L_1^5}(S)} = 1, \frac{\partial \alpha_{R_5}(S)}{\partial \mu_{L_2^5}(S)} = 0.667$$

The sensitivity of the truthness of *rule-1* to the membership value of the related fuzzy set of the α variable is *1*. This is since the *z* fuzzy set used for this rule has the top of the trapezoid for *z = 2.22*. Similar comments hold for rule-5 activation value's sensitivity to the related fuzzy set membership value of the α variable. Note that the sensitivities of *rule-1* and *rule-5* activation values to the *z*-variable membership values actually compete. This is due to the strong fuzzy partitioning used. As the two triangle membership values of the α variable intersect, they (hence the sensitivity to the value of the *z* membership function) sum up to *1* at every intersection point.

For the sensitivities of the four individual fuzzy sets whose values are non-zero for the given value of states, we have the following numerical values:

$$\frac{\partial \mu_{L_1^1}(S)}{\partial s_r^{L_1^1}} = 0.6337, \frac{\partial \mu_{L_1^1}(S)}{\partial v_r^{L_1^1}} = 0.9549, \frac{\partial \mu_{L_1^1}(S)}{\partial s_l^{L_1^1}} = 0, \frac{\partial \mu_{L_1^1}(S)}{\partial v_l^{L_1^1}} = 0$$

$$\frac{\partial \mu_{L_2^1}(S)}{\partial s_r^{L_2^1}} = 0, \frac{\partial \mu_{L_2^1}(S)}{\partial v_r^{L_2^1}} = 0, \frac{\partial \mu_{L_2^1}(S)}{\partial s_l^{L_2^1}} = 0, \frac{\partial \mu_{L_2^1}(S)}{\partial v_l^{L_2^1}} = 0$$

$$\frac{\partial \mu_{L_1^5}(S)}{\partial s_r^{L_1^5}} = 0, \frac{\partial \mu_{L_1^5}(S)}{\partial v_r^{L_1^5}} = 0, \frac{\partial \mu_{L_1^5}(S)}{\partial s_l^{L_1^5}} = 0.3183, \frac{\partial \mu_{L_1^5}(S)}{\partial v_l^{L_1^5}} = -0.9549$$

$$\frac{\partial \mu_{L_2^5}(S)}{\partial s_r^{L_2^5}} = 0, \frac{\partial \mu_{L_2^5}(S)}{\partial v_r^{L_2^5}} = 0, \frac{\partial \mu_{L_2^5}(S)}{\partial s_l^{L_2^5}} = 0, \frac{\partial \mu_{L_2^5}(S)}{\partial v_l^{L_2^5}} = 0$$

Note from the above values that the sensitivity of the membership function value used for the *z*-variable is 0 to all its parameters. Actually, the same *z*-fuzzy set is used for both *rule-1* and *rule-5* and the given *z*-value corresponds to the top of the trapezoid, where its value is insensitive to differential changes in its parameters. Note also that the membership function value sensitivities are *0* for triangular membership functions of α-variable for the "other side" variables of the fuzzy set. Similar comments may be declared for the values evaluated below:

- For (-7π/9, 17.77): $A_t = \{R_4, R_8\}$

$\alpha_{R_4} = 0.333$, $o^4 = -0.7854$, $\mu_{L_1^4} = 0.333$, $\mu_{L_2^4} = 1$

$s_l^{L_1^4} = 0, v_l^{L_1^4} = -3.1416, s_r^{L_1^4} = 1.0472, v_r^{L_1^4} = -3.1416$

$s_l^{L_2^4} = 5, v_l^{L_2^4} = 17.5, s_r^{L_2^4} = \infty, v_r^{L_2^4} = \infty$

$\alpha_{R_8} = 0.667$, $o^8 = 0.7854$, $\mu_{L_1^8} = 0.667, \mu_{L_2^8} = 1$

$s_l^{L_1^8} = 1.0472, v_l^{L_1^8} = -2.0944, s_r^{L_1^8} = 1.0472, v_r^{L_1^8} = -2.0944$

$s_l^{L_2^8} = 5, v_l^{L_2^8} = 17.5, s_r^{L_2^8} = \infty, v_r^{L_2^8} = \infty$

Using these numerical values in equations (4.1) to (4.7), we get:

$$\frac{\partial Y(S)}{\partial \alpha_{R_4}(S)} = -0.7854, \frac{\partial Y(S)}{\partial \alpha_{R_8}(S)} = 0.7854$$

$$\frac{\partial Y(S)}{\partial o^4(S)} = 0.333, \ \frac{\partial Y(S)}{\partial o^8(S)} = 0.667$$

$$\frac{\partial \alpha_{R_4}(S)}{\partial \mu_{L_1^4}(S)} = 1, \ \frac{\partial \alpha_{R_4}(S)}{\partial \mu_{L_2^4}(S)} = 0.333; \frac{\partial \alpha_{R_8}(S)}{\partial \mu_{L_1^8}(S)} = 1, \frac{\partial \alpha_{R_8}(S)}{\partial \mu_{L_2^8}(S)} = 0.667$$

$$\frac{\partial \mu_{L_1^4}(S)}{\partial s_r^{L_1^4}} = 0.6366, \frac{\partial \mu_{L_1^4}(S)}{\partial v_r^{L_1^4}} = 0.9549, \frac{\partial \mu_{L_1^4}(S)}{\partial s_l^{L_1^4}} = 0, \frac{\partial \mu_{L_1^4}(S)}{\partial v_l^{L_1^4}} = 0$$

$$\frac{\partial \mu_{L_2^4}(S)}{\partial s_r^{L_2^4}} = 0, \frac{\partial \mu_{L_2^4}(S)}{\partial v_r^{L_2^4}} = 0, \ \frac{\partial \mu_{L_2^4}(S)}{\partial s_l^{L_2^4}} = 0, \frac{\partial \mu_{L_2^4}(S)}{\partial v_l^{L_2^4}} = 0$$

$$\frac{\partial \mu_{L_1^8}(S)}{\partial s_r^{L_1^8}} = 0, \frac{\partial \mu_{L_1^8}(S)}{\partial v_r^{L_1^8}} = 0, \frac{\partial \mu_{L_1^8}(S)}{\partial s_l^{L_1^8}} = 0.3183, \frac{\partial \mu_{L_1^8}(S)}{\partial v_l^{L_1^8}} = -0.9549$$

$$\frac{\partial \mu_{L_2^8}(S)}{\partial s_r^{L_2^8}} = 0, \frac{\partial \mu_{L_2^8}(S)}{\partial v_r^{L_2^8}} = 0, \ \frac{\partial \mu_{L_2^8}(S)}{\partial s_l^{L_2^8}} = 0, \frac{\partial \mu_{L_2^8}(S)}{\partial v_l^{L_2^8}} = 0$$

- For (7π/9, 2.22): $A_t = \{R_{21}, R_{25}\}$

$\alpha_{R_{21}}$ = 0.667 , o^{21} = -0.5236, $\mu_{L_1^{21}} = 0.667, \mu_{L_2^{21}} = 1$

$s_l^{L_1^{21}} = 1.0472\,, v_l^{L_1^{21}} = 2.0944\,, s_r^{L_1^{21}} = 1.0472\,, v_r^{L_1^{21}} = 2.0944$

$s_l^{L_2^{21}} = 0\,, v_l^{L_2^{21}} = 0\,, s_r^{L_2^{21}} = 5\,, v_r^{L_2^{21}} = 2.5$

$\alpha_{R_{25}}$ = 0.333 , o^{25} = -0.2618, $\mu_{L_1^{25}} = 0.333, \mu_{L_2^{25}} = 1$

$s_l^{L_1^{25}} = 1.0472\,, v_l^{L_1^{25}} = 3.1416\,, s_r^{L_1^{25}} = 0\,, v_r^{L_1^{25}} = 3.1416$

$s_l^{L_2^{25}} = 0\,, v_l^{L_2^{25}} = 0\,, s_r^{L_2^{25}} = 5\,, v_r^{L_2^{25}} = 2.5$

Using these numerical values in equations (4.1) to (4.7), we get:

$$\frac{\partial Y(S)}{\partial \alpha_{R_{21}}(S)} = -0.5236\,, \frac{\partial Y(S)}{\partial \alpha_{R_{25}}(S)} = -0.2618$$

$$\frac{\partial Y(S)}{\partial o^{21}(S)} = 0.667\,,\ \frac{\partial Y(S)}{\partial o^{25}(S)} = 0.333$$

$$\frac{\partial \alpha_{R_{21}}(S)}{\partial \mu_{L_1^{21}}(S)} = 1\,,\ \frac{\partial \alpha_{R_{21}}(S)}{\partial \mu_{L_2^{21}}(S)} = 0.667\,; \frac{\partial \alpha_{R_{25}}(S)}{\partial \mu_{L_1^{25}}(S)} = 1\,, \frac{\partial \alpha_{R_{25}}(S)}{\partial \mu_{L_2^{25}}(S)} = 0.333$$

$$\frac{\partial \mu_{L_1^{21}}(S)}{\partial s_r^{L_1^{21}}} = 0.3183\,, \frac{\partial \mu_{L_1^{21}}(S)}{\partial v_r^{L_1^{21}}} = 0.9549\,, \frac{\partial \mu_{L_1^{21}}(S)}{\partial s_l^{L_1^{21}}} = 0\,, \frac{\partial \mu_{L_1^{21}}(S)}{\partial v_l^{L_1^{21}}} = 0$$

$$\frac{\partial \mu_{L_2^{21}}(S)}{\partial s_r^{L_2^{21}}} = 0\,, \frac{\partial \mu_{L_2^{21}}(S)}{\partial v_r^{L_2^{21}}} = 0\,,\ \frac{\partial \mu_{L_2^{21}}(S)}{\partial s_l^{L_2^{21}}} = 0\,, \frac{\partial \mu_{L_2^{21}}(S)}{\partial v_l^{L_2^{21}}} = 0$$

$$\frac{\partial \mu_{L_1^{25}}(S)}{\partial s_r^{L_1^{25}}} = 0\,, \frac{\partial \mu_{L_1^{25}}(S)}{\partial v_r^{L_1^{25}}} = 0\,, \frac{\partial \mu_{L_1^{25}}(S)}{\partial s_l^{L_1^{25}}} = 0.6366\,, \frac{\partial \mu_{L_1^{25}}(S)}{\partial v_l^{L_1^{25}}} = -0.9549$$

$$\frac{\partial \mu_{L_2^{25}}(S)}{\partial s_r^{L_2^{25}}} = 0\,, \frac{\partial \mu_{L_2^{25}}(S)}{\partial v_r^{L_2^{25}}} = 0\,,\ \frac{\partial \mu_{L_2^{25}}(S)}{\partial s_l^{L_2^{25}}} = 0\,,\ \frac{\partial \mu_{L_2^{25}}(S)}{\partial v_l^{L_2^{25}}} = 0$$

- For (7π/9, 17.77): $A_t = \{R_{24}, R_{28}\}$

$\alpha_{R_{24}} = 0.667$, $o^{24} = -0.7854$, $\mu_{L_1^{24}} = 0.667, \mu_{L_2^{24}} = 1$

$s_l^{L_1^{24}} = 1.0472$, $v_l^{L_1^{24}} = 2.0944$, $s_r^{L_1^{24}} = 1.0472$, $v_r^{L_1^{24}} = 2.0944$

$s_l^{L_2^{24}} = 5$, $v_l^{L_2^{24}} = 17.5$, $s_r^{L_2^{24}} = \infty$, $v_r^{L_2^{24}} = \infty$

$\alpha_{R_{28}} = 0.333$, $o^{28} = 0.7854$, $\mu_{L_1^{28}} = 0.333, \mu_{L_2^{28}} = 1$

$s_l^{L_1^{28}} = 1.0472$, $v_l^{L_1^{28}} = 3.1416$, $s_r^{L_1^{28}} = 0$, $v_r^{L_1^{28}} = 3.1416$

$s_l^{L_2^{28}} = 5$, $v_l^{L_2^{28}} = 17.5$, $s_r^{L_2^{28}} = \infty$, $v_r^{L_2^{28}} = \infty$

Using these numerical values in equations (4.1) to (4.7), we get:

$$\frac{\partial Y(S)}{\partial \alpha_{R_{24}}(S)} = -0.7854, \frac{\partial Y(S)}{\partial \alpha_{R_{28}}(S)} = 0.7854$$

$$\frac{\partial Y(S)}{\partial o^{24}(S)} = 0.667, \frac{\partial Y(S)}{\partial o^{28}(S)} = 0.333$$

$$\frac{\partial \alpha_{R_{24}}(S)}{\partial \mu_{L_1^{24}}(S)} = 1, \frac{\partial \alpha_{R_{24}}(S)}{\partial \mu_{L_2^{24}}(S)} = 0.667; \frac{\partial \alpha_{R_{28}}(S)}{\partial \mu_{L_1^{28}}(S)} = 1, \frac{\partial \alpha_{R_{28}}(S)}{\partial \mu_{L_2^{28}}(S)} = 0.333$$

$$\frac{\partial \mu_{L_1^{24}}(S)}{\partial s_r^{L_1^{24}}} = 0.3183, \frac{\partial \mu_{L_1^{24}}(S)}{\partial v_r^{L_1^{24}}} = 0.9549, \frac{\partial \mu_{L_1^{24}}(S)}{\partial s_l^{L_1^{24}}} = 0, \frac{\partial \mu_{L_1^{24}}(S)}{\partial v_l^{L_1^{24}}} = 0$$

$$\frac{\partial \mu_{L_2^{24}}(S)}{\partial s_r^{L_2^{24}}} = 0, \frac{\partial \mu_{L_2^{24}}(S)}{\partial v_r^{L_2^{24}}} = 0, \frac{\partial \mu_{L_2^{24}}(S)}{\partial s_l^{L_2^{24}}} = 0, \frac{\partial \mu_{L_2^{24}}(S)}{\partial v_l^{L_2^{24}}} = 0$$

$$\frac{\partial \mu_{L_1^{28}}(S)}{\partial s_r^{L_1^{28}}} = 0, \frac{\partial \mu_{L_1^{28}}(S)}{\partial v_r^{L_1^{28}}} = 0, \frac{\partial \mu_{L_1^{28}}(S)}{\partial s_l^{L_1^{28}}} = 0.6366, \frac{\partial \mu_{L_1^{28}}(S)}{\partial v_l^{L_1^{28}}} = -0.9549$$

$$\frac{\partial \mu_{L_2^{28}}(S)}{\partial s_r^{L_2^{28}}} = 0, \frac{\partial \mu_{L_2^{28}}(S)}{\partial v_r^{L_2^{28}}} = 0, \frac{\partial \mu_{L_2^{28}}(S)}{\partial s_l^{L_2^{28}}} = 0, \frac{\partial \mu_{L_2^{28}}(S)}{\partial v_l^{L_2^{28}}} = 0$$

Similarly, we may calculate the sensitivity measures for different state values, and for different controllers.

4.4.2 Reinforcement Learning Module Sensitivity Analysis

For the reinforcement learning sensitivity analysis, we may start with the reward equation, which is dependent on the control aim:

$$r_{t+1} = f\left(S_{t+1}, S_t, \overline{n}\right) \qquad (4.8)$$

$\overline{n}$ being a parameter vector. When the reward is fed back to the agent, the first operation performed is calculating the temporal difference error estimation, as given in Chapter 2:

$$\tilde{\varepsilon}_{t+1} = r_{t+1} + \gamma \cdot V_t(S_{t+1}) - V_t(S_t) \qquad (2.26)$$

where the current and future state value estimates, using the current critic function, are

$$V_t(S_t) = \sum_{R_i \in A_t} v_t^i \cdot \alpha_{R_i}(S_t) \qquad (2.24)$$

$$V_t(S_{t+1}) = \sum_{R_i \in A_{t+1}} v_t^i \cdot \alpha_{R_i}(S_{t+1}) \quad (4.9)$$

the individual value estimates for the rules are updated using

$$v_{t+1} = v_t + \beta_t \cdot \tilde{\varepsilon}_{t+1} \cdot \overline{\Phi}_t \qquad (2.39)$$

giving

$$v_t^i = v_{t-1}^i + \beta_{t-1}^i \cdot \tilde{\varepsilon}_t \cdot \overline{\Phi}_{t-1}^i \qquad (4.10)$$

hence, the temporal difference error equation given in (2.26) may be rewritten as follows:

$$\tilde{\varepsilon}_{t+1} = r_{t+1} + \gamma . \sum_{R_i \in A_{t+1}} v_t^i . \alpha_{R_i}(S_{t+1}) - \sum_{R_j \in A_t} v_t^j . \alpha_{R_j}(S_t) \qquad (4.11)$$

The update rule for the eligibility trace is

$$\overline{\Phi}_t = \Phi_t + \gamma \cdot \lambda \cdot \overline{\Phi}_{t-1} \quad (2.35)$$

And for the learning rate, we have

$$\Delta\beta_t^i = \begin{cases} \kappa & \text{,if } \overline{\delta}_{t-1}^i \cdot \delta_t^i > 0 \\ -\sigma \cdot \beta_t^i & \text{,if } \overline{\delta}_{t-1}^i \cdot \delta_t^i < 0 \\ 0 & \text{,otherwise} \end{cases} \quad (2.38)$$

With $\delta_t^i = \tilde{\varepsilon}_{t+1} \cdot \overline{\Phi}_t^i$ and $\overline{\delta}_t^i = (1-\varphi)\delta_t^i + \varphi\overline{\delta}_{t-1}^i$. We may say the following where h(.) is the piecewise relation given in (2.38)

$$\Delta\beta_t = h(\overline{\delta}_{t-1}, \delta_t, \beta_t) \quad (4.12)$$

All these relations show partial dynamics of the actor and may be represented in a discrete-time graph as in Figure 4.4.4.

For the actor part of the reinforcement learning module the individual rule-action weights are updated according to equation (2.37) given in Chapter 2, as follows:

$$w_{t+1} = w_t + \tilde{\varepsilon}_{t+1} \cdot e_t \quad (2.37)$$

Where e_t is the eligibility trace (it is a matrix) for the action weights, where for rule R_i it is calculated as given in equation

$$e_t^i(U^i) = \begin{cases} \lambda' \cdot e_{(t-1)}^i(U^i) + \Phi_t^i & \text{, if } U^i = U_t^i \\ \lambda' \cdot e_{(t-1)}^i(U^i) & \text{, otherwise} \end{cases} \quad (2.36)$$

We may represent this relation with a function g(.,.) as

$$e_t^i(U^i) = g(e_{(t-1)}^i, \Phi_t^i) \quad (4.13)$$

Using the weights of individual rule-actions, the resultant output action is calculated using the epsilon-greedy process described in Chapter 2, and upon application of this value to the system, the state dynamics determine the next state.

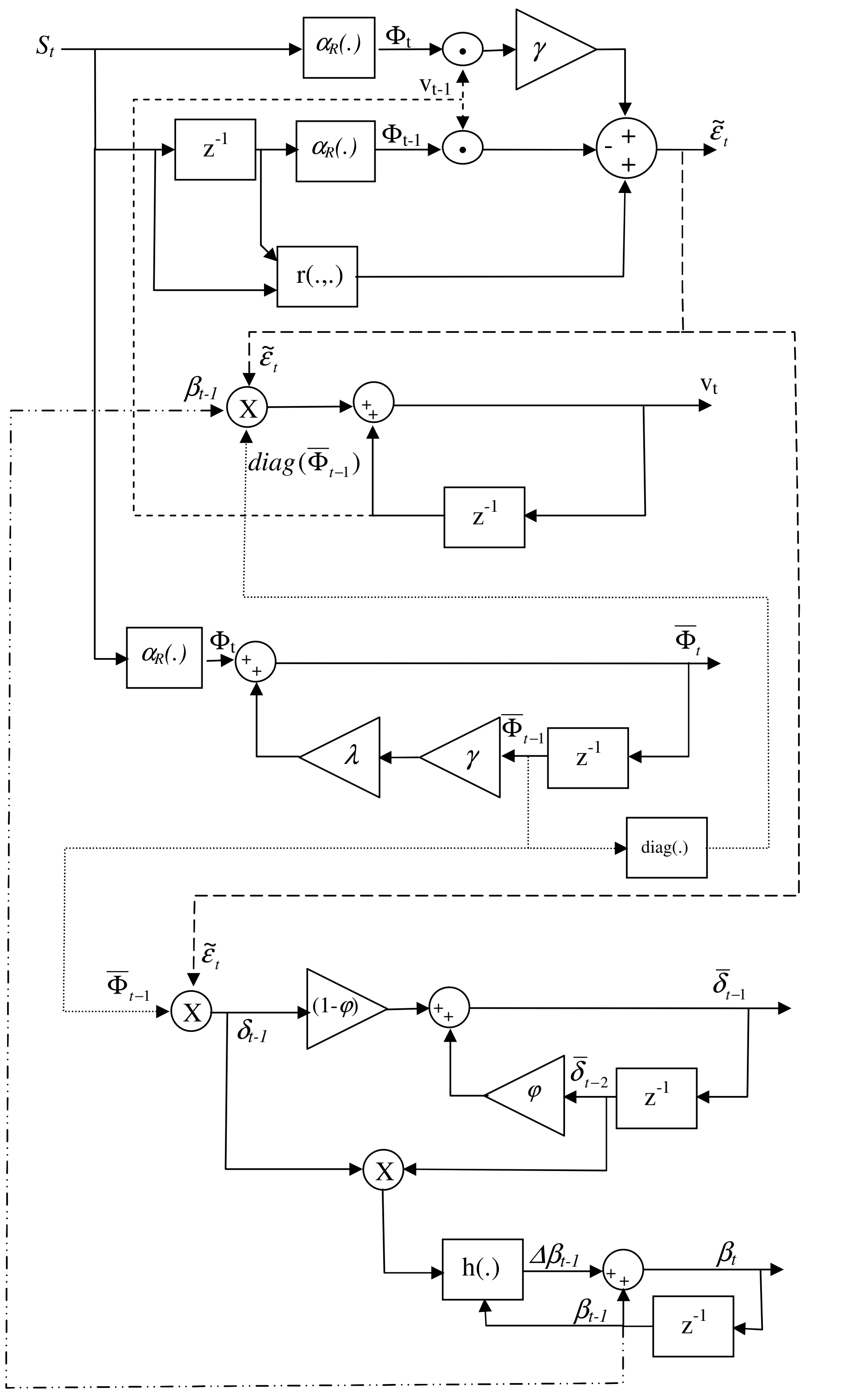


Figure 4.4.4 – The dynamics of value function estimate, i.e. the critic, assuming that the states are independent inputs to the system.

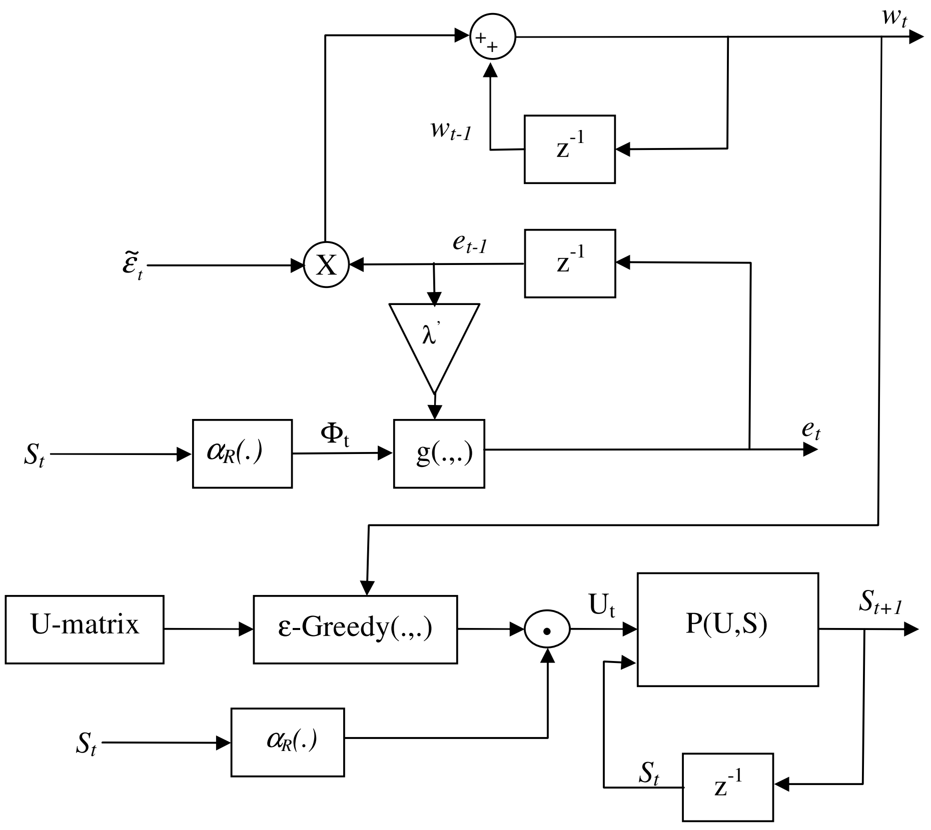


Figure 4.4.5 – The dynamics of the actor part and the plant. Assuming that the temporal difference error estimate is an input to the system.

The discrete-time dynamics of the actor part of the reinforcement learning module is described schematically in Figure 4.4.5. As seen in Figures 4.4.4 and 4.4.5, the dynamics of the actor and the critic are coupled over the temporal difference error, $\tilde{\varepsilon}$, which is determined by the critic and used as an input in the actor dynamics; and the state of the system, *S*, which is indirectly determined by the actor and used as an input in the critic dynamics.

We can infer from the above equations and graphs that we have a non-linear discrete-time dynamical system. If the system were linear, we would be able to find transfer functions (in the z-domain) from any input to any output and make usual sensitivity analysis using those transfer functions. In our case, however, we cannot perform direct sensitivity analysis; instead, we use partial derivatives as measures of parameter sensitivity.

Starting with the temporal difference error, $\tilde{\varepsilon}_t$:

$$\frac{\partial \tilde{\varepsilon}_t}{\partial \alpha_{R_i}(S_t)} = \gamma \cdot v_{t-1}^i \qquad (4.14)$$

$$\frac{\partial \tilde{\varepsilon}_t}{\partial \alpha_{R_i}(S_{t-1})} = -v_{t-1}^i \qquad (4.15)$$

$$\frac{\partial \tilde{\varepsilon}_t}{\partial r_t} = 1 \qquad (4.16)$$

$$\frac{\partial \tilde{\varepsilon}_t}{\partial v_{t-1}^i} = \begin{cases} \gamma \cdot \alpha_{R_i}(S_t) & \text{, if } R_i \in A_t \text{, and } R_i \notin A_{t-1} \\ -\alpha_{R_i}(S_{t-1}) & \text{, if } R_i \in A_{t-1} \text{, and } R_i \notin A_t \\ \gamma \cdot \alpha_{R_i}(S_t) - \alpha_{R_i}(S_{t-1}) & \text{, if } R_i \in A_{t-1} \text{, and } R_i \in A_t \end{cases} \qquad (4.17)$$

$$\frac{\partial \tilde{\varepsilon}_t}{\partial \gamma} = \sum_{R_i \in A_t} v_{t-1}^i \cdot \alpha_{R_i}(S_t) \qquad (4.18)$$

It is seen in equations (4.14) and (4.15) that the local value estimate of a rule plays the same role in both current rule activation and past rule activation sensitivity of the temporal difference error. This is expected, since for policy update the same value function estimate must be used for both the current state and one past state (one cannot update without knowing the performance of the current value estimate). Note also that, the discount factor proportionally increases the sensitivity of the temporal difference error to the activation of a rule for the current state (equation (4.15)). Equation (4.16) shows the importance of the reward function in the temporal difference error estimate, as the sensitivity is directly *1*, regardless of any parameters. The sensitivity of the temporal difference error to the individual value estimate is a piecewise function, which is dependent on how the related rule is participated in the past and current actions (equation (4.17)). The sensitivity of the temporal difference error to the reinforcement learning discount factor is the (total) value estimate of the current state as given in equation (4.18).

For the value function component of rule i, R_i, v_t^i we have the following measures:

$$\frac{\partial v_t^i}{\partial v_{t-1}^i} = 1 \qquad (4.19)$$

$$\frac{\partial v_t^i}{\partial \beta_{t-1}^i} = \tilde{\varepsilon}_t \cdot \overline{\Phi}_{t-1}^i \qquad (4.20)$$

$$\frac{\partial v_t^i}{\partial \tilde{\varepsilon}_t} = \beta_{t-1}^i \cdot \overline{\Phi}_{t-1}^i \qquad (4.21)$$

$$\frac{\partial v_t^i}{\partial \overline{\Phi}_{t-1}^i} = \beta_{t-1}^i \cdot \tilde{\varepsilon}_t \qquad (4.22)$$

As given in equation (4.19), the current value estimate is directly dependent on the one past value estimate; this is because the current estimate is built on the past estimate. Learning rate sensitivity of the value estimate increases with increasing temporal difference error and increasing eligibility trace and vice versa (equations (4.20) to (4.22)). This is since the updates in value function estimates are performed using multiplication of these 3 values as given in equation (4.10). Note that eligibility trace entry and the learning rate have always positive values; hence the current temporal difference error determines the update direction of the value estimate and the signs of sensitivities given in equations (4.20) and (4.22).

For the eligibility trace entry of value function component of *rule-i*, R_i , $\overline{\Phi}_t^i$ our sensitivity measure approach gives the following relations:

$$\frac{\partial \overline{\Phi}_t^i}{\partial \overline{\Phi}_{t-1}^i} = \gamma \cdot \lambda \qquad (4.23)$$

$$\frac{\partial \overline{\Phi}_t^i}{\partial \gamma} = \lambda \cdot \overline{\Phi}_{t-1} \qquad (4.24)$$

$$\frac{\partial \overline{\Phi}_t^i}{\partial \lambda} = \gamma \cdot \overline{\Phi}_{t-1} \qquad (4.25)$$

$$\frac{\partial \overline{\Phi}_t^i}{\partial \Phi_t^i} = 1 \qquad (4.26)$$

The direct relation given in equation (4.26) results from the fact that we are using accumulating eligibility traces (not replacing ones). In equation (4.23) it is seen that the sensitivity of the current value of the eligibility trace entry to the one past eligibility trace entry is proportional to both γ and λ values, since the multiplication of these two constants are used as the discount factor of the eligibility trace. Note from equations (4.24) and (4.25) that the sensitivity of eligibility trace entry to γincreases with λ, and its sensitivity to λ increases with γ.

The sensitivity measures for the $\overline{\delta}$ parameter are given as follows:

$$\frac{\partial \overline{\delta}_{t-1}^i}{\partial \overline{\delta}_{t-2}^i} = \varphi \qquad (4.27)$$

$$\frac{\partial \overline{\delta}_{t-1}^i}{\partial \varphi} = \overline{\delta}_{t-2}^i - \tilde{\varepsilon}_t \cdot \overline{\Phi}_{t-1}^i \qquad (4.28)$$

$$\frac{\partial \overline{\delta}_{t-1}^i}{\partial \tilde{\varepsilon}_t} = (1-\varphi) \cdot \overline{\Phi}_{t-1}^i \qquad (4.29)$$

$$\frac{\partial \overline{\delta}_{t-1}^i}{\partial \overline{\Phi}_{t-1}^i} = (1-\varphi) \cdot \tilde{\varepsilon}_t \qquad (4.30)$$

The updates to the learning rate are performed by inspecting the change in temporal difference error. $\delta$is used to incorporate the eligibility trace in the learning rate updates and $\bar{\delta}$ is employed in order to suppress the effect of instantaneous fluctuations in δ value (i.e., increasing momentum). Hence, these are ad hoc selected parameters rather than important contributors of the dynamics.

For the dynamics of the value function learning rates' sensitivity measures, we have the following equations:

$$\frac{\partial \beta_t^i}{\partial \beta_{t-1}^i} = 1 \qquad (4.31)$$

$$\frac{\partial \beta_t^i}{\partial \Delta \beta_{t-1}^i} = 1 \qquad (4.32)$$

$$\frac{\partial \Delta \beta_{t-1}^i}{\partial \beta_{t-1}^i} = \begin{cases} 0 & \text{,if } \bar{\delta}_{t-2}^i \cdot \delta_{t-1}^i > 0 \\ -\sigma & \text{,if } \bar{\delta}_{t-2}^i \cdot \delta_{t-1}^i < 0 \\ 0 & \text{,otherwise} \end{cases} \qquad (4.33)$$

$$\frac{\partial \Delta \beta_{t-1}^i}{\partial \sigma} = \begin{cases} 0 & \text{,if } \bar{\delta}_{t-2}^i \cdot \delta_{t-1}^i > 0 \\ -\beta_{t-1}^i & \text{,if } \bar{\delta}_{t-2}^i \cdot \delta_{t-1}^i < 0 \\ 0 & \text{,otherwise} \end{cases} \qquad (4.34)$$

$$\frac{\partial \Delta \beta_{t-1}^i}{\partial \kappa} = \begin{cases} 1 & \text{,if } \bar{\delta}_{t-2}^i \cdot \delta_{t-1}^i > 0 \\ 0 & \text{,if } \bar{\delta}_{t-2}^i \cdot \delta_{t-1}^i < 0 \\ 0 & \text{,otherwise} \end{cases} \qquad (4.35)$$

The equations (4.31) and (4.32) declares that the current value of the critic learning rate is directly dependent on its past value, and change in it is past value used for update; which is straightforward to expect. In equations (4.33) to (4.35) it is seen that the sensitivity of the change in critic learning rate is a piecewise function of the $\delta \cdot \overline{\delta}$ product for all the related parameters. Note also that, when this product is 0 there is no update in the learning rate. If this product is positive the update is done using κ parameter; resulting in the direct sensitivity given in equation (4.35). If the $\delta \cdot \overline{\delta}$ product is negative, the update is performed by decreasing the learning rate by a fraction (determined by σ) of the past value of the learning rate. Hence in this case, the sensitivity of the learning rate update to σ is proportional to the past value of the learning rate, and vice versa.

The above partial derivates given in equation (4.14) to equation (4.35) give measures about the sensitivity of critic's dynamics to various parameters.

For the actor's sensitivity analysis, we should first note a few points: First of all the system dynamics is dependent on which controller we are concerned about and we are not interested in the sensitivity of the controller to changes in the system. Moreover, the *ε-Greedy* action selection algorithm is used for a suitable searching of the state-space and it is not the concern of the controller at all. Any other approach in order to search the state space may be incorporated with the algorithm. Therefore, we will only investigate the sensitivity of the rule-action weights and the related eligibility trace.

For an individual rule-action weight (action *j* of rule *i*):

$$\frac{\partial w_t^{ij}}{\partial w_{t-1}^{ij}} = 1 \qquad (4.36)$$

$$\frac{\partial w_t^{ij}}{\partial \tilde{\varepsilon}_t} = e_{t-1}^{ij} \qquad (4.37)$$

$$\frac{\partial w_t^{ij}}{\partial e_{t-1}^{ij}} = \tilde{\varepsilon}_t \qquad (4.38)$$

Note again the direct sensitivity of the rule-action weight's current value to its past value given in equation (4.36), due to usual updates. In equations (4..37) and (4.38) it is seen that the sensitivity of the rule-action weight to the temporal difference error is proportional with the eligibility trace entry of that rule-action; and vice versa. Hence as the eligibility trace entry increases the weight is more sensitive to the temporal difference error; and as the temporal difference error increases the weight is more sensitive to the eligibility trace entry.

For the related eligibility trace index, we have:

$$\frac{\partial e_t^{ij}}{\partial e_{t-1}^{ij}} = \lambda' \qquad (4.39)$$

$$\frac{\partial e_t^{ij}}{\partial \lambda'} = e_{t-1}^{ij} \qquad (4.40)$$

$$\frac{\partial e_t^{ij}}{\partial \Phi_t^i} = \begin{cases} 1 & \text{, if } U^{ij} = U_t^i \\ 0 & \text{, otherwise} \end{cases} \qquad (4.41)$$

Sensitivity of the current value of the related eligibility entry to its past value is measured with the actor recency factor, λ'; whereas its sensitivity to the recency factor is measured with the past value of the eligibility trace entry (equations (4.39) and (4.40)). The eligibility trace entry has directly sensitive to the activation value of the related rule ($\Phi_t^i = \alpha_{R_i}(S_t)$), if the action offered by that rule is the one corresponding to that eligibility trace entry (equation (4.41)).

Taking the above ideas as a base, and using equations from (4.14) to (4.41) we perform an instance of the reinforcement learning module sensitivity analysis for the head target reaching angle controller as follows:

We consider the training of the controller from the beginning and create a target at position (10,10) in a 20x20 environment. We prefer this symmetry in order to homogeneously distribute the random initial states in the state space. It is worth to restate that (see Tables 3.4.1 and 3.4.2 above) we have 7 fuzzy sets for the α variable, and 4 fuzzy sets for the *z* variable, resulting in 28 rules. For the actor, we have the same 5-element action set for each rule, which gives 28x5 = 140 individual rule-action weights. We investigate the controller dynamics in the 40th episode of the training:

The 40th episode, in our sample simulation, starts with (α,z) = (-0.75088,19.570) at time step $t_0 = 0$, and finishes with $(\alpha,z) = (-0.16311,19.754)$ at time step $t_{\text{end}} = 19$. We take the middle portion of this episode end consider the times *t-2=8, t-1=9* and *t=10.* Starting with the temporal difference error, calculated at time step 9, $\tilde{\varepsilon}_{10}$, using equation (2.26), we get:

$$(\alpha_{t-1},z_{t-1}) = (-0.36102,19.773),\ (\alpha_t,z_t) =(-0.33347,19.770)$$

$$A_{t-1} = \{R_{12},R_{16}\}$$

$$A_t = \{R_{12},R_{16}\}$$

$$\alpha_{R_{12}}(S_{t-1}) = 0.3448, \alpha_{R_{16}}(S_{t-1}) = 0.6552$$

$$\alpha_{R_{12}}(S_t) = 0.3184, \alpha_{R_{16}}(S_t) = 0.6816$$

$$\tilde{\varepsilon}_t = -0.0385$$

$$v_{t-1}^{12} = 0.3054, v_{t-1}^{16} = 0.4945$$

$$r_t = 0$$

substituting the above values in the equations (4.14) to (4.18):

$$\frac{\partial \tilde{\varepsilon}_t}{\partial \alpha_{R_{12}}(S_t)} = 0.2749\,, \quad \frac{\partial \tilde{\varepsilon}_t}{\partial \alpha_{R_{16}}(S_t)} = 0.4451$$

$$\frac{\partial \tilde{\varepsilon}_t}{\partial \alpha_{R_{12}}(S_{t-1})} = -0.3054\,, \quad \frac{\partial \tilde{\varepsilon}_t}{\partial \alpha_{R_{16}}(S_{t-1})} = -0.4945$$

$$\frac{\partial \tilde{\varepsilon}_t}{\partial r_t} = 1$$

$$\frac{\partial \tilde{\varepsilon}_t}{\partial v_{t-1}^{12}} = \text{-}0.0582, \quad \frac{\partial \tilde{\varepsilon}_t}{\partial v_{t-1}^{16}} = \text{-}0.0418$$

$$\frac{\partial \tilde{\varepsilon}_t}{\partial \gamma} = 0.4343$$

It is seen that the temporal difference error is more sensitive to the changes in the activation value of *rule-16* than those in activation value of *rule-12.* This arises from the fact that the current local value estimate of *rule-16* is bigger than that of *rule-12*. For the sensitivity of the temporal difference error to local value function estimates of each rule, we see that both rules are activated at time steps *t-1* and *t.* Hence the 3rd line of the piecewise function given in (4.17) is used for the calculations. This means that, these sensitivities are affected by the reinforcement learning discount factor, current value of the rule activation value, and the past value of the rule activation value. For the sensitivity of the temporal difference error to the discount factor; we have the value estimate of the current state, calculated with the value function of one passed time; which is found to be 0.4343 numerically. Hence, for our case temporal difference error is mostly sensitive to reward, past and current activation values of the rules and γ. Sensitivity of temporal difference error to the changes in local value function estimates is smaller by a factor of 10.

Continuing with the local value estimate of each rule: We should first note that, in the implementation, we use eligibility traces of length 20 for all FACL controllers. Hence, we have a maximum of 20 value estimates updated in a learning sampling time. Moreover, in our case, only entries of related rules are non-zero; due to the small recency factor (0.1) used. First let's give the necessary variable's numerical values:

$$\overline{\Phi}_{t-1}^{12} = 0.4135,\ \overline{\Phi}_{t-1}^{16} = 0.6854$$
$$\beta_{t-1}^{12} = 0.00005823,\ \beta_{t-1}^{16} = 0.00004874$$

Using these values in the equations (4.19) to (4.22), we have the following numerical results:

$$\frac{\partial v_t^{12}}{\partial v_{t-1}^{12}} = 1,\ \frac{\partial v_t^{16}}{\partial v_{t-1}^{16}} = 1$$
$$\frac{\partial v_t^{12}}{\partial \beta_{t-1}^{12}} = -0.0159,\ \frac{\partial v_t^{16}}{\partial \beta_{t-1}^{16}} = -0.0264$$
$$\frac{\partial v_t^{12}}{\partial \tilde{\varepsilon}_t} = 2.4078\times 10^{-5},\ \frac{\partial v_t^{16}}{\partial \tilde{\varepsilon}_t} = 3.3406\times 10^{-5}$$
$$\frac{\partial v_t^{12}}{\partial \overline{\Phi}_{t-1}^{12}} = -2.2419\times 10^{-6},\ \frac{\partial v_t^{16}}{\partial \overline{\Phi}_{t-1}^{16}} = -1.8765\times 10^{-6}$$

From these equations, it is clear that for our case study, the local value estimates of rules are mostly sensitive to the related learning rates (apart from the direct sensitivity to their past values). This explains why the initial learning rate selection deeply affects the performance of learning as given in tables (4.2.1) to (4.2.7).

For the eligibility trace indices of the activated rules, we have the following sensitivity measures, which are directly calculated using equations (4.23) to (4.26) and the given parameter values as:

$$\frac{\partial \overline{\Phi}_t^{12}}{\partial \overline{\Phi}_{t-1}^{12}} = 0.09, \ \frac{\partial \overline{\Phi}_t^{16}}{\partial \overline{\Phi}_{t-1}^{16}} = 0.09$$

$$\frac{\partial \overline{\Phi}_t^{12}}{\partial \gamma} = 0.04135, \ \frac{\partial \overline{\Phi}_t^{16}}{\partial \gamma} = 0.06854$$

$$\frac{\partial \overline{\Phi}_t^{12}}{\partial \lambda} = 0.3721, \ \frac{\partial \overline{\Phi}_t^{16}}{\partial \lambda} = 0.6169$$

$$\frac{\partial \overline{\Phi}_t^{12}}{\partial \Phi_{t-1}^{12}} = 1, \ \frac{\partial \overline{\Phi}_t^{16}}{\partial \Phi_{t-1}^{16}} = 1$$

Hence the eligibility traces are mostly sensitive to the eligibility trace recency factor, which gives insight about the effects of this parameter whenever eligibility trace usage is an important factor of learning (i.e., the controller duty is complicated enough).

For the learning rate, and learning rate update values we have:

$$\overline{\delta}_{t-2}^{12} = -0.0164, \ \overline{\delta}_{t-2}^{16} = -0.0238$$

$$\overline{\delta}_{t-1}^{12} = -0.0163, \ \overline{\delta}_{t-1}^{16} = -0.0259$$

$$\delta_{t-1}^{12} = -0.0159, \ \delta_{t-1}^{16} = -0.0264$$

We can infer from these values that the temporal difference error is negative for a few recent steps; hence the value function estimate moves in the same direction. This implies that the learning rates of the critic should be increased.

Using equations (4.27) to (4.35), we have:

$$\frac{\partial \overline{\delta}_{t-1}^{12}}{\partial \overline{\delta}_{t-2}^{12}} = 0.2, \ \frac{\partial \overline{\delta}_{t-1}^{16}}{\partial \overline{\delta}_{t-2}^{16}} = 0.2$$

$$\frac{\partial \overline{\delta}_{t-1}^{12}}{\partial \varphi} = -4.8025 \times 10^{-4}, \ \frac{\partial \overline{\delta}_{t-1}^{16}}{\partial \varphi} = 0.0026$$

$$\frac{\partial \bar{\delta}_{t-1}^{12}}{\partial \tilde{\varepsilon}_t} = 0.3308\,,\ \frac{\partial \bar{\delta}_{t-1}^{16}}{\partial \tilde{\varepsilon}_t} = 0.5483$$

$$\frac{\partial \bar{\delta}_{t-1}^{12}}{\partial \bar{\Phi}_{t-1}^{12}} = -0.0308\,,\ \frac{\partial \bar{\delta}_{t-1}^{16}}{\partial \bar{\Phi}_{t-1}^{16}} = -0.0308$$

$$\frac{\partial \beta_t^{12}}{\partial \beta_{t-1}^{12}} = 1\,,\frac{\partial \beta_t^{16}}{\partial \beta_{t-1}^{16}} = 1$$

$$\frac{\partial \beta_t^{12}}{\partial \Delta\beta_{t-1}^{12}} = 1\,,\frac{\partial \beta_t^{16}}{\partial \Delta\beta_{t-1}^{16}} = 1$$

$$\frac{\partial \Delta\beta_{t-1}^{12}}{\partial \beta_{t-1}^{12}} = 0\,,\frac{\partial \Delta\beta_{t-1}^{16}}{\partial \beta_{t-1}^{16}} = 0$$

$$\frac{\partial \Delta\beta_{t-1}^{12}}{\partial \sigma} = 0\,,\frac{\partial \Delta\beta_{t-1}^{16}}{\partial \sigma} = 0$$

$$\frac{\partial \Delta\beta_{t-1}^{12}}{\partial \kappa} = 1\,,\frac{\partial \Delta\beta_{t-1}^{16}}{\partial \kappa} = 1$$

Note that the sensitivity of $\bar{\delta}$ to the temporal difference error is greater than its sensitivity to the eligibility trace entries. Hence, we can get the idea that for this controller, eligibility trace usage may not increase the performance significantly.

The above findings may be used to gain insight about the dynamics of the critic part of the FACL for the head target reaching angle controller. Moreover, similar analysis steps may be repeated for other controllers, possibly with more computational effort, but basically with the same approach.

For the actor, we continue the analysis using the same simulation made for the critic analysis, and analyze the actor part. We have 7 possible actions for each of the rules. We again use an eligibility trace of length 20. Hence, only 20 dominating rule-action pairs exist in the eligibility trace. In our case, since we fix λ' to the value of 0.1, and the eligibility trace erasure threshold at 0.00001, erasure of an eligibility trace index is easy (just in 5-6 steps after, if not repeated). As the result of this fact, we see that, only two rule/actions have non-zero eligibility trace entries which are action 7 (ω = π/4) of rule 12, and action 4 (ω = 0) of rule 16. If we use these values in equations (4.36) to (4.41) with values $e_{t-1}^{12,7} = 0.4186$, $e_{t-1}^{16,4} = 0.6926$:

$$\frac{\partial w_t^{12,7}}{\partial w_{t-1}^{12,7}} = 1, \ \frac{\partial w_t^{16,4}}{\partial w_{t-1}^{16,4}} = 1$$

$$\frac{\partial w_t^{12,7}}{\partial \tilde{\varepsilon}_t} = 0.4186, \ \frac{\partial w_t^{16,4}}{\partial \tilde{\varepsilon}_t} = 0.6926$$

$$\frac{\partial w_t^{12,7}}{\partial e_{t-1}^{12,7}} = -0.0385, \ \frac{\partial w_t^{16,4}}{\partial e_{t-1}^{16,4}} = -0.0385$$

$$\frac{\partial e_t^{12,7}}{\partial e_{t-1}^{12,7}} = 0.1, \ \frac{\partial e_t^{16,4}}{\partial e_{t-1}^{16,4}} = 0.1$$

$$\frac{\partial e_t^{12,7}}{\partial \lambda'} = 0.4186, \ \frac{\partial e_t^{16,4}}{\partial \lambda'} = 0.6926$$

$$\frac{\partial e_t^{12,7}}{\partial \Phi_t^{12}} = 1, \ \frac{\partial e_t^{16,4}}{\partial \Phi_t^{12}} = 1$$

Again, note that the sensitivity of action weights to the temporal difference error is much more dominant than the sensitivity to the eligibility trace entries; which verifies our idea of using eligibility trace for this simple controller.

CHAPTER 5

CONCLUSION

5.1 Summary and Conclusive Remarks

In this study a high-level controller architecture for complex hyper-redundant robots is designed, developed and simulated. Since the system worked on is highly non-linear, and the environment in which the system runs is complex, techniques dealing with such problems are used. These techniques include fuzzy inference systems that use reinforcement learning algorithms for tuning their defuzzifier parameters in the high level, and a genetic algorithm implementation that gives flexibility of optimizing the motion dynamics with respect to any desired parameter in the low level. The developed high-level controller architecture consists of several controllers that learn how to perform a specific control duty from interaction with the environment. The architecture is modular and it is suitable for parallel processing implementations. In the lower level, previously memorized gait types (which may be stated as handicaps of similar studies) are not used, and motion of robot links are not constrained by human knowledge. In addition to these, since the designed architecture is based on learning controllers, it may be extended to perform several tasks other than target reaching and obstacle avoidance. In order to verify this property, a grasping scheme is designed and implemented successfully for a 12-link robot.

The only study performed is not just implementation of the algorithm; instead upon implementation of the developed controller and verification via simulations, some mathematical analysis is performed on the system in order to gain insight of how the algorithm works, and which parameters dominate in the performance of successful learning

5.2 Future Work

Mathworks' advanced computational environments MATLAB and Simulink are used in the development and simulation of the controller. This approach gives the advantage of motivating on the ideas rather than computer programming. The handicap is that since the MATLAB codes are run in an interpreter, the time needed for simulation runs are longer than a C or similar compiler based language implementation. Hence, the MATLAB codes may be converted to a compiler-based language in a future study.

For the sensitivity analysis performed in this study, since the closed form input-output relations may not be obtained for most of the cases due to complex dynamics, simplifications have to be done. The performed analysis may be extended by using different approaches that avoid these simplifications, possibly dealing with more complicated mathematics.

As stated before, genetic algorithms technique is used to decide on joint angles to perform a desired displacement by optimizing a fitness function that includes both positional error and total energy consumption. It is known that genetic algorithms work well off-line; but several precautions must be taken in order to make them work on-line in real-time. Other techniques whose real-time implementations are easy, but possibly handicapped in other issues, might be used instead of our approach. It should also be noted that the lower level controller implementation is not the main focus of this study; hence in order to be able to apply our algorithms to a physical robot the low level control problem should be solved beforehand, which may also be focus of a future study.

Another possible future work might be the realization of the developed algorithm in an embedded control system and application to a physical robot. However, the developed controller implementations are computationally heavy in both higher-level learning and lower-level dynamics simulation and genetic algorithm control. Use of MATLAB further makes the implementations slower on a regular PC, which works using sequential computation steps. However, in a real-world implementation, use of parallel processing abilities on the silicon level (devices similar to FPGAs and DSPs) should speed up the computations and allow a real-time run with some modifications in the algorithm. In the design phase of the algorithm, this issue is considered and the controller architecture is suitable to be computationally distributed to different processors placed on different robot links. Moreover, the low-level control of joint angle actuators are neglected and assumed to be ideal during simulations, which might be a problem to be solved using several possible techniques in a real-world application. Hence, all these issues should be considered before a real-world application of this study.

REFERENCES

[1] Murphy, Robin R., "Activities of the Rescue Robots at the World Trade Center from 11-21 September 2001", IEEE Robotics and Automation Magazine, September 2004, pp.50-61.

[2] Saito M., Fukaya M., and Iwasaki T., "Serpentine Locomotion with Robotic Snakes", IEEE Control Systems Magazine, February 2002, pp. 64-81.

[3] Gevher, M., "Sensor Based On-line Path Planning For Serpentine Robots," M.Sc. Thesis, Middle East Technical University, Ankara, Dec. 2001.

[4] Erkmen I., Erkmen A.M., Matsuno F., Chatterjee R., Kamegawa T., "Snake Robots to the Rescue!", IEEE Robotics and Automation Magazine, September 2002, pp. 17-25.

[5] Ma S., et. al. "Analysis of Creeping Locomotion of a Snake Robot on a Slope", Proc. Of the IEEE International Conference on Robotics and Automation, 2003, pp.2073-2078.

[6] Cai Z., "Intelligent Control: Principles, Techniques and Applications", World Scientific, 1997.

[7] Chirikjian G. S., Burdick J.W., "The Kinematics of Hyper-Redundant Robot Locomotion", IEEE Transactions on Robotics and Automation", Vol. 11, No. 6, December 1995, pp. 781-788.

[8] Chirikjian G. S., Burdick J.W., "A Modal Approach to Hyper-Redundant Manipulator Kinematics", IEEE Transactions on Robotics and Automation", Vol. b10 No. 3, June 1994, pp. 343-354.

[9] Nilsson M., "Snake Robot Free Climbing", IEEE Control Systems, February 1998, pp.21-26.

[10] Shan Y., Koren Y., “Design and Motion Planning of a Mechanical Snake”, IEEE Transactions on Systems, Man and Cybernetics, Vol. 23, No. 4, July / August 1993, pp. 794-805.

[11] Mori M., Hirose S., “Three-Dimensional Serpentine Motion and Lateral Rolling by Active Cord Mechanism ACM-R3”, Proceesings of the 2002 IEEE / RSJ International Conference on Intelligent Robots and Systems, Lausanne, Switzerland, October 2002, pp. 829-834.

[12] NEC Corporation, “Orochi 12 DOF Snake-Like Robot”, Press Release, NEC Corporation, Melvilee, NY., January 1996, 6 pages

[13] Matsuno F., Suegana K., “Control of Redundant 3D Robot Based on Kinematic Model”, Proceedings of The 2003 IEEE International Conference on Robotics & Automation, Taipei, Taiwan, September 14-19 2003, pp. 2061-2066.

[14] Prautsch P., Mita T., “Control and Analysis of the Gait of Snake Robots”, Proceedings of the 1999 IEEE International Conference on Control Applications, Kohala Coast-Island of Hawai’i, Hawai’i, USA, August 22-27,1999, pp. 502-507.

[15] Kamegawa T., Matsuno F., Chatterjee R., “Proposition of Twisting Mode of Locomotion and GA-based Motion Planning for Transition of Locomotion Modes of 3-Dimensonal Snake-like Robot”, Proceedings of the 2002 IEEE International Conference on Robotics & Automation, Washington DC, May 2002, pp. 1507-1512.

[16] Chirikjian G.S., Burdick J.W., “Kinematics of Hyper-Redundant Locomotion with Applications to Grasping”, 1991 IEEE Conference on Robotics and Automation, April 1991, pp. 567-574.

[17] Ma S., Li W.J., Wang Y., “A Simulator to Analyze Creeping Locomotion of a Snake-like Robot”, Proceedings of the 2001 IEEE International Conference on Robotics & Automation, Seoul, Korea, May 21-26, 2001, pp. 3656-3661.

[18] Ma S., “Analysis of Snake Movement Forms for Realization of Snake-like Robots”, Proceedings of the 1999 IEEE International Conference on Robotics & Automation, Detroit, Michigan, May 1999, pp. 3007-3013.

[19] Kulali, G. M., “Intelligent Gait Synthesizer for Serpentine Robots” M.Sc. Thesis, Middle East Technical University, Ankara, Dec. 2001.

[20] Halici, Ugur, “Introduction to Theory of Neural Networks”, EE-543 Lecture Notes, Middle East Technical University, Department of Electrical and Electronics Engineering, 2001.

[21] Haykin, S., “Neural Networks: A Comprehensive Foundation”, McMillan Inc., 1994.

[22] Sutton R.S., Barto A.G., “Reinforcement Learning: An Introduction”, The MIT Press, Cambridge, Massachusetts, 1999.

[23] Jouffe L., “Fuzzy Inference System Learning by Reinforcement Methods”, IEEE Transactions on System, Man and Cybernetics – Part C: Applications and Reviews, Vol. 28, No. 3, August 1998, pp. 338-355.

[24] Barto A.G., Sutton R.S., Anderson C.W., “Neuron-like Adaptive Elements That Can Solve Difficult Learning Control Problems”, IEEE Transactions on Systems, Man and Cybernetics, Vol. SMC-13, Sept. 1983, pp. 834-846.

[25] Jang J.R., “Self-Learning Fuzzy Controllers Based on Temporal Back Propagation”, IEEE Transactions on Neural Networks, Vol. 3, No. 5, September 1992, pp. 714-723.

[26] Berenji H.R., Khedkar P., “Learning and Tuning Fuzzy Logic Controllers Through Reinforcements”, IEEE Transactions on Neural Networks, Vol. 3, No. 5, September 1992, pp. 724-740.

[27] Jacobs R.A., “Increased Rates of Convergence Through Learning Rate Adaptation”, Neural Networks, vol. 1, 1988, pp. 295-307.

[28] Ari E.O., Erkmen I., Erkmen A.M., "An FACL Controller Architecture for a Grasping Snake Robot," in Proc. of IEEE International Conference on Intelligent Robots and Systems (IROS), August 2005, pp. 3339-3344.

[29] Beom, H.R., Cho H.S., "A Sensor-Based Navigation for a Mobile Robot Using Fuzzy Logic and Reinforcement Learning", IEEE Transactions on Systems, Man and Cybernetics, Vol. 25, No. 3, pp. 464-477, March 1995.

[30] Dowling K., "Limbless Locomotion: Learning to Crawl", in Proceedings of IEEE International Conference on Robotics and Automation, Detroit, Michigan, May 1999, pp. 3001-3006.

[31] Liu H., Yan G. and Ding G., "Research on the Locomotion Mechanism of Snake-like Robots," in Proceedings of IEEE Int. Symposium on Micro-mechatronics and Human Science, pp. 183-188, 2001.

[32] Murphy R. R., "Marsupial and Shape-shifting Robots for Urban Search and Rescue", IEEE Intelligent Systems, March/April 2000, pp. 14-19.

[33] Shammas E., Wolf A., Brown H.B., Choset H., "New Joint Design for Three-dimensional Hyper Redundant Robots", Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, Las Vegas, Nevada, October 2003, pp. 3594-3599.

[34] Brown H.B., Weghe J.M.V., Bererton C.A., Khosla P.K., "Milibot Trains for Enhanced Mobility", IEEE/ASME Transactions on Mechatronics, Vol. 7, No. 4, December 2002, pp. 452-461.

[35] Yim M., Zhang Y., Roufas K., Duff D., Eldershaw C., "Connecting and Disconnecting for Chain Self-Reconfiguration With PolyBot", IEEE/ASME Transactions on Mechatronics, Vol. 7, No. 4, December 2002, pp. 442-451.

[36] Wolf A., Brown H.B., Casciola R., Costa A., Schwerin M., Shamas E., Choset H., "A Mobile Hyper Redundant Mechanism for Search and Rescue Tasks", Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, Las Vegas, Nevada, October 2003, pp. 2889-2895

[37] Ma S., Ohmameuda Y., Inoue K., Li B., "Control of a 3-Dimensional Snake-like Robot", Proceedings of the 2003 IEEE International Conference on Robotics and Automation, Taipei, Taiwan, September 14-19, 2003, pp. 2067-2072.

[38] Matsuno F., Suegana K., "Control of Redundant 3D Snake Robot based on Kinematic Model", Proceedings of the 2003 IEEE International Conference on Robotics and Automation, Taipei, Taiwan, September 14-19, 2003, pp. 2061-2066.

[39] Chirikjian G.S., Burdick J.W., "Kinematically Optimal Hyper-Redundant Manipulator Configurations", IEEE Transactions on Robotics and Automation, Vol. 11, No. 6, December 1995, pp. 794-806.

[40] Linnemann R., Paap K.L., Klaassen B., Vollmer J., "Motion Control of a Snakelike Robot", 1999 IEEE International Conference on Robotics and Automation.

[41] Dowling K., "Limbless Locomotion: Learning to Crawl with a Snake Robot", Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, 1997.

[42] Gravagne I.A., Walker I.D., "Manipulability, Force, and Compliance Analysis for Planar Continuum Manipulators", IEEE Transactions on Robotics and Automation, Vol. 18, No. 3, June 2002.

[43] Liu H., Yan G., Ding G., "Research on the Locomotion Mechanism of Snake-like Robot", IEEE International Symposium on Micromechatronics and Human Science, 2001, pp. 183-188.

[44] Burdick J.W., Radford J., "A "Sidewinding" Locomotion Gait for Hyper-Redundant Robots", IEEE 1993, pp. 101-106.

[45] Desai R., Rosenberg C.J., Jones J.L., "Kaa: An Autonomous Serpentine Robot Utilizes Behavior Control", International Conference on Intelligent Robots and Systems, IROS'95, Pittsburgh, PA, V.3 pp. 250-255, August 1995.

[46] Malachi D., Munerato F., "Snake-like Mobile Micro Robot Based on 3 DOF Parallel Mechanism", PKM'99, Milan, November 1999.

[47] Goldberg, D. E., “Genetic Algorithms in Search, Optimization and Machine Learning”, Addison-Wesley Publishing, 1989.

APPENDIX

PAPER PRESENTED IN IEEE INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS (IROS)-2005 AND PAPER SUBMITTED TO THE IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION (ICRA)-2006

Throughout this thesis study, two papers were submitted to international conferences. The paper related to the first steps of the study, i.e. the control of the planar robot, is submitted to the IROS-2005 conference, it was accepted and published. Near to the end of the study, a paper describing the final controller architecture that is focused on the grasping behavior is submitted to the ICRA-2006 conference. In this part of the thesis, these papers are given for the completeness of the work and for making these papers easy to reach for readers of the thesis.

2005 IEEE/RSJ International Conference on Intelligent Robots and Systems

An FACL Controller Architecture for a Grasping Snake Robot

Evrim Onur Ari
Microwave and System Tech. Div., EHD-CCC, ASELSAN Inc., Ankara, Turkey
oari@aselsan.com.tr

Ismet Erkmen and Aydan M. Erkmen
Middle East Technical University, Department of Electrical and Electronics Engineering, Ankara, Turkey
{erkmen, aydan}@metu.edu.tr

***Abstract* - In this study, a distributed intelligent fuzzy learning controller architecture is developed for a snake robot to avoid obstacles while reaching a target, in a dynamic environment. Moreover, in order to use the robot in object carrying search and rescue (SAR) applications, a grasping scheme of desired objects is added to the abilities of the controller. Behavior-based approach is also incorporated by "target reaching", "link following", "obstacle avoidance" and "object grasping" behaviors. Each link of the robot is separately controlled and able to select a behavior at any time. Behaviors are all realized as Fuzzy Actor Critic Learning (FACL) controllers [1]. In order to move the robot in between different configurations and take dynamics into account, an on-line optimization technique based on genetic algorithms is employed.**

Index Terms – snake robots, biologically inspired robots, search and rescue robotics, reinforcement learning, FACL.

I. Introduction

Recent experiences of natural disasters (earthquakes, tsunami etc.) and man made catastrophes (terrorism) have put more emphasis to the area of Search and Rescue (SAR) and emergency management. Tool developments for possible future SAR applications, including autonomy, high mobility, robustness and reconfigurability for terrain and task adaptation have been the recent focus. Despite having drawbacks, like their control difficulty, inefficiency in locomotion due to high friction etc. [2], snake-like robots attracted the attention of researchers for applications not suitable for wheeled and legged robots, such as ruins of collapsed buildings or narrow passages such as pipelines. Snake-like robots are advantageous over wheeled vehicles for terrainability, traction, universal penetration capabilities, high adaptability and task shapability due to kinematic redundancies, which offer graceful degradation, increasing reliability when made modular [3],[4],[5]. However, serpentine SAR devices still possess the disadvantage of payload, speed, complexity and uncertainty handling in motion gait planning and control due to high degrees of freedom [6],[7]. In our work, we focus on task adaptability on irregular terrain affecting the serpentine gaits for target searching despite obstacles, and also task shapability where snake redundancy is partially distributed to lasso-type grasping, dragging or lifting tasks; while the remaining portion undergoes serpentine gaits. Control complexity and uncertainty in gait planning and execution are overcome by a distributed fuzzy actor critic learning (FACL) controller architecture, first introduced by Jouffe [1], that we modify for our serpentine robot coupling it with an on-line optimization technique modified from Dowling's studies [8] for dynamic gait configuration changes. Moreover, in order to fully take the advantage of the hyper-redundant mechanism, we added a lasso-type grasping scheme to the abilities of our robot, making it superior than wheeled and legged robots in SAR applications.

II. Demonstrative Hyper-redundant Snake-like Robot Model

Currently we are pursuing the 3-D implementation of our integrated serpentine locomotion and grasping control architecture. However, in this paper we present our approach as an initial phase, on a 2-D (planar) snake model, as shown in Fig. 1.

Each link of the robot is assumed to be dynamically identical, with interchangeable extreme links defined as the "head link" and the "tail link". Dynamical analysis of such a mechanism, moving on a plane with friction has already been worked out by several researchers [2],[4],[7]. We implement the dynamics using MATLAB Simmechanics software package in our computer simulations, for which a snapshot is given in Fig. 2.

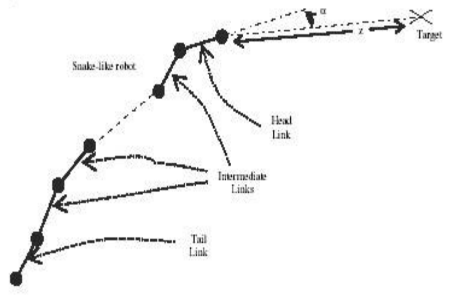


Fig. 1 The planar snake-like robot mechanism used.

Our serpentine robot is equipped with proximity sensors detecting surrounding obstacles on each side of the links; the head link is further equipped with additional proximity sensors for sensing obstacles in front of the robot. The sensors are assumed to be grouped in sensor suits (similar to [9]), each giving the distance of the closest obstacle for the side sensors; distance and angular position of the closest obstacle for the frontal sensors. There are two sensor suits for side sensing and two sensor suits for frontal sensing, as illustrated in Fig. 3, for the head segment; whereas, only two side sensor suits exist on

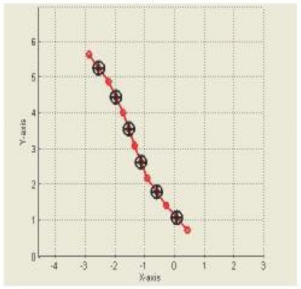


Fig. 2 A snapshot from the Simmechanics software visualization of a 6-link planar serpentine structure. CGs are shown with crosses, while joints are shown by hollow circles.

the following links. The robot is assumed to know the location of the target to be reached implicitly, by measuring α and z variables shown in Fig. 1. This assumption is necessary for error feedback to the controller and was also present in earlier similar studies [6],[7],[9].

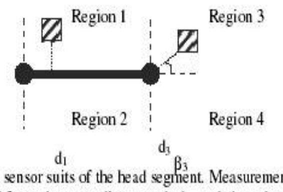


Fig. 3 Ultra-sound sensor suits of the head segment. Measurements of sensor suits 1 and 3 are shown as distances d_1,d_3, and obstacle sight angle β_3. Obstacles are shown as striped rectangles. Measurements for suits 2 and 4 are similar to those for 1 and 3 respectively.

III. The FACL Gait Controller Architecture

As stated before, there are several studies on snake-like robots; which include controller designs for directing the robot towards a target while avoiding collision with the obstacles in the environment [5],[7],[10]. In our study, we aimed to design a controller architecture, which has the following properties

i. Dealing with uncertainties, the non-linearity of the complex hyper-redundant structure, and the dynamic behavior of the environment through learning control.

ii. Being segment-modular, such that every link having the same mechanical structure is controlled individually in the highest control level, making the whole structure a robot network.

iii. Being control-modular, such that different duties of the controller are carried out by different processing units rendering the control architecture, a control network, thus yielding property iv.

iv. Allowing distributed control and parallel processing in the implementation; such that the computational burden of the complex algorithms is divided between the controllers of individual links, overlaying the robot network with the control network.

In order to achieve the above design goals, we divide the basic control aims first into the two classical behaviors of "target reaching" and "obstacle avoidance" [9]; then, extending them by the "object grasping" behavior for grasping desired objects during serpentine motion, when necessary. Having three control objectives for the robot, we further distribute these objectives throughout the robot network's individual links as separate controllers. Our approach yields the general controller architecture block diagram as shown in Fig. 4.

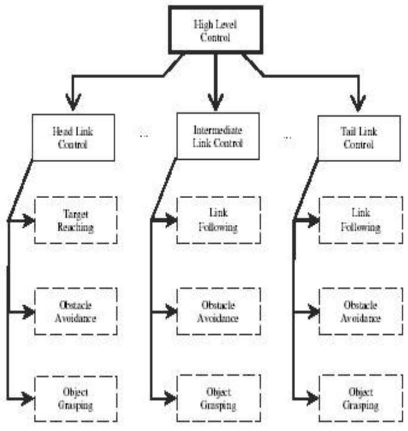


Fig. 4 The block diagram of the highest-level control architecture.

As the result, we have a segment modular and control modular high-level architecture, i.e. a robot network overlaid by a control network, where each segment applies an individual behavior for a given state. When target-reaching behavior is active, the head link tries to reach the target; as if its an individual robot, and other links try to follow the immediate preceding link. In the existence of nearby obstacles, a link may switch to obstacle avoidance behavior individually (a potential field approach similar to [9] is employed whose equations are given below, and other approaches may also be incorporated).

$$E_{rep} = \sum_{j=1}^{4} \frac{K_r a_{\max} \Delta d_j}{2a_{\max} d_j - (\Delta d_j)^2} \qquad E_{att} = \frac{\sqrt{2z^2 \cos^2 \alpha + 4a_{\max} z} - z\cos\alpha}{a_{\max}}$$

$$E = E_{att} + E_{rep} \qquad c_a = (\Delta E)_{avoidance} \qquad c_g = (\Delta E)_{goal}$$

$$\Pi(t) = \Gamma(t) f(c_a) - (1 - \Gamma(t)) f(c_g)$$

$$f(x) = e^{0.5x} \qquad \Gamma(t+1) = (1 + e^{-\lambda \Pi(t)})^{-1}$$

$$\Pi(t) < 0 \Rightarrow avoidance \qquad \Pi(t) > 0 \Rightarrow target\_reaching$$

Moreover, when object grasping is needed, target-reaching behavior is temporarily switched to object grasping behavior; such that target reaching becomes a secondary objective and the robot tries to grasp the desired object without colliding into surrounding obstacles, until grasping is successfully completed, or canceled. The implementation of this high level architecture is realized in our work using Fuzzy Actor Critic Learning (FACL) in the individual controllers; where uncertainties such as vagueness, imprecision, and interval valuedness are modeled as fuzzy learning inference with fuzzy rules.

A. FACL Controller

There are several controller architectures in which a fuzzy logic controller is trained to tune its rule base using learning techniques, such as ANFIS [11], GARIC [10], FQL, and FACL [1]. FACL is one of these architectures introduced by Jouffe, in which the states are fuzzified using strong fuzzy partitioning and fuzzy rules are used for sensorimotor mapping. A reinforcement learning technique similar to adaptive-heuristic critic architecture of Sutton [12] is used in order to tune the output stages of the fuzzy logic controller.

A brief description of the working principle of this controller, given in Fig. 5, is as follows: For a given state ($[S_1 \ldots S_i \ldots S_n]$), activation of each rule ($R_1,\ldots,R_j,\ldots,R_N$) is a direct mapping of this state. Every rule incorporates a discrete action set ($\upsilon_1,\ldots,\upsilon_j,\ldots,\upsilon_N$) and weights ($w^j[1],\ldots, w^j[k],\ldots, w^j[|\upsilon_j|]$ for rule j) associ-

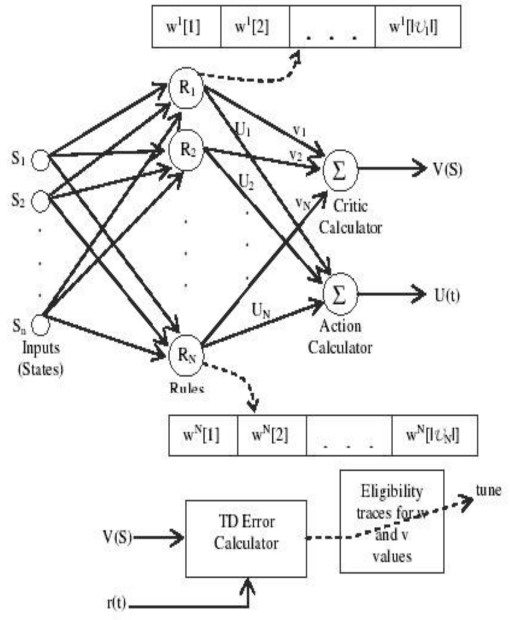


Fig. 5 General FACL Controller Structure.

ated with the elements of this set, used for exploiting action selection. At each time step, local actions ($U_1,\ldots,U_N$) from active rules are fired using an ε-greedy policy allowing both exploratory and exploiting action selection. The global action (U(t)) is calculated as the weighted sum of activated rules' individual actions, where the weights are the activation values of the rules. After application of the global action, U(t), the states change and the temporal difference error calculator is supplied with a reward value. The value function estimate of each state is calculated as the weighted sum of local value estimates ($v_1,\ldots, v_j,\ldots,v_N$) of rules (which are mapped to states). These estimates along with the reward feedback are used for calculating the temporal difference error, which is used to tune the action weight and local value function estimates of the rules. Eligibility trace and meta learning rule techniques are employed in the process to deal with the delayed reward and stable learning. Detailed explanation of this architecture may be found in [1].

B. Adaptation to Our Problem

As stated before, the control objectives of our robot is distributed between the links. The head link is the leading part; it decides on its heading velocity and angular speed at a given time, like an individual robot, which completely defines its next state. It only uses the local measurements of target position for target reaching, and sensor suit measurements for the obstacle avoidance behavior. The block diagram of head controller is shown in Fig. 6.

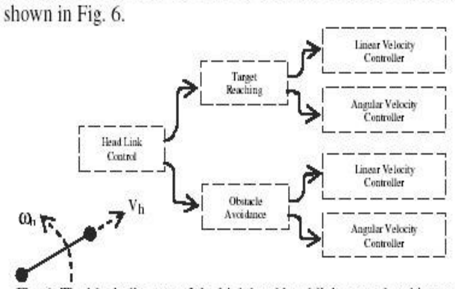

Head Link Control
Target Reaching
Obstacle Avoidance
Linear Velocity Controller
Angular Velocity Controller
Linear Velocity Controller
Angular Velocity Controller
ω_h
V_h

Fig. 6 The block diagram of the high level head link control architecture for target reaching and obstacle avoidance behaviors.

The other links (intermediate links) are all defined as a single group of "follower" links. Controllers of these links are identical and trained at once, which may be further divided into groups for the case of excessive number of links in order to avoid collisions between the links of the robot. A link controller controls the connected joint angular velocity, $\omega_{(j-1)}$, as shown in Fig. 7.

After processing of each controller, the robot decides on its next states, each state decided by individual controllers. (The head link controller determines the spatial positioning of the robot, and the other link controllers determine the internal mechanism structure, i.e. joint angles).

Each controller of the robot is made of individual FACL controllers. For the head link, the target reaching behavior controller uses (α,z) as the state feedback. The obstacle avoidance controller uses the values of d_j for j=1,...,4 and the distance sight angle β_j for j=3,4.

For the "follower" links, each behavior controls the joint angular speed between two consecutive links. For the "link following" behavior, the inputs to the controller are the current

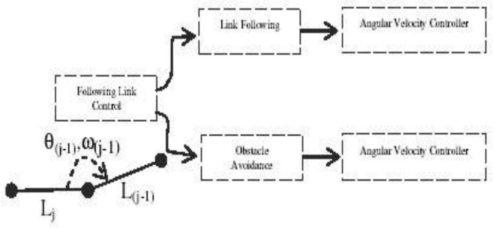

Following Link Control
Link Following
Obstacle Avoidance
Angular Velocity Controller
Angular Velocity Controller
$\theta_{(j-1)},\omega_{(j-1)}$
L_j
$L_{(j-1)}$

Fig. 7 The block diagram of the high level following link control architecture for target reaching and obstacle avoidance behaviors.

angular speed of the related angle ($\omega_{(j-1)}$,which are estimated from optical encoder readings on the hardware implementation) and the current value of the related angle ($\theta_{(j-1)}$). For the "obstacle avoidance" behavior d_1 and d_2 measurements of the sensor suits and the angular velocity of the related angle, $\omega_{(j-1)}$, are used. The block diagram of these controllers is shown in Fig. 7.

C. *Incorporating the Object Grasping Behavior*

One of the applications of snake-like robots which may be very useful in future SAR applications is the lasso-type grasping [5]. This ability of the robot may be used to carry parts in ruins of collapsed buildings and to clean the way of the robot. Moreover, a victim may be grasped and carried, or can be freed of debris pressing onto him. Hence, one of our main goals was to implement a grasping scheme during serpentine locomotion for our snake-like robot dedicated to SAR applications.

While our robot is moving in an environment, it tries to reach a previously defined target point; and avoid obstacles on the way. In a SAR application, a camera for image acquisition is employed and the user can see this image using RF techniques, outside of the robot's workspace. When an object to be grasped is seen in the camera, the robot is instructed to grasp the object by the outside rescue team. After getting the grasping command, the robot switches from "target reaching" behavior to "object grasping behavior"; while "obstacle avoidance" behavior is still active. Hence, "object grasping" is an alternative to "target reaching". When "object grasping" becomes active, "target reaching" becomes a secondary objective and it is postponed until the object is grasped successfully or the grasping operation is canceled by the user.
Our "object grasping" behavior implementation consists of three stages. First stage is for getting closer to the object to be grasped. In this stage, head link moves through the object to be grasped, while the "follower" links still apply the "link follo-

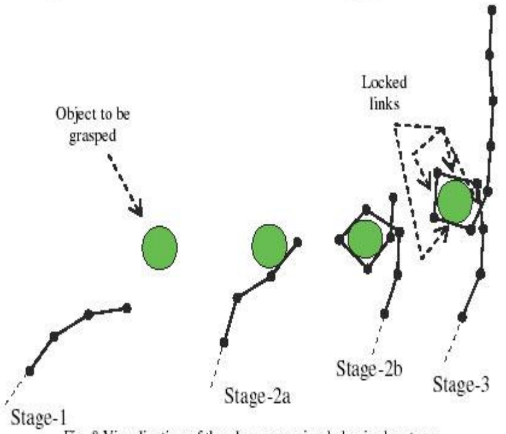


Fig. 8 Visualization of the planar grasping behavior by stages.

wing"behavior. This stage is achieved using the "target reaching" behavior with target set to the object. When the head object, beginning to enwrap the object. For the "follower" links, a link sensing the object switches from "link following" behavior to "object grasping" behavior. The aim of the link becomes the same as the head link in the second stage: going around the object and enwrapping it. In the third stage, the head finishes going around the object (turning 360 degrees in plane), while the "follower" links are still going around the object. At this point the object is surrounded by robot links, the robot decides on switching back to "target reaching" behavior, if it decides that it can carry the object, otherwise more links go around the object in order to apply higher pulling force (power grasp) to the object. After successful completion of object grasping, the links around the object become "inactive" from the control point of view, hence their angles are forced to be constant, i.e. lasso links stiffen; while the remaining links continue normal serpentine operation in the "target reaching" behavior instead of "object grasping" behavior.

Object grasping behavior is also realized with FACL controllers. For the head link, at the first stage the "target-reaching" controller is used as the target is set to be the object to be grasped (Fig. 8 Stage-1). In the second stage, distance to the object (sensed by obstacle sensors) and the total angular position change of the head (measured from the α value of Fig. 1) are used as the input to the controller. Outputs are still the heading and the angular velocities. Similarly, for the "follower" links, first the "link following" behavior is active, and when the object is sensed via obstacle sensors "object grasping" behavior is turned on. In this behavioral stage, the distance to the object, controlled angle angular speed and angle value are used as inputs, while the output is still the angular velocity of the angle between the link and the preceding link. When head completes a single turn around the object (Fig.8 Stage-2b), it switches back to "target reaching" behavior with target is set to the original position (Stage-2b). A "follower" link which completes the turn also switches back to "link following" behavior. When the robot decides that the power grasp may be succeeded, the grasping links are locked, and their link angles are fixed, while all the other links continue in "link following" ("target reaching" for the head link) behavior (Fig.8 Stage-3

D. *Dynamic Serpentine Gait Selection*

Up to this point of the paper, dynamics are somehow ignored, there are several studies on snake dynamics and implementation of snake-like motion in robotics such as [2],[3],[4], and [5]. In similar control studies, [6] and [7], previously memorized gaits are used for motion. However, these approaches are not generic and their origin comes from trying to mimic the locomotion of real snakes by link motions. In our study, we use an on-line optimization approach similar to one in [8]. A matrix of individual entries corresponding to the link angles at different time steps is used. This is an N x (t/T_s) matrix, where N is the number of links, t is the time length of the movement, T_s is the execution time step. A genetic algorithm implementation is, then, used to evaluate the best matrix, resulting in a best-fit solution to the desired position in state-space at each time length of the movement, which is designed to be short enough to relax the genetic algorithm run-time (whereas in [8] the optimization algorithm is used to make the robot learn a complete gait). As the result, the active FACL controllers produce the state of the robot for the next time, and the genetic algorithm is used to search the necessary angular movements in time, to achieve the desired position dynamically. With this approach, we incorporate the environmental conditions into gait programming. The drawback is that, the friction coefficients must be known (or estimated) in order to correctly evaluate the resulting motion in space due to changing link angles. Note also that, low-level control of link actuators are not taken into account and assumed to be ideal.

IV. Simulations and Results

Training simulations were performed in several steps. Which are:

1-Training of the head link for reaching the target: At first, the angular velocity FACL controller is trained to make the robot to align itself to the target (make α zero). Secondly, the head link learns to reach the target (making z zero).

2- Training of the "follower" links for the "link following" behavior: While the head link reaches the target, the second link learns to follow it. Then the trained controller is copied to other "follower" links, resulting in a robot trained to reach the target in an environment with no obstacles.

3- Training of the head for the obstacle avoidance behavior: A map with obstacles is constructed and the head was directed to the target. When it detects an obstacle with its sensor suits, its behavior is switched to "obstacle avoidance" and trained with collision punishments. Hence the head learns to escape from obstacles.

4- Training of "follower" links for obstacle avoidance: Up to this step, the head learnt to escape from obstacles, and the "follower" links learnt to follow the head. During training of obstacle avoidance controllers of the "follower" links, a link moving with the "link following" behavior encounters an obstacle, and switches to "obstacle avoidance" behavior; with its FACL learning procedure initiated.

5- Training of the head for object grasping: With no obstacles, and the presence of an object to be grasped in the map, the head learns to come as close as to the object boundary, and make a 360 degrees turn around it; thus it learns to wrap around the object by sliding on its boundary.

- Training of the "follower" links for object grasping: Similar to the head segment, when a "follower" link senses the object boundary to be grasped, it switches to "object grasping" behavior, which is trained to go as close as possible to the obstacle with the head link sliding around the object.

7- Incorporating the genetic search algorithm: Our snake-like robot moves using no active limbs; it crawls in the plane and the resulting friction forces give a net displacement of links.

The new position commands from the active FACL controllers, which are trained to find them, are converted to link angle motions using the genetic search algorithm, using sinusoidal modal functions.

An example of steps 1,2 and 7 are shown in Fig. 9. In this demonstrative case, a robot of 6-links is trained to reach a target in a 10mx10m map. The links are 50 cm in length and 5 cm in width, with a mass of 1 kg each, with a corresponding 0.0208 kg.m^2 moment of inertia with respect to center of mass. The Coulomb friction model is used with tangential and normal friction coefficients of 0.1 and 0.5 respectively.

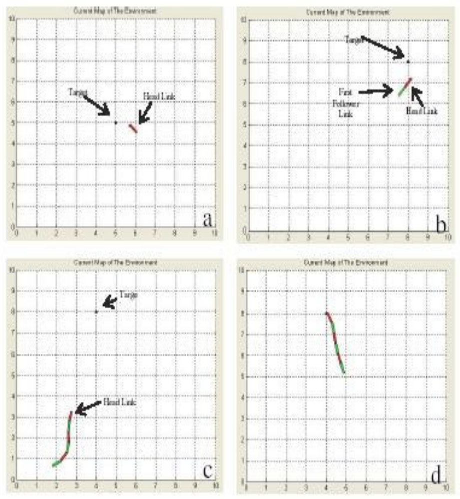


Fig. 9 Simulation of target reaching without any obstacles.

Demonstration of the steps 3 and 4, with 7 incorporated are given Fig. 10, a,b and c respectively. First the head learns to avoid obstacles while reaching to the target as before (10a and 10b), then the first "follower" link is learnt to deal with the same problem by altering the angular speed of the angle between it and the head link, resulting in (10c) when the trained controllers are executed in all links. In the simulations, obstacles in the range of 50 cm from the links were assumed to be sensed (i.e., farther obstacles are not sensed).

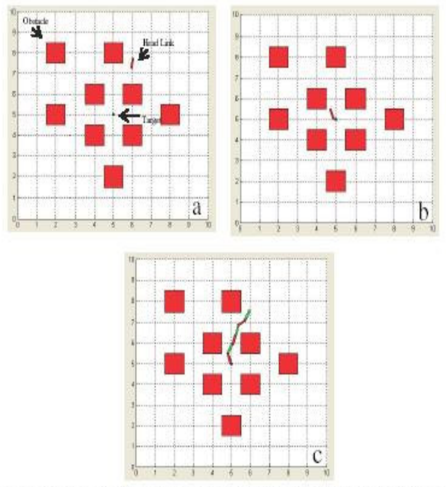


Fig. 10 Simulation of obstacle avoidance behavior learning for the 6-link robot.

The trained controllers were executed in several different environmental maps, having different configurations of obs tacles and targets. Three of these maps are shown in Fig. 11, with results obtained as in Table-1. The "Initial Success" column shows the number of successful reaching of the robot to the target in 100 trials, the "F.T. Success" column (F.T. = Further Training) shows the successful number of duties, after further training (100 episodes) in the environment (for the obstacle avoidance behavior); showing that, the snake robot adapts itself to different maps for terrainability. Considering a typical SAR application, in a single ruin, snake robot may be either trained for that ruin by several trials, or by first gathering the environmental map with a simpler scout robot sent first into the rubbles.

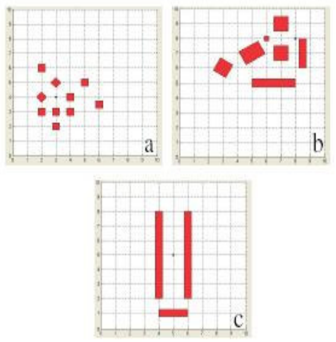


Fig. 11 Some example maps for evaluating target reaching and obstacle avoidance controllers developed. Red parts are obstacles, and the blue point is the target

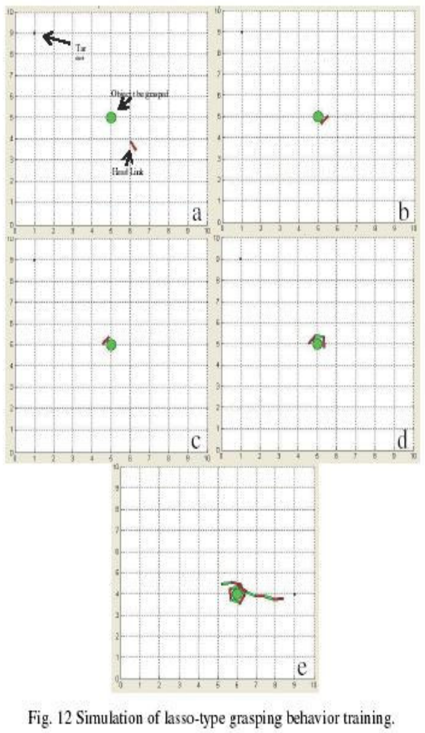


Fig. 12 Simulation of lasso-type grasping behavior training.

For the simulation of steps 5,6 with 7 incorporated, a 12-link planar robot is used, and the robot learns to grasp and drag a circular object in the environment with serpentine locomotion. The simulation of training for grasping is shown in Fig. 12. Training steps for the head link are shown in Fig 12a, 12b and 12c. In Fig 12d, a "follower" link had learnt to go closer to the object, and its grasping behavior controller is copied to the third link.In Fig 12e, full configuration of the robot is seen grasping the circular object.

TABLE I
SIMULATION RESULTS FOR THREE DIFFERENT MAPS

MAP	INITIAL SUCCESS	F.T. SUCCESS
a	60%	94%
b	64%	96%
c	85%	100%

REFERENCES

[1] Jouffe, L., "Fuzzy Inference System Learning by Reinforcement Methods," IEEE Trans. Systems, Man, And Cybernetics, Vol. 28, No. 3, pp. 338-355, Aug. 1998.
[2] Saito M., Fukaya M. and Iwasaki T., "Serpentine Locomotion with Robotic Snakes," IEEE Control Systems Magazine, pp. 64-81, Feb. 2002.
[3] Liu H., Yan G. and Ding G., "Research on the Locomotion Mechanism of Snake-like Robots," in Proc. IEEE Int. Symposium on Micromechatronics and Human Science, pp. 183-188, 2001.
[4] Ma S., "Analysis of Snake Movement Forms for Realization of Snake-like Robots," in Proc.of IEEE Int. Conf. on Robotics & Automation, Detroit, Micihigan , May 1999, pp. 3007-3013.
[5] Chirikjian G.S. and Burdick J.W., "The Kinematics of Hyper-Redundant Locomotion," IEEE Tran. Robotics And Automation, Vol. 11, No.6, pp. 781-793, Dec. 1995.
[6] Gevher, M., "Sensor Based On-line Path Planning For Serpentine Robots," M.Sc. Thesis, Middle East Technical Uni., Ankara, Dec. 2001.
[7] Kulali, G.M., "Intelligent Gait Synthesizer for Serpentine Robots," M.Sc. Thesis, Middle East Technical Uni., Ankara, Dec. 2001.
[8] Dowling K. , "Limbless Locomotion: Learning to Crawl," in Proc. IEEE Int.Conf. on Robotics and Automation, Detroit, Micihigan , May 1999, pp. 3001-3006.
[9] Beom, H.R. and Cho H.S., "A Sensor-Based Navigation for a Mobile Robot Using Fuzzy Logic and Reinforcement Learning," IEEE Trans. Systems, Man, And Cybernetics, Vol. 25, No. 3, pp. 464-477, March 1995.
[10]Berenji, H.R. and Khedkar, P., "Learning and Tuning Fuzzy Logic Controllers Through Reinforcements," IEEE Tran. Neural Networks, Vol. 3, No.5, pp. 724-740, Sept. 1992.
[11]Jang J.R., "Self-Learning Fuzzy Controllers Based on Temporal Back Propagation ," IEEE Tran. Neural Networks, Vol. 3, No.5, pp. 714-723, Sept. 1992.
[12]Sutton R.S. and Barto A.G., "Reinforcement Learning: An Introduction," MIT Press, Cambridge, MA, 1998.

FACL Based 3D Grasping Controller for a Snake Robot During Locomotion

Double Blind

Double Blind

Abstract **– In a previous study [1], a distributed intelligent fuzzy learning controller architecture was proposed for a planar snake robot able to avoid obstacles and grasp 2D objects by enwrapment. In this paper, we extend that controller for serpentine locomotion and lasso-type grasping in 3D environments. The novelty of our approach is not only in the control modularity but on a lasso type grasping having as a moving base the remaining snake links undergoing serpentine gait. Dynamical constraints due to environmental conditions are considered online by a genetic algorithm based evaluator of suggested behaviors by the high level controller. Our approach is demonstrated on the simulation of our modular 12 link snake robot.**

Index Terms – snake robots, biologically inspired robots, search and rescue robotics, lasso- type grasping, FACL.

I. INTRODUCTION

Recent experiences of natural disasters (earthquakes, tsunami etc.) and man made catastrophes (terrorism) have put more emphasis to the area of Search and Rescue (SAR) and emergency management. Tool developments for such applications that necessitate autonomy, high mobility, robustness and reconfigurability for terrain and task adaptation have been a recent focus, where snake-like robots [2] have revealed to be versatile for locomotion trough ruins of collapsed buildings or narrow passages such as pipelines. Snake-like robots are advantageous over wheeled vehicles for terrainability, traction, universal penetration capabilities, high adaptability and task shapability due to kinematic redundancies, which offer graceful degradation, increasing reliability when made modular [3],[4],[5]. Terrainability still challenges the problem of obstructed but evacuable passages or victims traumatized by being squeezed in narrow spaces that could be enlarged. Serpentine devices can be candidates to meet these challenges still possess the disadvantage of payload, speed, complexity and uncertainty handling in motion gait planning and control due to high degrees of freedom [6]. In this work, we particularly focus on task shapability using snake redundancy for lasso-type grasping, dragging tasks; with the remaining portion undergoing serpentine gaits. Control complexity and uncertainty problem created when grasping during gait planning and execution are overcome by a distributed fuzzy actor critic learning (FACL) controller architecture, first introduced by Jouffe [7], that we modify for our serpentine robot coupling it with an on-line optimization technique modified from Dowling's studies [8] for dynamic gait configuration changes. Moreover, this paper focuses on developing a 3D grasping controller for enwrapping and dragging an object with serpentine motion. The motivation behind this work is that this lasso type grasping while serpentine locomotion has been a critical need in SAR robot missions to enlarge passages around victims or while passing through an evacuable but blocked area during exploratory navigation or transportation.

II. HYPER-REDUNDANT SNAKE-LIKE ROBOT MODEL

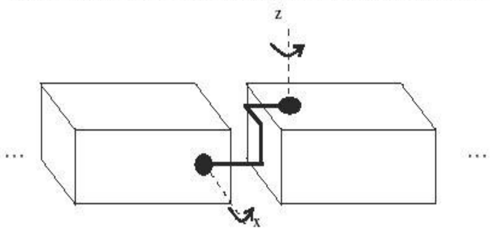


Fig. 1 – Schematic showing the 3D robot joints each having 2 degrees of freedom.

In a previous study [1], a high level controller architecture was developed for a planar (2D) snake-like robot for demonstration purposes. However, in this work, we use a robot that can move in 3D space [6] with each robot link connected by a 2 DOF joint enabling yaw and uniquely directional pitch motion as shown schematically in Fig. 1.

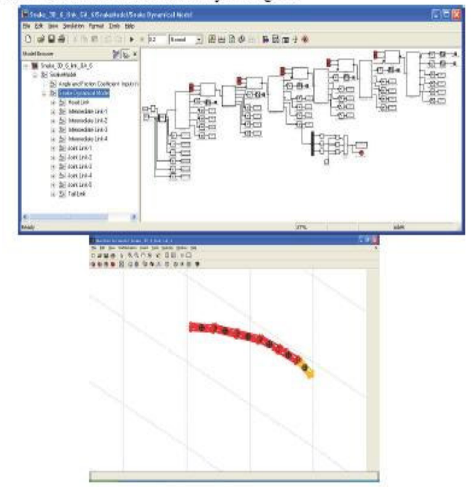


Fig. 2 – The Simulink diagram showing the link level in the Simulink hierarchy and a snapshot from the Simmechanics visualization of a 6-link snake robot

The links are mechanically identical and our robot is segment modular such that these mechanically identical links are controlled individually in the highest control level, making the whole structure a robotic link network. The sensory structure of the robot is grouped into suits, each giving the distance to the closest obstacle through the side sensors and distance and angular position of the closest obstacle by the frontal sensors. There are 2 sensor suits for side sensing on each body link. The head link has 2 frontal sensor suits and 2 oppositely located side sensor suits. In this paper, for demonstrative purposes of our novel architecture, we provide the dynamical simulations of the robot in the MATLAB environment using Simmechanics blockset of Simulink as shown in Fig. 2.

III. The General FACL Gait Controller Architecture

For the sake of completeness of this paper, we present here a brief summary of all high-level control architecture and then concentrate further on the grasping controller embedded in that high-level architecture.

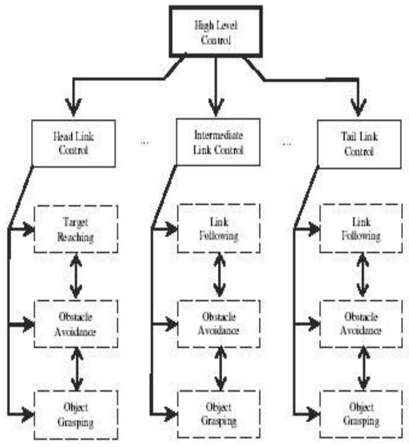


Fig. 3 The block diagram of the highest-level control architecture.

The segment modularity of our robot is enhanced at the control level by a control modular architecture such that the different duties of the controller are carried out by different modules. More than being a distributed control structure, our architecture is a control network.

The basic control duties connected together on the control network are "target reaching" and "obstacle avoidance" behaviors which are the classical ones [9], extended by the "object grasping" behavior for grasping desired objects during serpentine motion. The three control behaviors are further distributed throughout the robot network on the individual links as separate controllers. Our approach yields the general controller architecture block diagram as shown in Fig. 3.

As the result, we have a segment modular and control modular high-level architecture, i.e. a robot network overlaid by a control network, where each segment applies an individual behavior for a given state. When target-reaching behavior is active, the head link tries to reach the target, as if its an individual robot, and the other links try to follow the immediate preceding link. In the existence of nearby obstacles, a link may switch to obstacle avoidance behavior individually such that there is a competition between control behaviors. Moreover, when object grasping is needed, target-reaching behavior is temporarily switched to object grasping behavior; such that target reaching becomes a secondary objective and the robot tries to grasp the desired object without colliding into surrounding obstacles, until grasping is successfully completed, or canceled. The implementation of this high level architecture is realized in our work using Fuzzy Actor Critic Learning (FACL) in the individual controllers; where uncertainties such as vagueness, imprecision, and interval valuedness are modeled as fuzzy learning inference with fuzzy rules. We now concentrate on the object grasping control module.

A. *Object Grasping Control Stages*

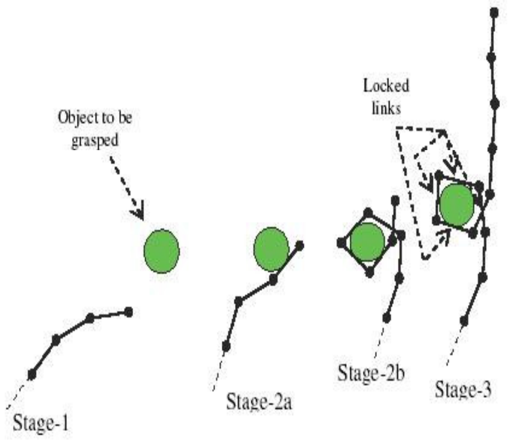


Fig. 4 Planar visualization of the "object grasping behavior."

During "object grasping behavior" there are three stages as shown in Figure 4. In the first stage the target changes its identity as "the object to be grasped" and the "target reaching behavior" is activated for all the link controllers, target meaning object here. The second stage is the initiation of enwrapment where, the robot has reached the object to be grasped and starts to enwrap it. At this stage, a link that senses the object to be grasped in its sensory space switches to "object grasping controller". At that instant a signal is sent on the control network to first switch the head link to object grasping mode then the following link controllers switch to their "object grasping" controller modes. At the end of the second stage the robot links enwrap the object to be grasped. At the third stage, the robot has fully enwrapped the object, and any link that completes enwrapping switches back to "target reaching" behavior with the target identity set back to the global target at hand that has to be reached by locomotion. At the end of this last stage, the robot locks its enwrapping links and all links perform "target reaching behavior" with the robot undergoing serpentine gaits towards the global target.

B. *Details of Object Grasping Controller Module*

"Object grasping controller" is used for a link when it starts to enwrap the object during control stage 2 of the object grasping behavior. The aim of the object grasping controller is making the link move "as close as possible" to the object to be grasped; while not allowing any collisions with the object to be grasped, and making the link to "turn around" the object and land onto it with fence closure.

Grasping controllers are two types, one is for the head and the other is for the following links.

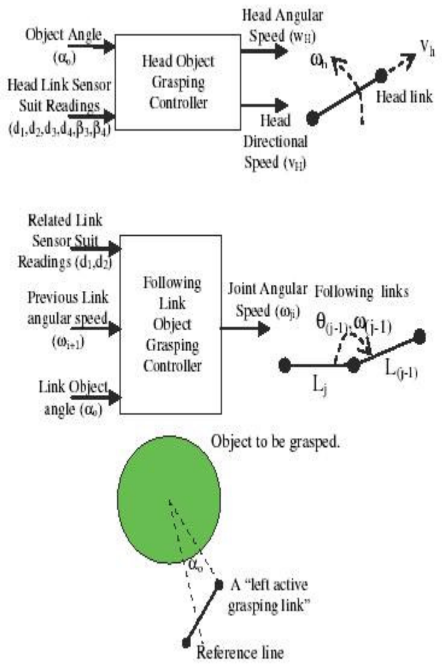

Object Angle (α_o)
Head Link Sensor Suit Readings ($d_1,d_2,d_3,d_4,\beta_3,\beta_4$)
Head Object Grasping Controller
Head Angular Speed (w_H)
Head Directional Speed (v_H)
ω_h
V_h
Head link
Related Link Sensor Suit Readings (d_1,d_2)
Previous Link angular speed (ω_{i+1})
Link Object angle (α_o)
Following Link Object Grasping Controller
Joint Angular Speed (ω_{ji})
Following links
$\theta_{(j-1)}$, $\omega_{(j-1)}$
L_j
$L_{(j-1)}$
Object to be grasped.
α_o
A "left active grasping link"
Reference line

Fig. 6 – Input and outputs for the "object grasping controller"s and the definition of the object angle α_o

For the head link "object grasping controller", the controlled variables are the instantaneous angular and directional velocities of the head link. For the following link object grasping controllers, the controlled variable is the angle of the front joint. The structure of the FACL controller is shown in Fig. 7.

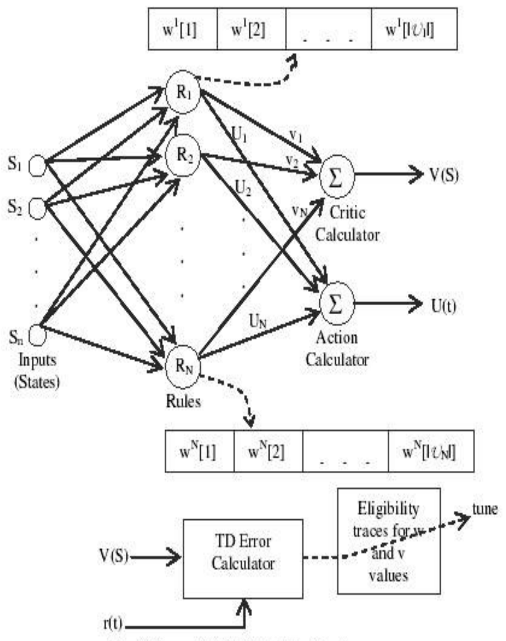


Fig. 7 General FACL Controller Structure.

In this structure, the input states Si's are the variables input to the controllers as shown in Fig. 6, being different for the head and for the following links. The input state vector S ($[S_1\ S_2\ \ldots\ S_n]^T$) activates the rules ($R_i$'s). In our example implementation we have 5103, and 63 rules for head and following link object grasping controllers respectively due to sample input sets given in Table II. The Ui's are elements of the action sets which are shown as the "angular speed" and "directional speed" in the same table. At each time step, local actions ($U_1,\ldots,U_N$) from active rules are fired using an ε-greedy policy allowing both exploratory and exploiting action selection. The global action (U(t)) is calculated as the weighted sum of activated rules' individual actions, where the weights are the activation values of the rules. After application of the global action, U(t), the states change and the temporal difference error calculator is supplied with a reward value, r(t). The reward generation mechanism in the object grasping controllers is the linear combination of three measures representing the above aims, and it is given as

$$r = k_\alpha \Delta m_\alpha + k_d \Delta m_d + k_h \Delta m_h$$

Where Δm_α, Δm_d, Δm_h are measures for angular, distance, and hitting the object, respectively and k's are related weights for linear combination.

The value function estimate of each state is calculated as the weighted sum of local value estimates ($v_1,\ldots, v_j,\ldots,v_N$) of rules (which are mapped to states). These estimates among with the reward feedback are used for calculating the temporal difference error (TD error), which is used to tune the action weight and local value function estimates of the rules. Eligibility trace and meta learning rule techniques (as given in

[7]) are employed in the process to deal with the delayed reward and stable learning. The outputs of the controllers for the different links are as shown in Fig. 6.

C. *Dynamic Serpentine Gait Selection While Grasping*

Gait selection dragging a grasped load uses an on-line optimization approach similar to one in [8] (using Fourier series coding). A matrix of individual entries corresponding to the link angles at different time steps is used. This is an N x (t/T_s) matrix, where N is the number of links, t is the time length of the movement, T_s is the execution time step. A genetic algorithm implementation is, then, used to evaluate the best matrix, resulting in a best-fit solution to the desired position in state-space at each time length of the movement, which is designed to be short enough to relax the genetic algorithm run-time (whereas in [8] the optimization algorithm is used to make the robot learn a complete gait). This approach is shown schematically in Fig 8.As the result, the active FACL controllers produce the desired gait state of the robot for the next time, and the genetic algorithm is used to search the necessary angular movements in run time, to best fit the desired gait dynamically.

With this approach, we incorporate the environmental conditions into gait realization based on the feasibility of the desired gait by the FACL high level controllers. The drawback is that, the friction coefficients must be known (or estimated) in order to correctly evaluate the resulting motion in space due to changing link angles since we have an object to drag. In our applications we estimate these coefficients for certain prototypical surfaces. Note also that, low-level control of link actuators are not taken into account and assumed to be ideal.

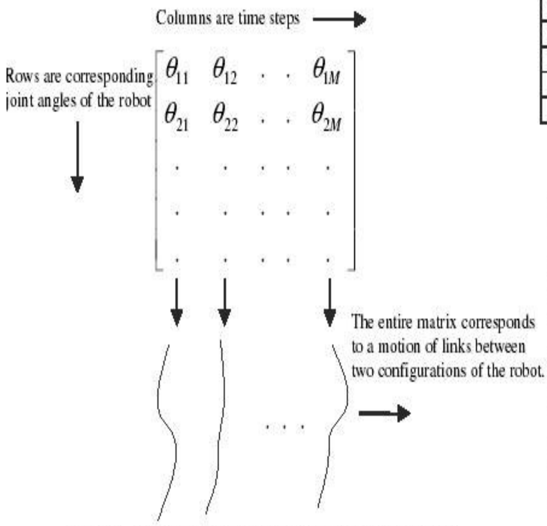

Columns are time steps
Rows are corresponding joint angles of the robot
θ_{11} θ_{12} . . θ_{1M}
θ_{21} θ_{22} . . θ_{2M}
The entire matrix corresponds to a motion of links between two configurations of the robot.

Figure 8 – Representation of the motion generation via genetic algorithms and motion matrix.

As the result of this approach, that tunes -using GA- the FACL generated desired gaits, we have the feedback architecture shown in Fig. 9.

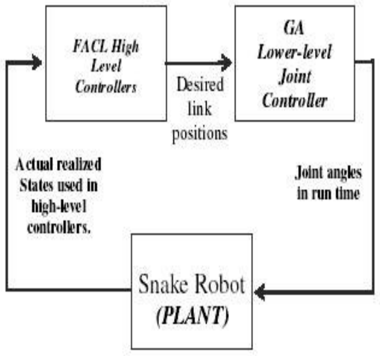

FACL High Level Controllers
Desired link positions
GA Lower-level Joint Controller
Actual realized States used in high-level controllers.
Joint angles in run time
Snake Robot (PLANT)

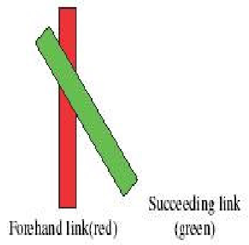
Figure 9 – Block diagram showing the feedback architecture of the system.

For a robot having N links, we have 10+10+(N-1)x10x2 parameters as the input vector of the fitness function (chromosomes) (10 for amplitude and frequency coding, (N-1)x10x2 for phase coding). This means, number of parameters increases as the number of links of the robot increases. Hence, genetic algorithm may limit the number of segments of the robot used. Critical parameters of the genetic algorithm used for the 12-link robot is shown in Table I.

TABLE I
CRITICAL PARAMETERS OF THE GENETIC ALGORITHM USED FOR THE 12-LINK SERPENTINE ROBOT

Parameter	*Value Used*
Population Size	20
Crossover	Intermediate / Ratio: 0.5
Fitness Function	Position Error & Energy Consumption
Mutation	Uniform / Rate : 0.05
Elite Count	2
Selection Rule	Tournament

In [1], the controller architecture is on a planar snake robot and for its grasping the collisions between the grasping links are ignored by those authors, which in fact is unavoidable on a plane. However, for a robot that can move in 3D, precautions must be taken to avoid such collisions. We achieve this by raising the colliding link within an acceptable safety margin above the ground. If a head following link is colliding with its consecutive links, then the immediate consecutive links raise in a triangular bridge over their predecessor links to allow a passage for that preceding link. This is achieved by using a "height envelope" around the forehand link that avoids collision in space with its succeeding links (Fig. 10).



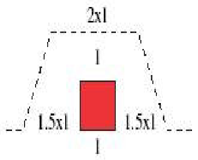


Figure 10 – A representation of two links colliding in plane, and the envelope function representation for the forehand link.

IV. Simulations and Results

For demonstrative purposes of our architecture we provide simulations whose results are grouped into three parts. The first one dwells on the learning performance of the FACL controllers used. The second is an execution performance of FACL controllers after the learning phase has been completed. Finally, the third part of the results demonstrates the performance of the genetic algorithm for realizing grasping and locomotion in space.

A. Training of Object Grasping Controllers

Since our high-level controller architecture is modular, training of individual controllers may be performed independently. We use this property extensively in this study. Hence, we first train the head object grasping controller. When this learning phase is completed, we pass to the training of the following link grasping controllers. At the end of the trainings we have a robot that can grasp an object, which is circular in Fig 11. These training phases are shown in Fig. 11.

The performance of the FACL training may be evaluated using the learning speed criteria, as done in [7]. Learning speed is measured using the number of training episodes needed before a 50 successive episodes ending in success. If number of training episodes exceeds 10000, then we claim that the FACL parameters used are not suitable and must be changed.

Our robot links are 50 cm in length and 5 cm in width. The grasped object is assumed here in the example to have circular cross section. A 25 cm radius circle is used for the training phase.

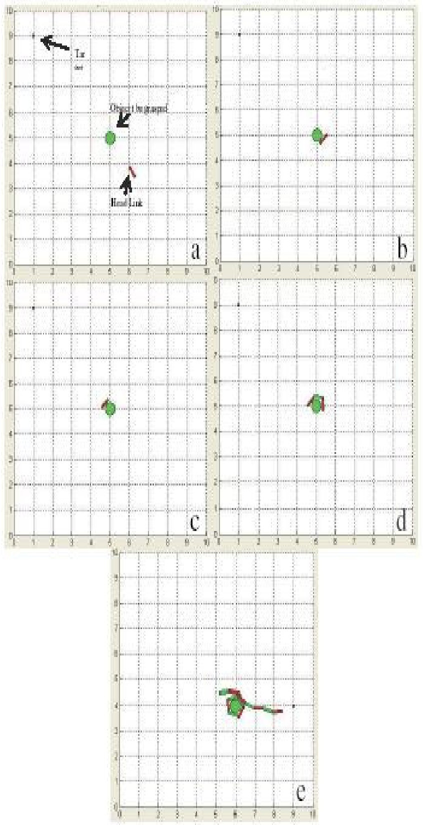


Fig. 11 Simulation of lasso-type grasping controller training.

The parameters fuzzy sets used for the inputs of the FACL controllers are shown in Figure 12 and Table II respectively. The same discrete action sets shown in this table are used for all the related rules shown. The critical parameters for the learning performance of the object grasping controllers are the discount factor of the reinforcement learning, the recency factor of the eligibility traces for the weight and value vectors of the rules, and the initial critic learning rate.

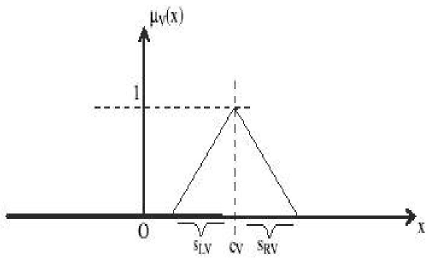


Figure 12 – Parameters of a typical triangular fuzzy set defined for the inputs.

TABLE II
FUZZY SET PARAMETERS FOR DIFFERENT INPUT AND OUTPUT VARIABLES OF OBSTACLE AVOIDANCE CONTROLLERS

VARIABLE	FUZZY SET	S_{LV}	C_V	S_{RV}
α_o	NVS	0	$-\pi$	$\pi/3$
	NS	$\pi/3$	$-2\pi/3$	$\pi/3$
	N	$\pi/3$	$-\pi/3$	$\pi/3$
	Z	$\pi/3$	0	$\pi/3$
	P	$\pi/3$	$\pi/3$	$\pi/3$
	PB	$\pi/3$	$2\pi/3$	$\pi/3$
	PVB	$\pi/3$	π	0
Sensory Distances (d_i)	S	0	0	0.5
	M	0.5	0.5	0.25
	B	0.25	0.75	∞
Sensory Angles (β_3,β_4)	S	0	0	$\pi/4$
	M	$\pi/4$	$\pi/4$	$\pi/4$
	B	0	$\pi/2$	$\pi/4$
Angular speed (ω)	NB	0	$-\pi/4$	0
	NS	0	$-\pi/8$	0
	Z	0	0	0
	PS	0	$\pi/8$	0
	PB	0	$\pi/4$	0
Directional Speed (v)	NB	0	-2	0
	NS	0	-1	0
	Z	0	0	0
	PS	0	1	0
	PB	0	2	0

With these fuzzy set parameters (shown in Table II) for the inputs and outputs, we have 3x3x3x3x3x3x7=5103 rules and 5103*5*5=127575 weights for the head object grasping controller. For the following link object grasping controller we have 3x3x7=63 rules and 63*5=315 weights. Hence, for the head part the defined task is really very complicated and we expect a slower learning than the following link parts.

Reward generation scheme is another very important issue that affects the performance of the learning of desired controller. In this study we implement a scheme that incorporates the three properties described in the previous section. If a collision with the object occurs we end the episode with a failure (reward of -10), or if the head gets away from the object such that the object is out of sensory space, we again give a reward of -10 and end the episode with a failure. Otherwise we give all negative rewards related to distance measure and reference object angle changes. If the reference angle completely sweeps 360 degrees we give a reward of +10 and end the episode with a success. The reward is calculated as

$$r = k_\alpha \Delta m_\alpha + k_d \Delta m_d$$

and it is truncated to 0 if it is positive. Otherwise, the controller desires to continue the episode for making small positive increments, instead of finishing it with a positive big reward.

The learning speeds for different FACL parameters are given in Table III. The parameter γ is the discount factor of the reinforcement learning algorithm. λ and λ_a are the recency factors of the value and weight eligibility traces respectively. β_0 is the initial learning rate for the rule values.

TABLE III
LEARNING SPEED MEASURES FOR DIFFERENT FACL PARAMETERS FOR HEAD AND FOLLOWING LINK OBJECT GRASPING CONTROLLERS

FACL Parameters vector ***[γ λ λ_a β_0]***	***Number of Needed Episodes Head Learning – Following Link Learning***
[0.1 0.1 0.1 0.0001]	N.A. - N.A.
[0.5 0.1 0.5 0.0005]	N.A. - N.A.
[0.9 0.1 0.9 0.001]	8622 – 1633
[0.1 0.5 0.1 0.0001]	N.A. – N.A.
[0.5 0.5 0.5 0.005]	N.A. – N.A.
[0.9 0.5 0.9 0.001]	7643 – 1461
[0.1 0.9 0.1 0.0001]	N.A. – N.A.
[0.5 0.9 0.5 0.0005]	N.A. – N.A.
[0.9 0.9 0.9 0.001]	6429 – 1294
[0.9 0.9 0.9 0.0001]	5622 – 963

As seen in this table, the reinforcement learning discount factor parameter has a drastic effect on the learning, as expected. One must choose this parameter close to 1, but not that close such that every reward is not given the same importance. Similarly, the eligibility trace recency factor parameters should be chosen close to 1 in order to speed up learning. However, we should note that as these parameters are close to 1, the memory space needed to store the visited states increases. For the learning rate parameter, 0.0001 seems to be optimal.

B. Evaluation of Object Grasping Controllers

In order to evaluate the performance of the trained FACL object grasping controllers we use our 12 link serpentine robot, simulated in MATLAB environment. In these simulations, we run for 100 episodes and calculate the percentage of success. For the object grasping behavior alone, only a single map is used, since the only change that occurs is in the position of the object to be grasped, which is not an important parameter for the object grasping controllers since for our work the object can be grasped at any contact on its surface, the object grasping controller was successful in 98 trials. We also perform simulations which include object grasping in environments where obstacles are to be avoided and a target to be reached.

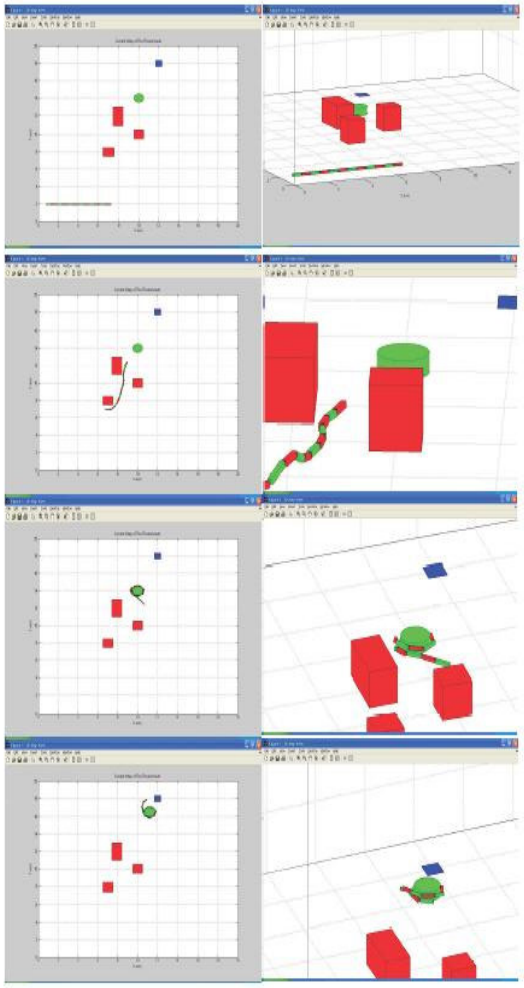


Figure 13 – Visualizations from a run of the developed high level controller, performing target reaching, obstacle avoidance and object grasping behaviors. Top views are on the left, 3-D views on the right.

In Fig. 13 a typical simulation is shown for a demo map. In the map the green cylinder represents the object to be grasped; the red prisms are obstacles which should be avoided; and the center of the blue planar square is the target point to be reached. As shown in this figure, the robot first approaches the object while avoiding the obstacles, then grasps the object and locks the grasping segments. Finally it moves to the target point while carrying the grasped object.

In Fig. 14, the distance of the tip of the head segment's planar projection to the object to be grasped is shown. As seen in this figure, target reaching and obstacle avoidance controllers work for 123 time steps. After this time object grasping controller is active for the remaining 105 time steps, and the distance to the center of the cylindrical object is between 0.6-0.7 m, where the radius of the object is 0.5 m.

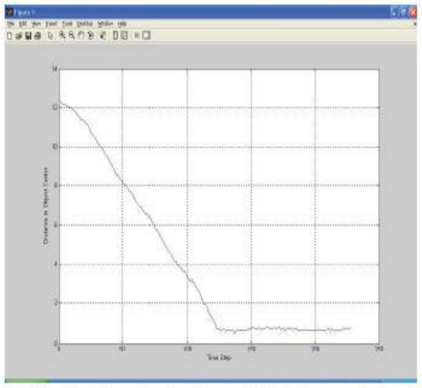


Figure 14 – Graph showing the distance of the head segment's tip point planar projection to the center of the object to be grasped during object grasping behavior.

C. *Performance of the Genetic Algorithm*

The evaluation of the genetic algorithm performance is carried out by investigating the necessary number of generations in order to make a transition between two successive configurations (given by the high level controller) of the robot. For the demo map given in Fig. 13, this number was 18.43 in average.

Another point to stress for the high level FACL controllers and lower level genetic algorithm interaction is the error between the final robot configuration desired by the FACL controllers and the configuration reached by the genetic algorithm. In our approach, we use a fitness function which is a linear combination of the cumulative position error and the total energy consumption. Hence, zeroing of position error should not be ideal. The high level controller desires a next position, but genetic algorithm could not exactly reach it; rather it approaches that position. The cumulative position error (sum of absolute values of errors in position of the 4 vertices of a prismatic link for all robot links) average for the demo map is 46.3 cm. Hence for a single vertex, this error is (46.3 / 12)/4 = 0,963 cm.

V. CONCLUSION

Snake robots possess a capability of traction and manipulability in complex, unstructured environments. Since they can not bee equipped with external manipulator, a new object manipulation technique need to be generated and controlled to increase the versatility of sech robots in object handling for dragging debris and enlarging narrow passages in disaster areas of SAR applications. Enwrapment grasping with snake robots have been performed mainly in planar cases or with tentacles, an elephant trunk with fixed base. Our work focuses on the 3D lasso type grasp control embedded in a novel high level FACL snake motion (locomotion + grasping) control architecture that propose the desired gaits for the required control behavior. The desired gait is further tuned by GA to accommodate to current dynamical conditions. Our approach reveals itself highly efficient for implementing SAR serpentine devices that not only navigates among rubbles but also enlarge passages to pass through or relieve a squeezed victim.

REFERENCES

[1] Ari E.O., Erkmen I., Erkmen A.M., "An FACL Controller Architecture for a Grasping Snake Robot," in Proc. of IEEE International Conference on Intelligent Robots and Systems (IROS), August 2005, pp. 3339-3344.

[2] Saito M., Fukaya M. and Iwasaki T., "Serpentine Locomotion with Robotic Snakes," IEEE Control Systems Magazine, pp. 64-81, Feb. 2002.

[3] Liu H., Yan G. and Ding G., "Research on the Locomotion Mechanism of Snake-like Robots," in Proc. IEEE Int. Symposium on Micromechatronics and Human Science, pp. 183-188, 2001.

[4] Ma S., "Analysis of Snake Movement Forms for Realization of Snake-like Robots," in Proc.of IEEE Int. Conf. on Robotics & Automation, Detroit, Micihigan , May 1999, pp. 3007-3013.

[5] Chirikjian G.S. and Burdick J.W., "The Kinematics of Hyper-Redundant Locomotion," IEEE Tran. Robotics And Automation, Vol. 11, No.6, pp. 781-793, Dec. 1995.

[6] Gevher, M., "Sensor Based On-line Path Planning For Serpentine Robots," M.Sc. Thesis, Middle East Technical Uni., Ankara, Dec. 2001.

[7] Jouffe, L., "Fuzzy Inference System Learning by Reinforcement Methods," IEEE Trans. Systems, Man, And Cybernetics, Vol. 28, No. 3, pp. 338-355, Aug. 1998.

[8] Dowling K. , "Limbless Locomotion: Learning to Crawl," in Proc. IEEE Int.Conf. on Robotics and Automation, Detroit, Micihigan , May 1999, pp. 3001-3006.

[9] Beom, H.R. and Cho H.S., "A Sensor-Based Navigation for a Mobile Robot Using Fuzzy Logic and Reinforcement Learning," IEEE Trans. Systems, Man, And Cybernetics, Vol. 25, No. 3, pp. 464-477, March 1995.