A SNAKE-LIKE ROBOT FOR SEARCHING, CLEANING PASSAGES FROM DEBRIS AND DRAGGING VICTIMS

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

OF

THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

ENGİN ÇAĞLAV

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

NOVEMBER 2006

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

Examining Committee Members

Prof. Dr. İsmet ERKMEN (METU, EE Chairperson)	
Prof. Dr. Aydın ERSAK (METU, EE)	
Prof. Dr. Aydan M. ERKMEN (METU, EE)	
Asst. Prof. Dr. Afşar SARANLI (METU, ME)	
Assoc. Prof. Dr. H. Gökhan İLK (Ankara Univ., EE)	

ABSTRACT

A SNAKE-LIKE ROBOT FOR SEARCHING, PASSAGES FROM DEBRIS AND DRAGGING VICTIMS

ÇAĞLAV, Engin
MSc., Department of Electrical and Electronics Engineering
Supervisor: Prof. Dr. Aydan M. ERKMEN
Co-Supervisor: Prof. Dr. İsmet ERKMEN
November 2006, 190 pages

In this thesis, a snake like robot is implemented for search and rescue applications. The "snake" is intentionally selected as a reference for their ability to move on various environments, but due to the mechanical limitations the implemented snake-like robot design could not be close to the biological counterparts. Although the implemented snake like robot is not a replica of biological snakes; it captured key aspects of snakes such as flexibility, redundancy and high adaptation.

To depart from the mechanical limitations; a model of the implemented robot is designed in MATLAB - SIMMECHANICS including a model for the environment. The implemented model is based on the implemented snake like robot but possessed extra features. The model is controlled to perform common snake gaits for navigation. Obstacle avoidance, object (debri or victim) reaching and object dragging behaviors are acquired for the implemented gaits. Object dragging is accomplished by pushing an object by head or the body of the robot without lifting.

For effective navigation, appropriate snake gaits are conducted by the model. All control operations such as obstacle avoidance for each gait and gait selection; a network of self tunable FACL (fuzzy actor critic) fuzzy controllers is used. Although the adapted snake gaits result in the movements which have properties that are not a replica of the real snake gaits, self tunable controllers offered best available combination of gaits for all situations.

Finally, truncated version of the controller network, where the implemented mechanical robot's abilities are not breached, is attached to the mechanical robot.

Keywords: SAR Robots, FACL, Snake/Inchworm Gaits, Object Dragging

ÖZ

ENKAZ TEMİZLEYEN, YOL AÇAN VE KAZAZEDE SÜRÜKLEYEN YILAN ROBOT

ÇAĞLAV, ENGIN
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Aydan M. ERKMEN
Tez Ortak Danışman: Prof. Dr. İsmet ERKMEN
Kasım 2006, 190 sayfa

Bu tezde arama ve kurtarma faaliyetleri için kullanılmak üzere bir "yılan robot" gerçekleştirilmiştir. Yılanların referans olarak seçilmelerindeki neden ise değişik ortamlardaki üstün hareket yetenekleridir. Mekanik kısıtlamalar nedeniyle gerçekleştirilen yılan robot, gerçek yılanların birebir kopyası olamamıştır. Fakat bu tezin konusu yapay bir yılan gerçekleştirilmesi olmadığı için; gerçekleştirilen robotun, yılanların tüm özelliklerinin yerine bazı önemli özelliklerine sahip olması yeterlidir. Bu özellikler esneklik, yol tutuş, küçük ön kesit ve çevreye uyum olarak sıralanabilir.

Gerçeklenen yılan robottaki var olan mekanik kısıtları aşmak ve tasarımdan daha uç noktalarda sonuçlar almak için; gerçeklenen yılan robot referans alınarak MATLAB SIMMECHANICS'de bir model gerçekleştirilmiş ve bu modele, referansında olmayan başka özellikler de eklenmiştir. Gerçekleştirilmiş bu modele bazı yılan hareketleri adapte edilerek uygulanmış, modelin çevresel koşullara göre en uygun yılan hareketini seçmesi sağlanmıştır. Ayrıca gerçekleştirilen yılan hareketlerine “engelden sakınma” ve “tespit edilen objeye (bir enkaz parçasına veya bir kazazedeye) ulaşma”, “obje sürükleme” davranışları kazandırılmıştır. Kontrol işlemleri (engelden sakınma, ve yılan hareketi seçimi) FACL tipi öğrenme kullanarak kendi kendini ayarlayabilen bulanık mantık denetleyicilerinden oluşan bir ağ ile gerçekleştirilmektedir. Tasarım farklılıklarından ötürü adapte edilmiş yılan hareketleri, gerçek yılanların yaptığı hareketlerden farklı özellikler gösterse de, adaptif denetleyiciler her zaman en doğru hareketler silsilesini gerçekleştirip her durumda etkili seyrüsefer sağlanmaktadırlar.

Son olarak gerçeklenen denetleyici ağı, mekanik robotun kaabiliyetlerini aşmayacak şekilde sadeleştirilip, mekanik robotun kontrolü sağlanmıştır.

Anahtar Kelimeler : Arama & Kurtarma Robotları, FACL, Yılan/Solucan Hareketleri, Obje Sürükleme

To My Family

ACKNOWLEDGEMENTS

I would like to thank Aydan M. Erkmen for her support throughout the development of this thesis; without her help and supervision this thesis would never exist.

I would also like to thank Ayhan Ircı and A. Egemen Yılmaz for their help during the preparation of written materials.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1. Motivation

Search and rescue operations are commonly performed to save human life in case of disasters, accidents, terrorist attacks, etc. For search and rescue operations, time is very critical. For example, the crew of a sunken submarine or a person in a collapsed building can survive for a limited period. As time passes the probability of survival for a victim decreases rapidly. For an ideal search and rescue operation a victim has to be searched and found very quickly and then has to be saved as soon as possible in order to minimize traumas and limit casualties.

For many cases, an efficient search and rescue (SAR) operations require abilities which are beyond that of a human, thus necessitating usage of instruments. SAR instruments assist searching and rescuing. For example a thermal camera can help find a human, which is not possible with naked eye. A crane can help clear out a passage for retrieval of a person who is trapped in a collapsed building. A chopper on the either side can help both searching and rescuing.

The utilization of autonomous intelligent robots in search and rescue operations (SAR) is a rather new and challenging field of robotics, dealing with tasks in extremely hazardous and complex disaster environments. Autonomy, high mobility, robustness and modularity are critical design issues of rescue robotics, requiring the ability to learn from prior rescue experience, compliant to environmental and victim conditions. Intelligent, biologically inspired mobile robots and, in particular, serpentine mechanisms have turned out to be widely used robot types, providing effective, immediate, and reliable responses to many SAR operations.

1.2. Objectives

Biological snakes can be found at various locations on earth. Their motion modes make them superbly adaptable for a wide variety of terrains, environments and climates that they live in. Biological snakes are a result of millions of year's evolutions so it would be very advantageous to imitate these movement types and use their capability in artificial snakes. After centuries of development wheeled and walking machines are still limited in the types of terrain they can be used. A snake robot which is able to glide, slide and slither can open many applications in exploration, hazardous environments, inspection.

A snake robot is able to wriggle into confined areas and cross terrain that would show many problems for traditional wheeled or legged robots. Some of the useful features of snake robots are:

- stability,
- terrain ability,
- good traction,
- high redundancy,
- sealed mechanisms.

Robots having these properties can open up several critical applications areas in exploration, reconnaissance and inspection. The main issue in a snake – like robot is to have flexibility and being able to perform more general snake movements.

The objective of this thesis work is to implement a snake/inchworm-like robot to be used in SAR operations that will be equipped with features of its biological counterpart found in nature. The implemented snake/inchworm -like robot will be able to conduct snake/inchworm gaits up to a certain limited similarity with its biological counterpart, since the imitation of biological snakes is not in the scope of this thesis and usually infeasible due to mechanical limitations. Although the snake-like robot will not have the complete capabilities of biological snakes; capturing the main features will be very advantageous over other types of robots in most of the environments. We aim at capturing not only serpentine and/or inchworm locomotion capabilities but also add to them the object (debri or victim) dragging capability that is not intentionally found in nature for such type of locomotion. Object dragging capabilities offers very useful tasks for SAR operations such as cleaning debri from the disaster environment, or relocation of an unconscious victim in the disaster area.

A control hierarchy will be built for implemented snake/caterpillar-like robot locomotion in unknown and unstructured environments by performing adaptable snake gaits according to surface friction but also drag objects.

A SIMULINK model of the hardware robot will be realized to help visualization of complex motion on hypothetical surfaces of unstructured terrains. The SIMULINK model will possess identical dynamical and kinematical properties with the mechanical snake robot, without the physical implementation restrictions of the hardware. Thus the model can be equipped with additional gaits and abilities that the mechanical robot lacks.

1.3. Goals

The proposed snake/ inchworm-like robot is controlled to implement:

- Accordion like,
- Rectilinear,
- Sidewinding,
- Lateral undulation gaits.

These mentioned gaits are adapted to the proposed snake/ inchworm-like robot. After adaptation, each of the adapted gaits is equipped by “obstacle avoidance” and “object reaching” behaviors which enable the robot to select among these behaviors while navigating by the described gaits. The robot selects one behavior at a time.

Each of the adapted gaits has different frictional characteristics, so the robot is controlled to select the best gait due to the varying friction of environment. Navigation in an unknown environment with varying friction is solved by first selecting the appropriate gait, then conducting the selected gait with determined behavior. Gait and behavior selection is accomplished by the controller hierarchy of the implemented snake/caterpillar-like robot.

The proposed snake/ inchworm-like robot can also perform debri cleaning and victim dragging. For debri cleaning and victim dragging tasks, new dragging behaviors are generated and added to the controller hierarchy of the robot. These new behaviors are rule base triggered as the robot encounters a victim or debri. With these described behaviors, the robot drags any encountered object (victim or debri) and carries the objects along. The robot switches back to “obstacle avoidance” behavior and leaves the objects behind after sensing a nearby obstacle. By following the described strategy; the robot can collect all distributed debris to the nearby obstacles, or push a debri away from its side thus opening gaps where bigger robots or victims can pass through. Dragging scheme can also be applied to an unconscious victim for his/her/its relocation in the site where removal of the victim from the site is possible or easier.

1.4. Methodology

All control operations are based on a hierarchy of fuzzy logic controllers equipped with fuzzy actor critic learning (FACL) enabling the proposed design to adapt itself to variable conditions of unknown environments.

The control hierarchy is formed of fuzzy logic controllers connected in two layers. First layer of fuzzy logic controllers handles "obstacle avoidance" and "object reaching" behaviors of the adapted gaits. Each gait has separate controllers for "obstacle avoidance" and "object reaching" behaviors. The selection between the behaviors of a gait is done by a rule based bi-stable selection. Bi-stable switches prevent the oscillations for selection of the behaviors. The controller on the second layer selects the best applicable gait due to environmental conditions.

This flexible controller architecture is trained on the SIMULINK model of the robot to effectively solve the navigation in unknown environment problem. The training is done on the SIMULINK model because non-tuned controller network probably leads to failures or bad situations, which are usually unrecoverable for the hardware mechanical robot in real world. Since the SIMULINK model of the robot possesses identical dynamic and kinematic properties, the trained controllers should also be able to handle the control task of the physical robot or at least would be a good beginning policy and shorten training time without major failures.

In the simulations, dragging task is implemented by pushing an object via body or head of the robot model without lifting the object. Since navigating with enwrapped or grasped objects exceeds the abilities of the hardware robot due to mechanical limitations, for consistency, dragging is implemented by only pushing in the simulations. In the simulations debri and victims will only be considered as an object without any mass. Mass, inertia, shape and size properties of debris or victims are ignored and will not be simulated explicitly in this thesis. All debris and victims in an environment will be identified as an "object" to be manipulated in this thesis.

Mechanical robot lacks some features of its model, so its abilities are limited. The control of the mechanical robot is done by a truncated version of the already tuned controller hierarchy via the model. Truncation of the controller hierarchy is done according to the abilities of the mechanical robot.

1.5. Contribution

Implemented snake-like robots in the literature can be divided into two main groups. Some works try to imitate or approximate a snake body motion in specific gaits. But the hardware outcomes of these works are mechanically ineffective in actual unstructured terrain of earthquake rubbles.

Other works focus on more mechanically effective designs with actuators that are not consistent with mimicking biological snakes. But these designs called "coupled bodies" are still based on serpentine motion. These last mentioned works, which have active thrusters with active or passive joints, are also known as coupled mobility vehicles in majority remotely controlled by a human. Instead of snake gaits, coupled mobility vehicles in the literature perform custom movements due to additional actuators that do not exist in biological snakes.

In this work an autonomous snake/inchworm-like robot design is proposed as a coupled mobility vehicle but enhanced with the ability to perform snake/inchworm gaits which offer extra advantages, when confronted with variable friction surfaces. Lateral undulation, accordion, rectilinear and sidewinding are adapted and implemented in the proposed design and tested under variable friction characteristics of the terrain.

In our approach terrainability or terrain adaptation to the presence of variable friction is not only coupled to obstacle avoidance but also to routing, cleaning of debris for accessing victims in SAR operations. Cleaning of debris is done by object dragging with head or body of the robot until to a stationary obstacle. Object dragging by head or by the body while conducting appropriate snake gaits are another contributions of this thesis. A victim can also be dragged by following the same scheme with debri dragging. But the complete task of removal of a victim from the disaster area can not be accomplished by the implemented robot, since no enwrapment or grasping behaviors are implemented. The victim will be left behind when the robot encounters an obstacle like in the debri dragging case.

1.6 Outline of the Thesis

In chapter 2 previously implemented snake – like robots in the literature will be introduced. Also some main biological snake gaits will be stated. Finally chapter 2 will end with theoretical background of fuzzy logic controllers with FACL training scheme.

In chapter 3, the simulation model of the implemented hardware robot will be introduced with its adapted snake gaits. Chapter 3 also includes the controller network architecture of the simulation model. Chapter 3 ends with simulation results and discussions.

In chapter 4, implemented hardware robot structure, actuators, sensors and low level controllers are introduced. Chapter 4 also includes the hardware implementation limitations and hardware high level controller structure. Chapter 4 ends with hardware robot results and discussions.

Chapter 5 includes conclusions, future works and references.

In appendix part, the details of the microcontroller port connections, implemented noise reduction schemes and a component list will be introduced.

CHAPTER 2

LITERATURE SURVEY

2.1. Implemented Snake-like Robots in Literature

2.1.1. Hirose's Active Cord Mechanism (ACM)

In the early 1970's work of Hirose and Umetami was among the first to explore and develop limbless locomotors. Hirose was very interested in limbless locomotion so he designed and developed many limbless robots. He called his designs Active Cord Mechanisms or ACMs. Hirose focused on developing robots that could perform lateral undulation which is a type of snake movement in which the forward force is obtained by pushing the surrounding the obstacles. In his later studies, he developed some wheel coupled-mobility devices that followed from this work.

Hirose's development of modeling and control first derived expressions of force and power as functions of distance and torque along the curve described by the snake. The curve was then derived and compared with results from natural snake locomotion. The curve, termed serpenoid, has curvatures that vary sinusoidal along the length of the body axis. The curvature equations are:

$$x(s) = sJ_o(\alpha) + \frac{4l}{\pi}\sum_{m=1} \frac{(-1)^m}{2m} J_{2m}(\alpha)\sin\left(m\pi\frac{s}{l}\right)$$

$$y(s) = \frac{4l}{\pi}\sum_{m=1}^{\infty} (-1)^{m-1} \frac{J_{2m-1}(\alpha)}{2m-1}\sin\left(\frac{2m-1}{2}\pi\frac{s}{l}\right)$$

(2. 1)

This curve above is different from sinusoidal or even clothoid curves. Comparisons with natural snakes across constant friction surfaces showed close agreement between the serpenoid curve and the empirical data.

Hirose then went on to develop models for the distribution of actuator forces along the body. This was done for normal and tangential forces as well as power distribution. Again, the developed models closely correlated to muscle exertion data and force measurements from natural snake movements. [1]

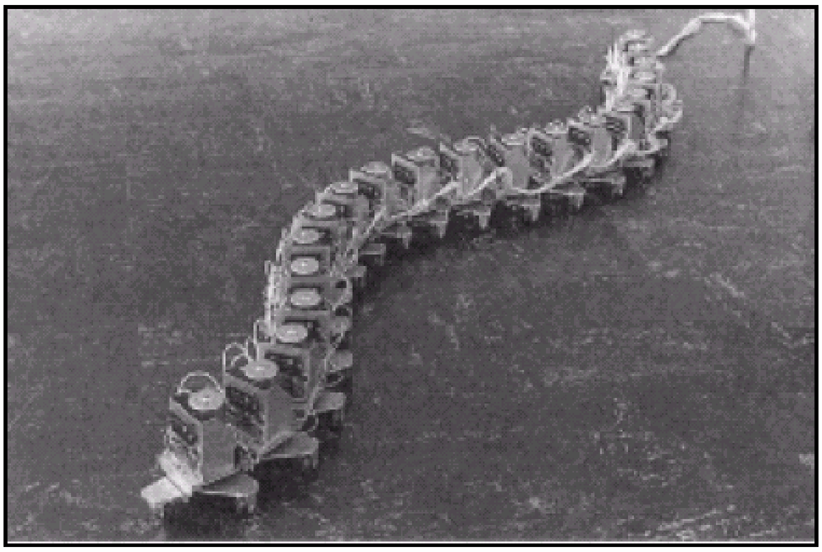


Figure 2.1. Hirose's ACM with actuated link with passive wheels.

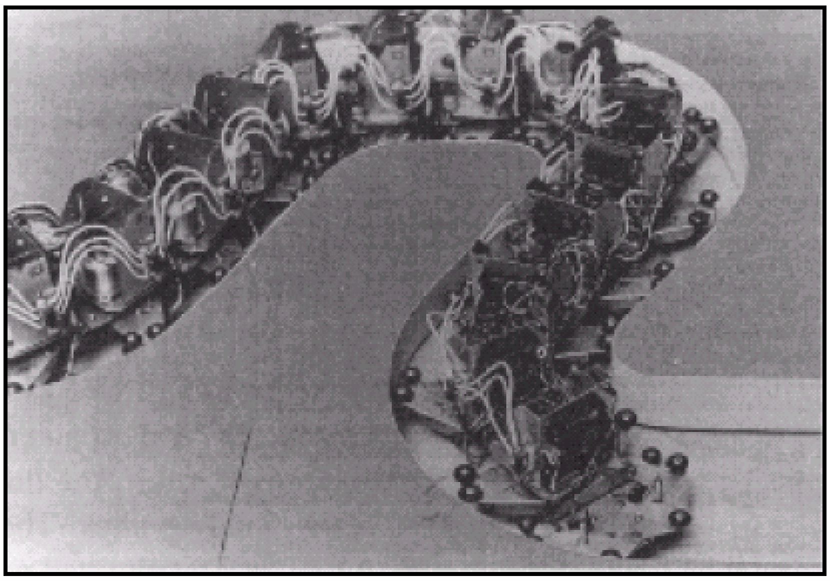


Figure 2.2. Hirose’s ACM with tactile sensing.

Hirose examined the construction of mechanisms that were able to perform lateral undulation. Several views of these machines are shown in the figure 2.1 and figure 2.2. By calculating torques, velocities and power required, Hirose was able to provide design guidelines for the actuators and drive trains. The next development was a distributed control scheme where in each link could respond independently. In Hirose’s work, the control took the form of angle commands at each joint. The variables were simply related closely to the amplitude, wavelength and velocity of the body axis. Steering of the robot was accomplished by biasing the control to adjust curvature in a section of the body.

To accommodate unknown environments required tactile sensing; this was the next step in Hirose’s work. Small contact switches provided this information to the controller. As shown in the Figure 2.2, this robot could negotiate and propel itself through winding tracks. The developments included a control technique called lateral inhibition tactile signal processing, which provided for contact and reflex motions. The shape of the body was varied according to the second derivative of the sensed contact pressure and responded appropriately to provide forward progress.

All of Hirose's locomotors used either powered wheels or passive casters and the only locomotion mode studied was lateral undulation. Hirose and his colleagues have gone on to develop an elastic elephant-like trunk, a large serpentine mechanism for interior inspection of turbines and small manipulators for surgical applications. It succinctly covers many years of development in serpentine mechanisms. Hirose's work in serpentine robots is probably the most complete of all work in this area. He dealt with issues of mechanism, control, sensing and modeling of natural animals. However, the mechanisms used wheels, the terrains for the ACM's were 2D only, and the mechanism used only lateral undulation as the locomotion mode. The configuration, while not practical for application use, was a great advance in serpentine robots.

2.1.2. Masashi Saito, Masakazu Fukaya, and Tetsuya Iwasaki's Serpentine Locomotor

Masashi Saito, Masakazu Fukaya, and Tetsuya Iwasaki as analyzed lateral undulation locomotion in more detail with formulating the steering. They also built a robot which emphasizes their work. Their design did not use passive wheels for generating high tangential friction with low normal friction (for lateral undulation) like Hirose's ACM, increasing ability in rough environments. [2]

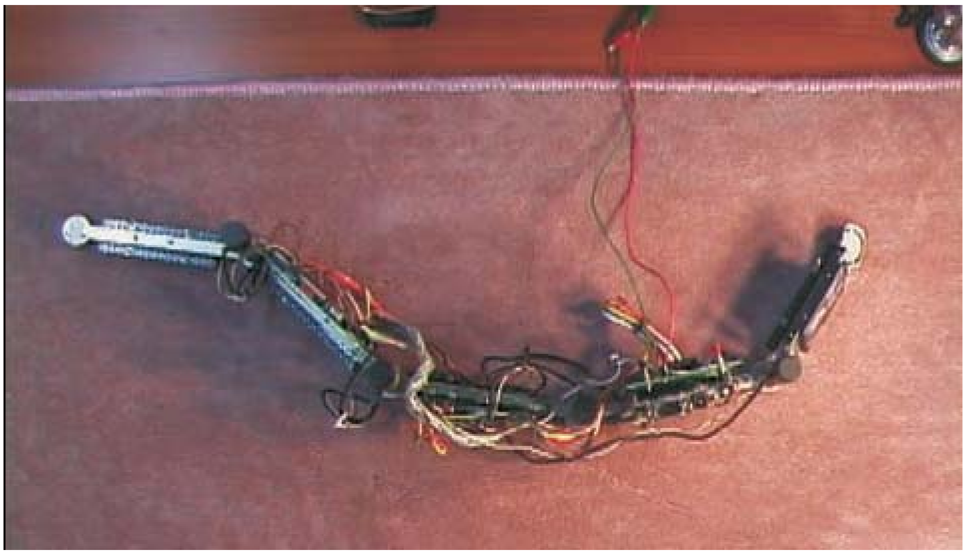


Figure 2.3. Top view of the robot.

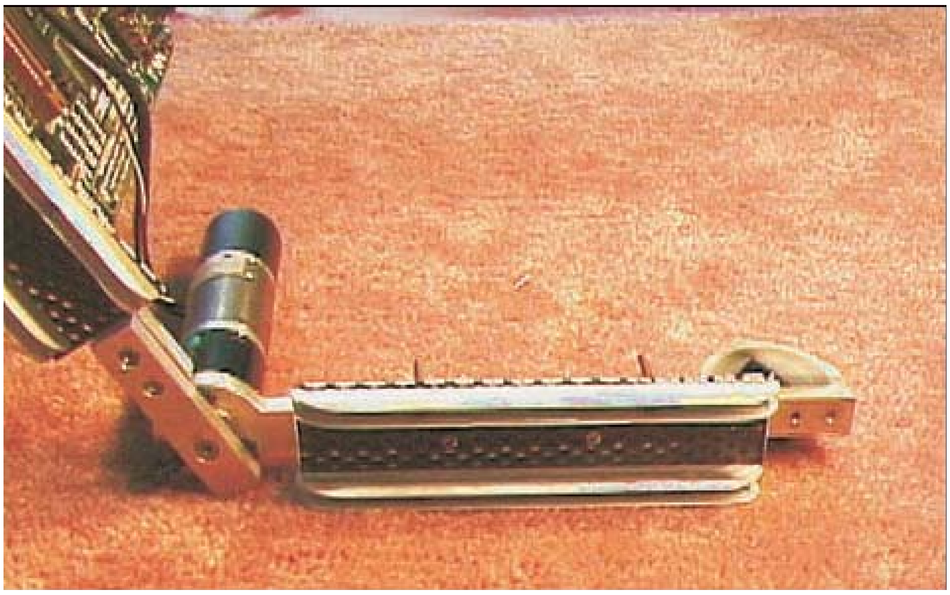


Figure 2.4. Belly of the Robot.

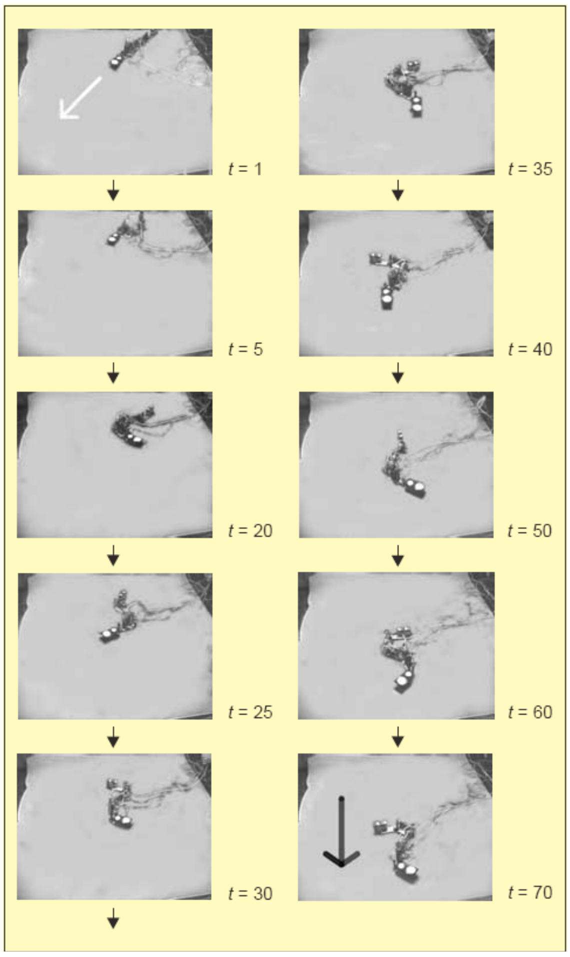


Figure 2.5. Motion of the Robot.

2.1.3. Karl Paap's GMD

Karl Paap and his group at GMD (German National Research Center) developed a snake-like device to demonstrate concepts and developments for real-time control. The device is a tensor device that uses short sections with cable winding mechanisms to control curvatures along several segments (figure 2.6). This snake robot is able to perform accordion movement. [3]

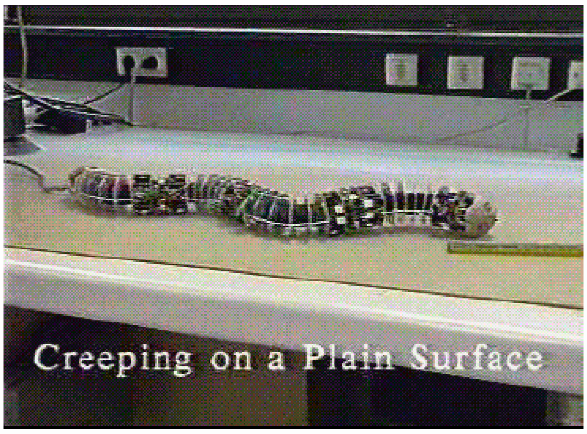


Figure 2.6. GMD robot.

The GMD snake was designed to move in different surroundings as similar as to real snakes without any wheels or legs. It consists of very flexible rubber joints which are controlled individually. GMD robot is able to perform movements that other implemented snake robots are unable to perform. [3]

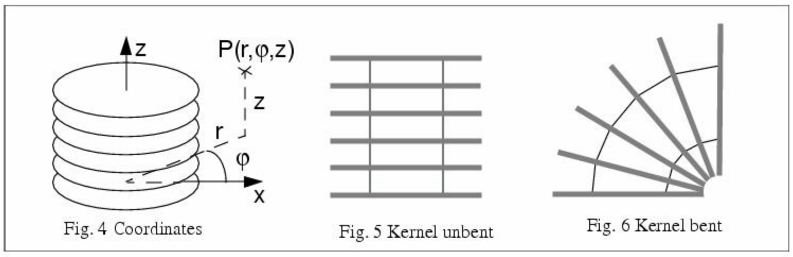


Figure 2.7. Flexible Joints

GMD robot's hardware is consisted of three main parts:

- Head
- Body sections
- Tail

Some sensors are located at the head section; four of these sensors are LDRs (light detecting resistors) and a touch sensor.

The sections in the body are identical. Each section has two joints. The joints are made of orthogonal aluminum plates which are connected by rubber pieces.

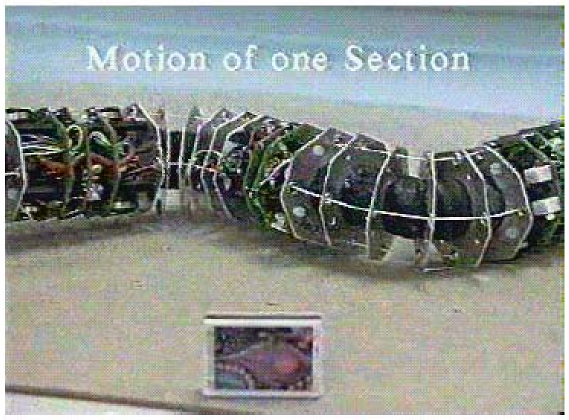


Figure 2.8. Body section of GMD robot

There are two drivers in each section which can bed the joint vertically and horizontally. These drivers are the main actuators of the GMD snake. For good manipulation these drivers should be properly monitored and controlled. For control the drivers can be monitored by:

- Measuring elapsed time.
- Counting the number of rotations of the driver.
- Measuring the angle between the innermost and the outermost plates.

Measuring the angle between the innermost and the outermost plates is the most effective monitoring type. But in GMD snake the monitoring is done by counting the numbers of rotations of the driver. For counting; each driver is coupled with a pair of reed contacts to count the rotation. These reed contacts are the main sensors for monitoring the snake’s position; and the information they deliver is central part of motion control. The motion is obtained by adjusted and synchronized bending at all sections.

2.1.4. Jörg Conradt, Paulina Varshavskaya’s Snake Robot

Jörg Conradt and Paulina Varshavskaya have built a biologically and neural inspired autonomous mobile robotic worm. The main aim was controlling a large D.O.F. and obtaining an elegant motion by using a simple neural system. The most important property of their project is that while closely imitating neural control of the lamprey, achieving a level of modularity. This snake robot performs side pushing. [4]

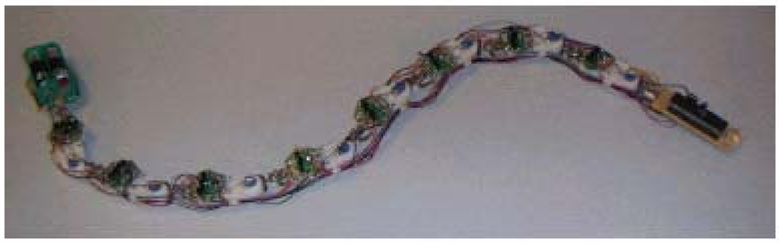


Figure 2.9. First Version of WormBot.

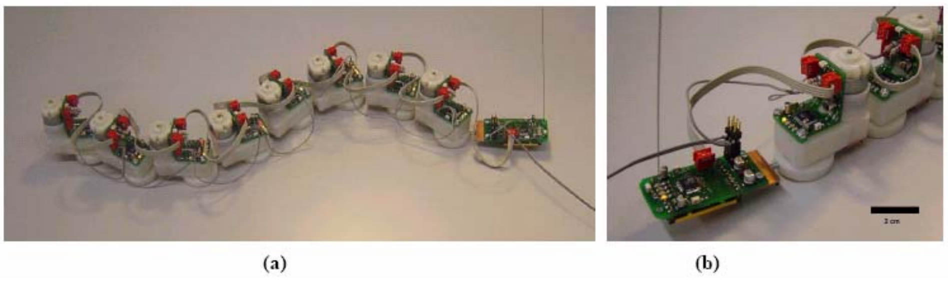


Figure 2.10. Version 2 of the WormBot. Modules are planar but exhibit true distributed control with an individual microcontroller on each segment. (b) Close-up view of the head and the first segment. The black bar in the image corresponds to 2cm.

The robot has segmented design (figure 2.9, 2.10). Each segment is identical except the head and the tail. The batteries are located at the tail. At the head a microcontroller (atmel mega163) is installed which drives eight CPGs. (Control Pattern Generator) Each segment can rotate their neighbors.

The second version of the design has a microcontroller at each segment. The motor of the joints are controlled by PWM signals. There are light and heat sensors located at the head. A two wired communication interface is built through the snake-robot. The head has a wireless connection to a PC. Each motor is controlled by CPG oscillators coupled with each other with the following relationship: [4]

$$\dot{\theta}_i = \omega_i + \sum_{j=1}^{N} \alpha_{ij} \sin\left(\theta_j - \theta_i - (i-j)\cdot\phi\right) \tag{2. 2}$$

Where:

ω_i : The frequency of the ith oscillator,

α_{ij} : Coupling coefficient between segments,

θ_i : The state of ith oscillator,

ϕ : The desired phase shift between neighbor oscillators.

For simplicity α_{ij} coefficient is set to zero for non-neighbor segments.
With this coupling algorithm they have successfully obtained highly accurate motion in both prototypes. But with lack of tangential forces and special skin which has different friction for different directions (like snakes have), the forward motion was poor. With appropriate skinning forward motion can be obtained.

Inoue, Ma and Jin [5] have followed a similar technique with Conradt [4] where they develop a neural oscillator network (NON) for their implemented snake-like robot. They control the yaw axis of their planar snake-like robot with their NON and observed the result for different phase shifts of the oscillators in the network.

2.1.5. Karl Paap's GMD2

With the motivation of the first GMD robot GMD robot 2 was constructed by Paap and his group. The first GMD robot had flexible rubber joints. These joints caused uncontrolled torsion effect which occurred when the snake lifted some part of its parts. From this experience the second generation GMD snake was built with universal (or cardanic) joints. [6]

The second generation GMD robot uses the serpentine locomotion. In serpentine locomotion the body follows the trajectory of the head. This is a very common way of snake movement. And this can only be done by propagating the stimulation from head to tail. This observation yielded to this procedure:

- The head motion is arbitrary determined. This can be done by an operator or head can follow a path to avoid obstacles.
- Each segment determines their position on the trajectory and executes the appropriate motion which is stored by the head. (The head stored the movement for the segment to perform it)

The GMD2 snake has five identical body segments (figure 2.11). There are two ultra sonic sensors and a video camera located at the head such that the scene can be monitored remotely. The batteries are at the tail section.

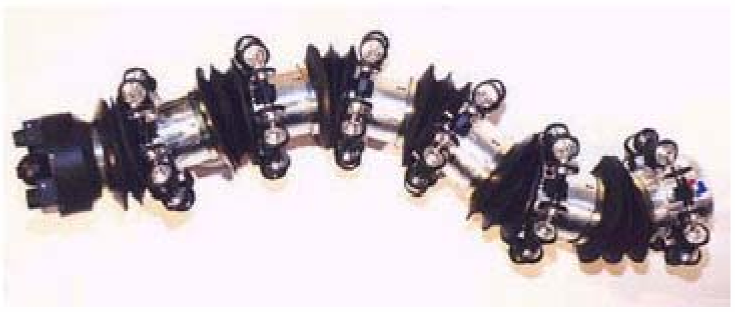


Figure 2.11. GMD 2.

A microcontroller is located at each body segment. These microcontrollers drive the local sensors and controls each joints. At each joint there are three 5w dc motors to control the universal joint. The microcontrollers are communicating with each other via a CAN bus.

Many numerous types of movement performed by snakes was observed and it is found that one essential technique was not implemented by snake-like robots was the forward motion done by thousands of active scales under the snake's body. To realize an equivalent movement a ring of wheels is attached to each joint. These wheels are driven by a separate dc motor so each segment can control their forward force.

The motion control is done with the following procedure. For a joint segment length "L" and wheel speed "v"; the actions are performed at discrete times given by t0, t1, t1… .The head chooses the actions arbitrary or chooses them to avoid obstacles. Then the body segments perform these motions with a certain delay. For he wheel's velocity, the delay is zero because the body moves with the same velocity. [6]

The section "i" has to execute the head's current action when it has traveled "i*L" length, but meanwhile the speed can be changed causing the actions being executed with a constant time delay. These delays must be modulated with speed. This task can be accomplished by a large shift register. So each segment completes their actions at determined time divisions and sends their completed action to back, while receiving new ones from front. This method is very close to biological snake's neural system.

GMD2 snake is very flexible and efficient. It combines the advantages of the serpentine movement and the wheeled movement. It can easily manipulate places that are not possible for man or other vehicles to reach.

2.1.6. Hirose's Genbu

The mobile robot Genbu is implemented by Hirose and his students. Since 1972 Hirose proposed the Active Cord Mechanism (ACM) and built many robots based with this mechanism. Hirose's previous robots usually consisted of passive wheel active joint robots. But Genbu is implemented in a different manner that it is has active wheels with passive joints (figure2.12). By using passive joints, the robot can change its posture quickly according to terrain. Moreover, it is resistant to shocks from rough terrain because passive joints have no vulnerable components such as gear head motor. [7]

Having passive joints has also same disadvantages. It is not possible pass over wide gaps with passive joints and since joints are passive the control of the robot is completely implemented only by the control of the active wheels which is a challenging task. The high level control is done by a human operator remotely.

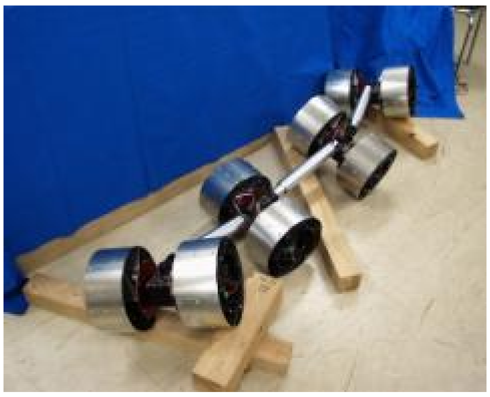


Figure 2.12. Genbu.

Genbu is loaded by the motor driver and battery in each wheel, and micro controller in each body. Pitch, roll and yaw angles are measured by using the new rigid body joint arm mechanism. Stable torque measuring system is accomplished with float differential mechanism in each wheel. In this robot system, adaptive control for the terrain is possible by using posture of the multi-wheeled robots. For practical use, this robot is waterproofed by using X-ring in each joint.

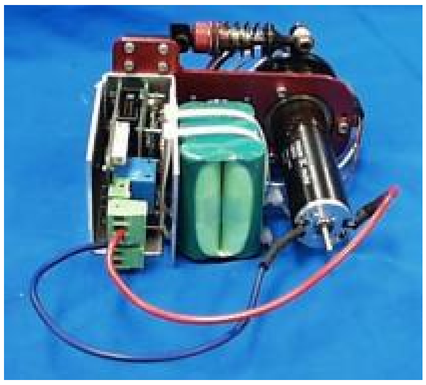


Figure 2.13. Inside of a wheel (Motor, gear head and batteries are included in a wheel system.).

2.1.7. Kohga by Kamegawa, Yamasaki, Igarashi, Matsuno

The snake-like robot Kohga is constructed by connecting multiple crawler vehicles serially with passive joints. Kohga can also be classified by passive joint active wheel robot like Genbu. Kohga is a rescue robot built especially to investigate collapsed buildings. This robot is driven by a human operator remotely. [8]

The main body joints are completely passive and have three degree of freedom. Kohga has 2 DOF active joints only at its tail and head crawlers. Two cameras are attached to the tail and head. By using active joints the view of cameras can be controlled. (The tail joint is raised like a scorpion tail giving the back camera a good view of scene with the robot as shown on figure 2.16.)

The robot is equipped with various sensors such as:

- Two CCD cameras attached at both ends of robot. Images are transmitted to the operator by a 1.2 Ghz transmitter.
- IR distance sensors positioned to five directions (Front, top, bottom, right, left directions) for the first and last crawlers; two directions for (right, left) middle crawlers.
- Potentiometer equipped joints. (For obtaining joint positions)

The operator of Kohga give commands corresponds to the first crawler and the lagging crawlers follow the trajectory of the head which is main principle of lateral undulation. Practically by this approach the robot can pass every narrow space that its head can pass.

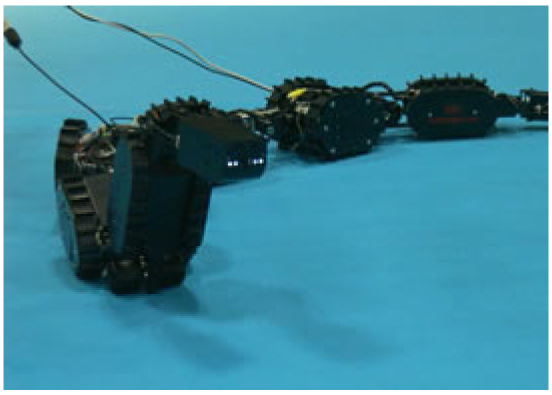


Figure 2.14. Kohga frontal view.

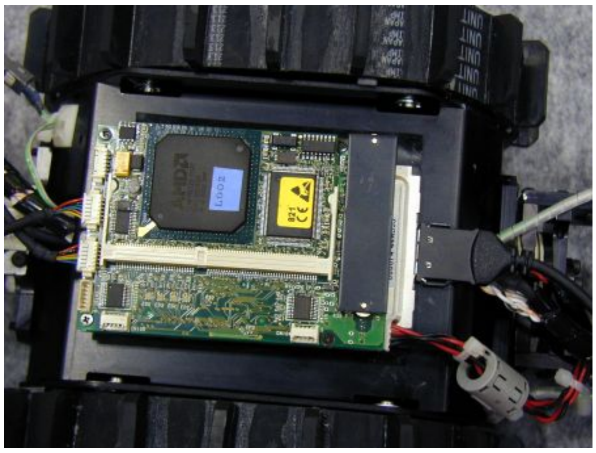


Figure 2.15. One segment of Kohga (Each segment includes a controller.).

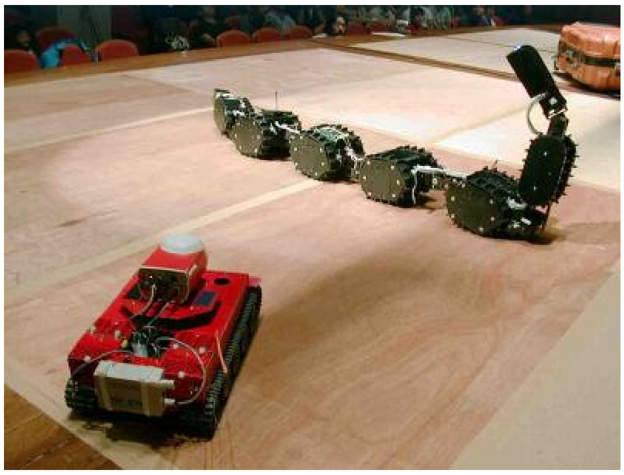


Figure 2.16. Kohga. CCD cameras are present in the tail and head. The tail and head joints are actively driven whereas the joints between the body segments are totally passive. But Kohga has sensors on passive joints to sense the body posture.

2.2. Object Dragging and Lasso Type Grasping or Enwrapment

Barış Atakan, Aydan Erkmen and İsmet Erkmen [9] solved the kinematic problem of 3D lasso type grasping and dragging while performing serpentine locomotion. The snake body model which they used in their work was a 3-dimentinal snake-like robot formed by serially connected links. Each link in their model had two degree of freedom rotated around local yaw and pitch axes while some joint also had a pair of passive wheels. The work of Barış Atakan, Aydan Erkmen and İsmet Erkmen also dealt with singular configurations, which is a big issue for hyper-redundant robots.

Evrim Onur Arı, Aydan Erkmen and İsmet Erkmen have worked on developing a flexible controller structure of a 2D grasping snake-like robot [10]. The robot model they used in their work was consisted of bodies interconnected by joints rotating on local yaw axis. Later they improved their work to 3D grasping with adding each joint the ability to rotate on local pitch axis [11]. The kinematic problem of locomotion of such body was solved by Burdic [12]. Their flexible controller structure was composed of fuzzy logic controllers utilizing FACL. They had implemented three behaviors of “target reaching”, “obstacle avoidance" and "object grasping". Each segment had one controller for each behavior and can select a behavior at a time independently.

2.3. Snake Gaits

2.3.1. Accordion

This type of gait is used in narrow and flat places. For this type of locomotion the snake folds and unfolds successively, while the whole body is touching the ground. The friction property of the snake skin, which shows high friction to the movements to backwards and low friction to the forward direction, converts the accordion movement to a forward motion. [18]

2.3.2. Rectilinear

For this type of gait the snake lift a part of its body at tail; and moves the lifted section along its body like a traveling wave. In fact the moving thing is not the lifted section but the lifted attribute and this attribute move along the snake like a traveling wave [18]. In other words, the snake simply fixes several body points by touching the ground and moves the part of the body in between the fixed points [19]. This gait is applicable because it requires less energy and can be used when there is no urgent situation for the snake.

2.3.3. Sidewinding

This gait is used on soft ground like sand. For this gait, snake only contacts the ground on two points which is very useful when the ground is very hot. Sidewinding is conducted by the snakes which live in deserts. [18]

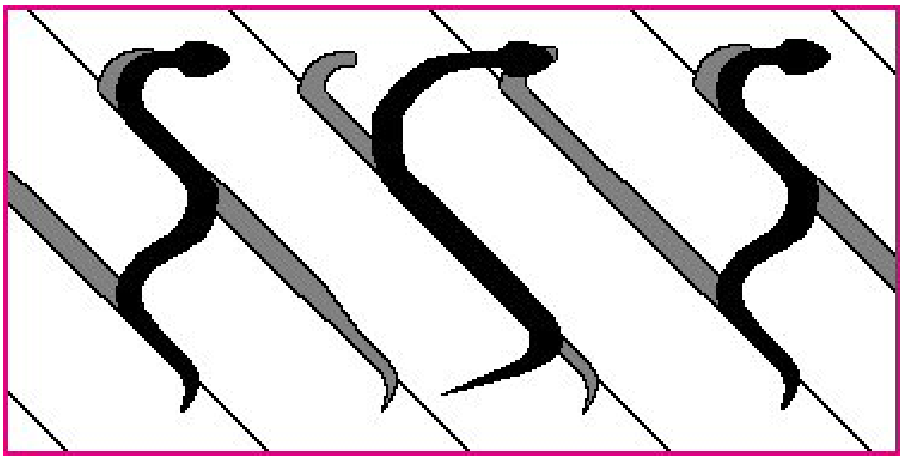


Figure 1.17. Side Winding [http://www.worldwidesnakes.com/anatomy/sidewinding1.jpg]

The snake touches the ground at only two points and then moves the touch point like a wave on its body to its side. At the right time snake touches the ground at new point. The direction of the movement has also a component on perpendicular direction of the head (figure 1.17). By this gait a snake can reach three kilometers per hour.

2.3.4. Lateral Undulation (Serpentine Locomotion)

Lateral undulation is most commonly used gait; almost all kind of snakes perform it. The body of snake moves in waves of muscular contraction from head to tail. The forward force is generated with the different friction coefficients of the snake body in the direction of tangent and the normal directions (figure 1.18). With this gait the snakes can also use stationary obstacles to help forward motion. There is no static contact. The most important feature of this gait is that the body of the snake follows the trajectory of the head. The characteristics of lateral undulation make it efficient in bumped grounds for long snakes. [18]

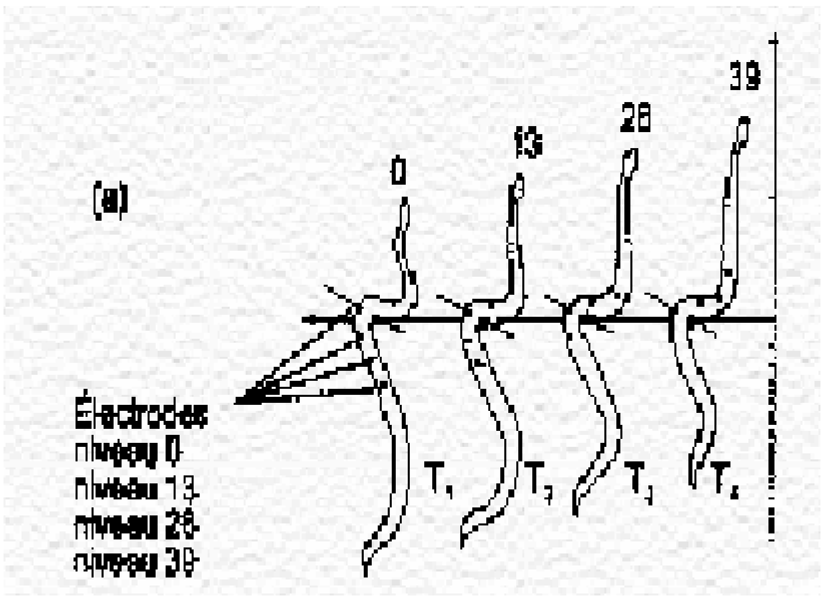


Figure 1.18. Lateral Undulation [http://chabin.laurent.free.fr/sn4.gif]

2.4. Theoretical Background

The implemented snake robot and its model use fuzzy logic controllers with fuzzy actor critic learning (FACL). Fuzzy actor critic learning is a solution of "reinforcement learning problem". There many alternative former techniques to implement a self-tuning fuzzy logic controller such as [13] and [14] in the literature. But usually only a reinforcement is available in real life applications for tuning. Self tunable fuzzy logic controller of [13] requires the knowledge of the desired trajectory of the system which is usually unknown, thus it is not utilized in this thesis. Due to the ease of implementation, Jouffe's actor critic learning scheme [16] preferred rather than GARIC architecture of [14]. In the preceding section, the mathematical background of "reinforcement learning problem" and FACL will be introduced.

2.4.1. Reinforcement Learning

Learning can take place with an absence of a direct teacher. In nature such learning always takes place. An infant can learn the consequence of actions in order to achieve a goal just by interaction with the environment. Or an infant can learn not to do certain actions for such situations he/she faces.

"Reinforcement learning" is learning what to do in a specific situation; in other words mapping actions to situations to maximize a reinforcement which is obtained by interaction with the environment. Reinforcement does not specify the correct actions, but it represents the goodness or badness of what have been done. For "reinforcement learning", the learner has to discover what to do for all possible situations by a "trial and error" method to maximize the reinforcement. While the reinforcement increases, the leaner learns to achieve the goal. The control problem is in fact doing the correct actions in all possible situations so a reinforcement learner is a controller. [15]

"Reinforcement learning" attacks the learning problems by dividing the situations in the environment to discrete states and developing an experience by performing discrete set of actions for all possible states. Reinforcement learning works in discrete time; it has discrete set of actions, discrete states but can handle continuous learning problems.

One important feature of "reinforcement learning" is that the current taken actions of the learner not only effects the immediate reinforcement but also it effects the reinforcement which will be received over time. This is because in practice there is a strong relationship between the subsequent actions; only one action's affect can be considered alone. This concept is called "delayed reward". "Delayed reward" and "trial & error" are the most important features of reinforcement learning. [15]

Reinforcement learning is different than supervised learning. Supervised learning requires an external supervisor which supplies the correct actions for all situations in which the learner might be in. But for practical applications it is impractical or sometimes impossible to obtain correct actions for all possible situations. Usually the only thing the learner has is a priory experience and an interaction with environment for learning. The learner has to learn to achieve the goal by this two means.

The main problem which arises for "reinforcement learning" is the trade off between exploration and exploitation. Exploration is the learning search for seeking better actions for maximizing the received reinforcement; but exploration may result in failure. For exploitation the learner follows its experience for taking an action, where in its experience the corresponding action has caused a high reinforcement, but this reinforcement is not guarantied to be the highest one. While the learner exploits from its current knowledge, if the current knowledge is good enough it will not fail as frequently as in exploration. But during exploration the learner missed actions which cause higher reinforcements. Neither exploration nor exploitation can be conducted without a failure. For reinforcement learning the learner somehow has to balance the exploration and exploitation. [15]

A reinforcement learning system has four main elements:

- Policy
- Reward function
- Value function
- Model of environment

Policy defines the learner's behavior for the all possible situations which the learner might be in. As the learner learns, the policy might change in time. Policy can take any from a look up table to a stochastic process.

Reward function maps the visited states of the environment, (situations of the environment) to a scalar reward which represents desirability of the current state. Reward is the numerical representation of the reinforcement. The learner can calculate the current reward by using the reward function.

Reward function defines the goodness and the badness of a state, so it can be used to alter the policy so better states can be visited. For example if the policy that the learner is following leads the learner to states which have low rewards, then the policy can be modified so in the future the leaner will not visit the low rewarded states again.

It should be noted that by the “delayed reward” concept the current reward might be a result of not only the last action, but the consequence of actions that the learner have performed. So it is wise to modify not only the current states policy, but also the policies of visited states which the learner has learner visited to reach its current state.

The value function defines the goodness of a state in long term. This value function is needed because the state goodness can not be defined only with the immediate reward. A state can have a low immediate reward, but after visiting that state the learner might experience states with high rewards in future which mean that the state is in fact a good state. Vice versa the immediate reward might be high but that state leads to states with bad reward in future which means the state is bad.

The learner’s aim is not to maximize the immediate reward, the aim is to maximize the total reward which be received over time. The value function defines the state goodness over long time, so the value function should be used for modifying the policy instead of the immediate reward. Unfortunately the value function is not as straightforward as the reward function. The reward function can be directly extracted from the environment. Usually there is no way to know or calculate the value function, but it can be estimated by calculating the received rewards after visiting that state. The value of a state changes as the policy changes. So during the learning process where the policy is modified due to the value function at that time, the value function has to be estimated from the observations of the learner for the new policy. Reinforcement learning problem is modifying the policy according to the current estimate of the value function and at the same time re estimating the value function for the modified policy. The value function can be in various forms which will be induced later.

In reinforcement learning, the learner consists of an action (or policy) modifier & state evaluator and an action performer. The composition of these two components will be referred as “agent”.

There are techniques which can solve reinforcement learning problems without estimating a value function. The most known of this type learning is the genetic algorithms. This type of learning method directly searches the policy space or finding a more skilled agent. This technique is analogous to the biological evolution where evolution produces more skilled agents as well as less skilled agents but only more skilled agents survive making the specie more skilled. In practice if the policy space is small or there are many good policies (or both) genetic algorithms works effectively or else it would take too much time to learn. These techniques are also very effective when the learner cannot sense its state in the environment accurately. Besides genetics algorithm’s advantages, they ignore most of the useful structure of the reinforcement. The evolutionary methods are more like producing agents which better suits the environment instead of performing learning operation for individual behaviors with interactions thus genetic algorithms and evolution can not be considered as learning. In nature both the evolution and reinforcement learning takes place at he same time. A biological being can learn to accomplish many tasks in its life time, but evolution takes many life time of a biological being to create more skilled specie. [15]

The last element of reinforcement learning is the model of environment. The model mimics the behavior of the environment so using the model; the learner can foresee the next states thus next rewards corresponding to all applicable actions in a specific state. If the model of the environment is completely known, the agent can utilize a DP (dynamic programming) technique to plan its next consequence of actions to be performed to maximize the incoming reward in long run; no experience is needed. But needlessly to say the model of the environment can not be known precisely for most of the cases. DP techniques can still be applicable for such cases, where the agent first approximates the model of the environment then utilizes the DP techniques over the approximated model. Also there methods which do need any model, these methods will be discussed in detail at section 2.4.9.

2.4.2. Action Selection

For the "reinforcement learning" problem the learner has to have a policy for action selection for all possible states. The most trivial action selection method is to use an "action quality" function for all actions which will be donated as $Q\ (a)$ where "*a*" stands for action which is any element of all actions. The agent holds separate $Q\ (a)$ for all states. The optimum action quality which the agent is seeking for will be donated by $Q^*(a)$ which is unknown but can be approximated. For now consider the case where the agent only faces one sate. The agent can approximate $Q^*(a)$ function by averaging the rewards after performing action "*a*".

$$Q_t\ (a) = \frac{r_1 + r_2 + r_3 + ... + r_{k_a}}{k_a} \tag{2. 1}$$

Subscript t is used for $Q\ (a)$ because $Q\ (a)$ changes as the new rewards are averaged. For the initial case $Q_t\ (a)$ should be started arbitrary. As the k_a approaches to infinity $Q_t\ (a)$ converges to $Q^*(a)$.

The most trivial action selection method is to select the action which has the highest quality. This method always exploits the current knowledge; but does not seek for a better alternate action. This method is also called "greedy action selection" where the agent tries to maximize the immediate reward. But as mentioned earlier the agent needs to improve long term reward, so it has to do some exploration. A better thing to do is to exploit the current knowledge (select the action which has the highest quality) but sometime with a small probability ε explore actions in random. This method is called "ε greedy action selection". When time goes to infinity; this method guaranties that every action is performed. Since all actions will be tried, $Q_t(a)$ will converge to $Q^*(a)$ when infinite time steps passes. But even the optimum quality function $Q^*(a)$ is reached, the probability of selecting the correct actions will be $1-\varepsilon$.

The classical ε greedy action selection has a drawback because it explores each action with equal probabilities. This means during the exploration the agent can also select worse actions which may be undesirable for the tasks where selecting bad actions can not be tolerated. A better way to explore among all actions is to weight the probability of each action by their qualities.

$$p_t(a) = \frac{Q_{t-1}(a)}{\sum\limits_{b \in all\_actions} Q_{t-1}(b)} \tag{2. 2}$$

Selection of an action can be done by a random process depending on the probabilities of the actions. Note that the action selection is done with the last time instant's (reinforcement learning works in discrete time.) action qualities. After the action is selected, the agent has to calculate the current action quality $Q_t(a)$ which is done by (2. 1).

Calculating the average of rewards by (2. 1) is not memory efficient because the agent has to memorize all past rewards, as the time goes to infinity such task require infinite memory. Implementing a sliding average will solve this problem.

$$\begin{aligned} Q_{t+1}(a) &= \frac{1}{t+1}\sum_{i=1}^{t+1} r_i = \frac{1}{t+1}\left[r_{t+1} + \sum_{i=1}^{t} r_i\right] = \frac{1}{t+1}\left[r_{t+1} + tQ_t\right] \\ &= \frac{1}{t+1}\left[r_{t+1} + tQ_t + Q_t - Q_t\right] = Q_t + \frac{1}{t+1}\left[r_{t+1} - Q_t\right] \end{aligned} \tag{2. 3}$$

The equation above is equivalent to the average. Note that the effects of rewards decreases as time increases. The new update method can be interpreted as follows:

$$new\ Estimate = old\ Estimate + step\_size\left[t\arg et\ \text{-}\ old\ Estimate\right] \tag{2. 4}$$

Update mechanisms that are dealt so far only considered one situation, state. But if the environment is changing over time, it is wiser to give recent rewards more weights. One way to do so is to use a fixed step size.

$$Q_{t+1} = Q_t + \alpha\left[r_{t+1} - Q_t\right] \tag{2. 5}$$

Where α is a fixed step size which is $0 < \alpha \leq 1$. With this scheme the learner can also adapt itself to the slowly changing environments.

2.4.2.1. Reinforcement Comparison

Determining if a reward is big or small is an important problem in reinforcement learning. An update mechanism with “reinforcement comparison” can be utilized in order to avoid this problem. The reinforcement comparison method does not maintain a Quality value for the actions but an overall level for the received reward for comparison. This is useful to determine if the reward is low or high. Let $\pi_t(a)$ be the preference of the action “a” for time “t”.

$$P_t(a) = \frac{P_{t-1}(a)}{\sum_{b \in alla\_actions} P_{t-1}(b)} \tag{2. 6}$$

After each play, the preference should be updated as:

$$P_{t+1}(a) = P_t(a) + \beta(r_t - \bar{r}_t) \tag{2. 7}$$

The term $\bar{r}_t$ stands for the reference reward and β is the update step. ($0 < \beta \leq 1$) This update encourages the preference of the selected action if the reward is higher than the reference reward.

Also the reference reward should be updated at each time step.

$$\bar{r}_{t+1}(a) = \bar{r}_t(a) + \alpha(r_t - \bar{r}_t); \qquad 0 < \alpha \leq 1 \tag{2. 8}$$

2.4.2.2. Pursuit Methods

Pursuit methods have both action value estimates and action preferences. Preferences always pursuits the action which is greedy according to the current action value estimate.

Just before the selecting the greedy action, the probability of selecting that action is enforced. Let a_t^* be the greedy action

$$\pi_t(a_t^*) = \pi_{t-1}(a_t^*) + \beta(1 - \pi_{t-1}(a_t^*)) \qquad 0 < \beta \leq 1 \tag{2. 9}$$

$$\pi_t(a_t) = \pi_{t-1}(a_t) + \beta\left(0 - \pi_{t-1}(a_t)\right) \qquad \text{For all } a \neq a_t^* \qquad (2.10)$$

The selected action's preference is increase to 1 and the preferences of other actions are decreased to zero.

2.4.3. Agent – Environment Interface

The most basic concept of the reinforcement learning is to learn from interaction to achieve a goal. The learner and the decision maker are called the agent, and everything outside the learner is environment. The agent selects new actions and the environment responds the actions by representing new states to the agent. The environment also represents rewards to the agent.

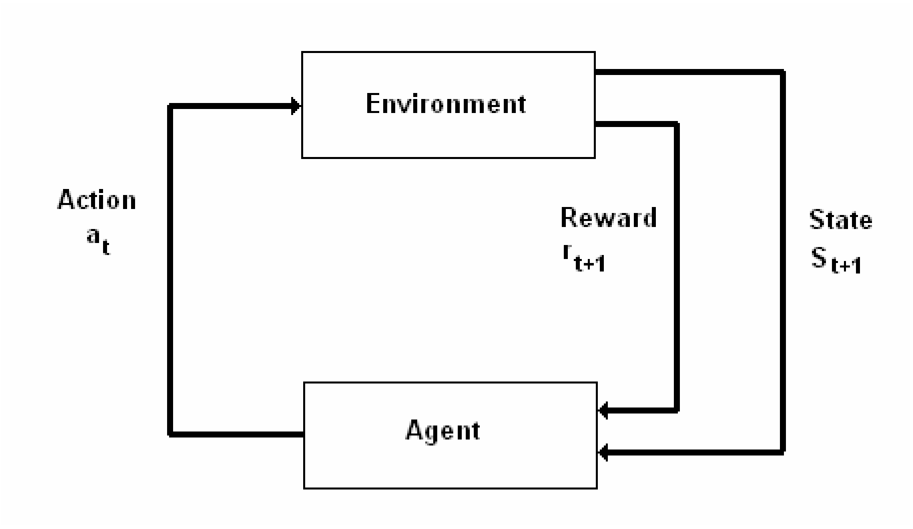


Figure 2.19. Typical Environment & Agent Interface

The agent takes action "*a*" at time "*t*" and the environment responds to that action with new state and reward at *t+1* time instant according to the last state. The 2.1 section had focused on the problems which had stationary states, dealing with only one state, but for the general reinforcement learning problem the agent may face many states, so it has to associate actions with each state, in other means the agent has to learn a policy. In the figure 2.17, the action taken by the agent with the last state forms the new state and new state creates the new reward. The system in the figure 2.19 may address the full reinforcement learning problem.

At each time step the learner uses its policy to select what actions to apply. The policy will be denoted as $\pi_t(s,a)$ which hold the probabilities of selecting the action "*a*" at state "*s*". The $\pi_t(s,a)$ term depends on "*t*" because during the reinforcement learning it will be tuned to reach the optimal policy $\pi^*(s,a)$ which produces the maximum reward at long run.

2.4.4. Goals and Rewards

As mentioned before, the purpose of the reinforcement leaner is to maximize the total reward in long run. So far a reinforcement learning, the reward determines the goal. The agent should be supplied with a reward signal such that; when the learner maximizes the reward in long time period, the leaner reaches the goal. This type of may seem very indirect and limiting, but in fact it is very flexible and applicable.

Suppose the goal of a robot is to go to a specific point but on the way it must avoid obstacles. During the robot run in the environment the reward should be kept 0, when it reaches the goal without colliding the reward can be set to *+1*. If it moves away from the goal point a reward of punishment) *-1* should be supplied.

One important property of reward signal is that it has to be prepared to achieve the main goals; not sub goals. For the reaching goal point without colliding example if the reinforcement is set to sub goals of avoiding the obstacles and getting near to the goal point is not suitable. Because the agent can find a way to both avoid obstacles while getting near to the goal point but without ever reaching the goal point. This is analogous to taking opponent's pieces in expense of losing the game. The rewards should always be set for the mail goal, not how the main goal is achieved.

2.4.5. Return

The reward which will be received in long run is called "return" and will be donated by "*R*". The agent's aim is to maximize the return. In the simplest way return can be calculated by summing all the rewards.

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + ... + r_T \qquad (2.\ 11)$$

The R_t term is the total return after time instant "t". The term r_T stands for the reward while reaching the terminal state. Terminal state is a terminating state such that it captures the agent, so the agent can not leave that state. And after the terminal state is reached, the reward will be zero. Only "episodic tasks" have terminal state where the task ends at some specific sates. Blackjack card game can be an example of episodic tasks. If a player reaches 21 the game ends and the player wins, if a player exceeds 21 again the game ends but this time player loses. At both cases the game has to be restarted.

However most of the time tasks are not episodic, they can not be broken into subsequences. Such common tasks are called "continuous tasks". For continuous tasks rewards until infinity should be summed; but doing so produces infinite return. To avoid infinite return, recent rewards are weighted in the sum.

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \qquad 0 < \gamma \leq 1 \qquad (2.\ 12)$$

The γ term stands for the discount rate. If γ approaches *0*, the return will be equal to immediate reward, if γ approaches *1* the return will be very far sighted.

2.4.6. Value Functions

All reinforcement learning algorithms use a value function for states (or state-action pairs) to estimate how good it is to be in a given state (or how good it is to perform an action at a given state). "How good" measure is defined in terms of the expected future return since the actual future return can not be known. All value functions depend on the policy which is followed by the agent. The quality of a state is:

$$V^{\pi}(s) = \mathrm{E}_{\pi}\{R_t | S_t = s\} = \mathrm{E}_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | S_t = s\right\} \qquad (2.\ 13)$$

"E" stands for expected value. The quality of a state – action is:

$$Q^{\pi}(s,a) = \mathrm{E}_{\pi}\{R_t | S_t = s, a_t = a\} = \mathrm{E}_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | S_t = s, a_t = a\right\} \qquad (2.\ 14)$$

Both V^{π} and Q^{π} are depended on the future rewards which are not available to the agent at time "t". Fortunately both V^{π} and Q^{π} functions can be estimated using experience. The agent can keep the averages of the rewards after visiting each state; as time instances reaches infinity the averages will converge to the actual state qualities. Likewise for state – action quality, the agent can keep averages of rewards after performing each action at a state. Again when the number of time instances reaches to infinity the average will converge to actual state – action qualities. This estimation type is called Monte

– Carlo. Another way of estimating V^{π} and Q^{π} is to use a parameterized functions for V^{π} and Q^{π}, then adjust the functions to best fit the received rewards.

2.4.7. Optimal Value Functions

A policy π is optimal if and only if $V^{\pi}(s) \geq V^{\pi}{}'(s)$ for all $s \in S$. In other word a policy is optimal if there is no other policy which has a higher state quality for any actions. And the optimal property will be donated with a star superscript.

$$V^{*}(s) = \max_{\pi} V^{\pi}(s) \textbf{ For all } s \in S \tag{2. 15}$$

Likewise for state – action quality:

$$Q^{*}(s,a) = \max_{\pi} Q^{\pi}(s,a) \qquad \textbf{For all } s \in S \textbf{ and } a \in A(s) \tag{2. 16}$$

The $V^{*}(s)$ term can be written in terms of $Q^{*}(s,a)$.

$$V^{*}(s) = \max_{a} Q^{\pi^{*}}(s,a) \tag{2. 17}$$

For reinforcement learning problem the agent have estimates of $Q^{*}(s,a)$ or $V^{*}(s)$ which help the agent to judge the policy which is followed. If the policy leads to a low quality states or selects low qualified actions for an action, the policy should be modified. V^{π} (estimates of $V^{*}(s)$)function depends on the policy which is followed by the agents; so also it has to be modified for becoming a better estimate.

2.4.8. Temporal Difference (TD) Learning

The reinforcement problem is formulated in sections 2.4.1 through 2.4.7. There are three main approaches for solving the reinforcement learning problem.

- Dynamic Programming
- Monte Carlo methods
- Temporal Difference Models

In this thesis "Temporal Difference Models with Actor Critic Learning" is used [16]; Q- learning technique of TD ([16] and explicitly in [17]) will not be introduced since it is not utilized.

Temporal difference is in fact a combination of Dynamic Programming and Monte Carlo ideas. Such that TD can learn from experience without an environment model, also it can update estimates without waiting for final (episode) outcome. TD methods can update the value function at each time step without waiting for the final outcome.

$$V(s_t) \leftarrow V(s_t) + \alpha\left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right] \qquad (2.\ 18)$$

The term r_{t+1} stands for the observed reward. The upper update is called TD (0) and it updates the value function of state "s" $(V(s_t))$ towards the target $r_{t+1} + \gamma V(s_{t+1})$. If the observed reward and new state value is higher than the value of the old estimate then the value of old state is increased else it is decreased. Since the whole value function is estimate (both $V(s_t)$ and $V(s_{t+1})$);

TD uses the experience (r_{t+1}) on another estimate to obtain a new estimate. This is called bootstrapping.

It was shown that:

$$V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\} \tag{2. 19}$$

$$= E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right\}$$

$$= E_{\pi}\left\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s\right\} = E_{\pi}\{r_{t+1} + \gamma V^{\pi}(S_{t+1}) | s_t = s\} \tag{2. 20}$$

MC method uses the equation (2. 19) as a target. MC target is an estimate because the real expected return R_t is not known, but a sampled return is used instead. DP methods use the equation (2. 20) for a target. DP methods know the expected return, because it is supplied with a complete model of the environment, but $V^{\pi}(S_{t+1})$ is unknown for the current policy π and the estimate $V_t(S_{t+1})$ is used instead so the DP target is also an estimate.

TD also uses the equation (2. 20) for a target. For both the expected value of r_{t+1} and the term $V^{\pi}(S_{t+1})$ are unknown estimates are used instead so the TD target is also an estimate.

2.4.8.1. Actor Critic Learning

For actor critic methods the action selection (policy) is separated from the value function. The policy structure is called as actor, the estimated value function is known as critic. This learning is performed by the critiquing the actor by the critic.

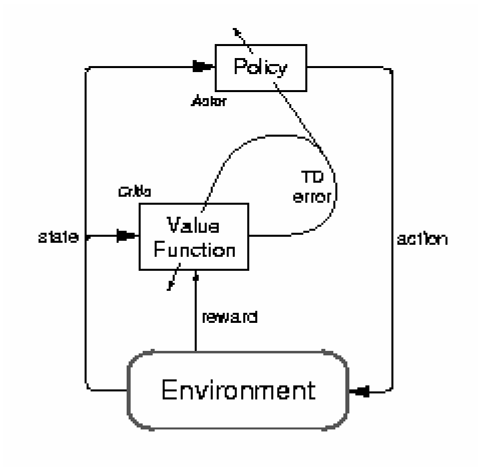


Figure 2.20. Fuzzy Actor Critic Learning

The critic forms an error signal called TD error, which tunes the policy and the critic both (figure 2.20). The TD is calculated by: [16]

$$TD\_error = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{2. 21}$$

The critic is updated as usual:

$$V(s_t) \leftarrow V(s_t) + \alpha TD\_error \tag{2. 22}$$

Where

α : Learning rate

Suppose the policy is in form:

$$\pi_t(s,a) = \frac{p(s,a)}{\sum_b p(s,b)} \tag{2. 23}$$

Where

$p(s,a)$: Probability of selecting action "*a*" is state "*s*".

Then this policy can be updates as:

$$p(s_t, a_t) \leftarrow p(s_t, a_t) + \beta TD\_error \tag{2. 24}$$

Where

β : Learning rate

2.4.9. FACL Controller

The state of the learner is calculated by use of a fuzzy inference system (FIS). The input to the FIS is the sensors which sense the state of the learner, the output of the FIS is the rule truth values which describe the state of the learner. The fuzzy inference system consists of fuzzy membership functions and rule conclusions:

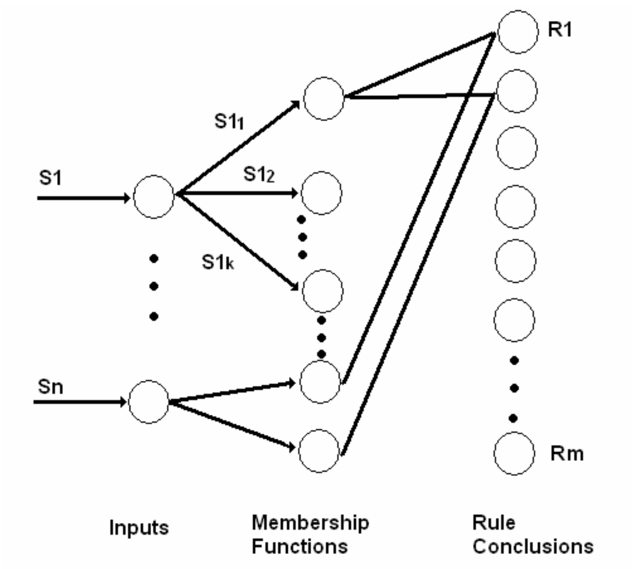


Figure 2.21. FIS

The first layer of the fuzzy inference system multiplexes the corresponding input to each fuzzy membership function. Membership functions calculate the membership degree of the input to the corresponding fuzzy label. [16]

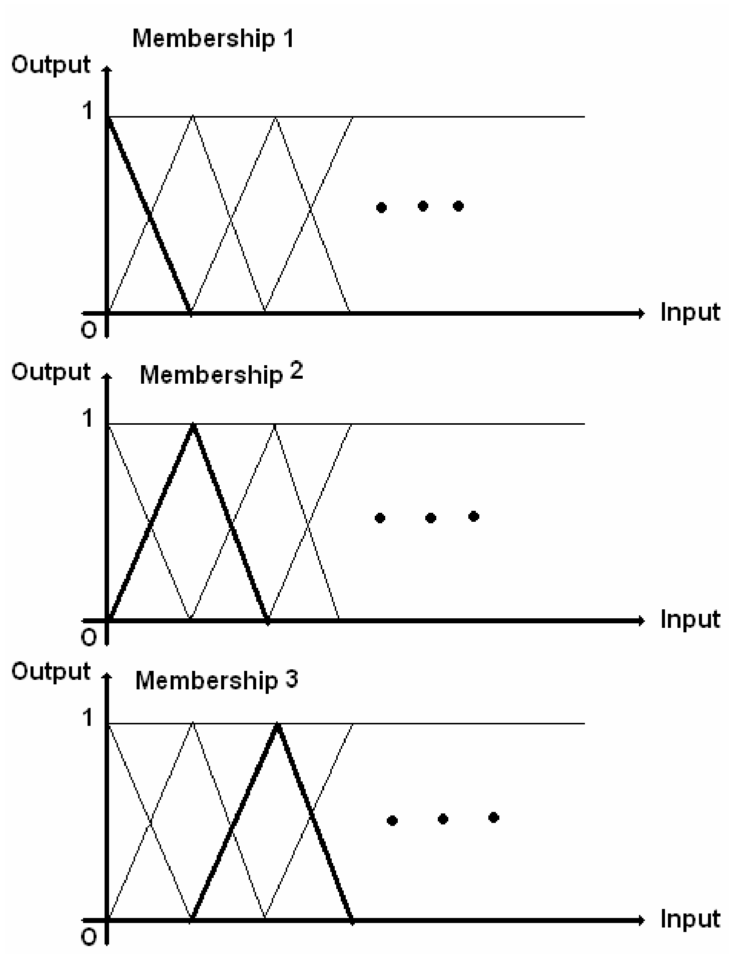


Figure 2.22. Membership Functions

The fuzzy membership functions illustrated above are in triangular, but infact it can be in various forms like trapezoidal, sigmoid. The triangular shape is selected for simplicity.

After the membership values of all labels are calculated, fuzzy inference system calculates the rule conclusions which are just a “multiplication” operation. If always any of the two fuzzy membership labels are activated at a time, the sum of rule conclusions will be guarantied to be *1*.

The main structure of a FIS is the shape and the number of fuzzy membership function and the number of inputs. The total number of output rule conclusions is determined by the number of inputs and the number of fuzzy labels for each input.

$$m = \prod_{i=1}^{n} k_i \quad \textbf{Where,} \tag{2. 25}$$

m	: Number of rules
N	: Number of inputs
k_i	: The number of fuzzy label for the ith input

For proper state determination FIS with sufficient number of inputs (sensors) and sufficient number of fuzzy labels with appropriate membership shapes should be used. For q-learning the FIS (fuzzy inference system) is not trained; it should be constructed by a supervisor. Fortunately this is not a very challenging task, with a priori knowledge of the sensors which will be used to sense the state of the learner, the number and the shape of the fuzzy memberships can be easily be constructed. Since the FIS will not be trained, use of simple discontinuous membership functions will not be disadvantageous. [16]

The action selection for each rule is done by a random process, in which the selection probability of an action is determined by its action quality. The probability of selecting the action "a" for the *i*th rule is:

$$P(w[a]^i) = {w[a]^i} \Big/ {\sum_{x=1}^{A} w[x]^i} \tag{2. 26}$$

If the state – action vector has non zero values for all actions, all actions will have a non zero selection probability. This rule selection guaranties exploration if the state – action vector is initiated with non zero values for all actions. Finally the total output is a weighted sum of the rules' selected actions:

$$Y_o = \sum_{i=1}^{m} U(w^i) Ri \tag{2. 27}$$

Y_o	: Total action output.
Ri	: *i*th rule strength.
$U(w^i)$	: Selected action for *i*th rule.

Fuzzy actor critic learning has a separate state evaluator which evaluates the states visited by the learner due to the policy followed by the learner as stated in section 2.4.1. In FACL the learner learns how to evaluate the states and at the same time the policy will be modified to visit better states. While the policy is being modified, the state evaluator is updated for the new policy.

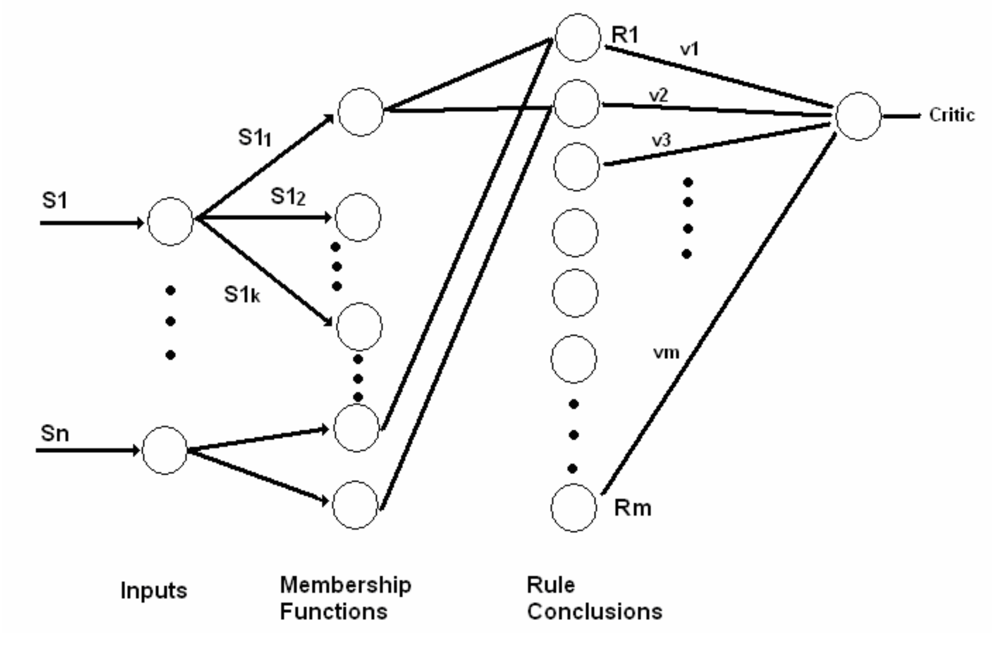


Figure 2.23. The Action Evaluator.

The critic output is:

$$Critic = V(S) = \sum_{i=1}^{m} R_i v_i \tag{2. 28}$$

For FACL, action selection vectors exist which are called as "state – action quality vector" and will be referred as action selection vector. [16]

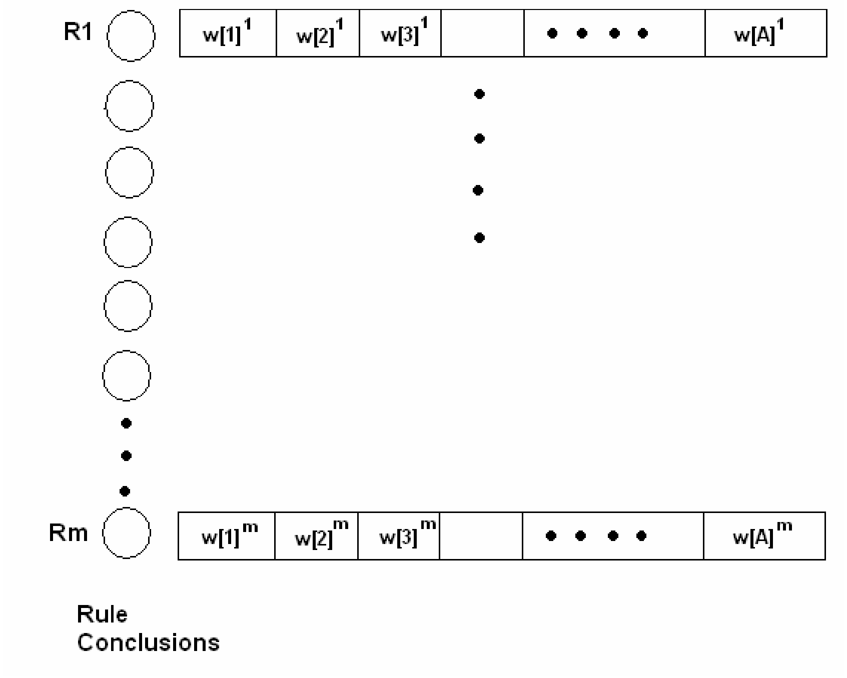


Figure 2.24. Action Selector.

Where: $w[k]^i$ k = 1, 2, 3 ,…, A : Discrete actions.

$w[A]^i$: The quality of the action A in state *i*.

TD error for FACL is:

$$\varepsilon_{t+1} = r_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t) \tag{2. 29}$$

r_{t+1}	: External reinforcement received at time *t+1*.
$S_t = [R_1^t R_2^t R_3^t ....R_m^t]$	: The state of the machine at time *t*.
$V_t(S_t)$	: The critic for state S_t which is calculated by (2. 29).

Both the state evaluator and the action selector are tuned by the TD error:

$$v_{t+1}^{i} = v_{t}^{i} + \beta \varepsilon_{t+1} \overline{\phi_t} \tag{2. 30}$$

$$w_{t+1}^{i}(U_t^i) = w_t^i(U_t^i) + \varepsilon_{t+1} e_i^t (U^i) \tag{2. 31}$$

The eligibility traces $\overline{\phi_t}$ and e^t are calculated as:

$$\overline{\phi_t} = \phi_t + \lambda \overline{\phi_{t-1}} \tag{2. 32}$$

$\overline{\phi_t}$	: Eligibility trace of the rules at time *t*.
$\phi_t = \left[R_1^t R_2^t R_3^t R_4^t ....R_m^t \right]^T$	: The vector of rule strengths at time *t*.
λ	: Eligibility rate.

The eligibility rate weights the old time steps. For *t* = 0 the eligibility trace is the strength of the rules at that time:

$$\phi_0 = \left[R_1^0 R_2^0 R_3^0 R_4^0 ....R_m^0 \right]^T \tag{2. 33}$$

After the eligibility of the rules is calculated, the eligibility of the state-action vector can be calculated as:

$$e_t^i(U^i) = \begin{cases} \lambda' e_{t-1}^i (U^i) + \phi_t^i, (U^i = U_t^i) \\ \lambda' e_{t-1}^i (U^i), otherwise \end{cases} \tag{2. 34}$$

λ'	: Eligibility rate.

So the state – action vectors update becomes:

$$w_{t+1}^{i}(U_t^i) = w_t^i(U_t^i) + \beta\varepsilon_{t+1}e_i^t(U^i) \qquad (2.\ 35)$$

Note that there is no learning rate term for action selection vector training in (2. 31). The reason for this absence is that the learning rate task is accomplished by the action evaluator. Action evaluator function determines not only the sign of TD error but also the magnitude; thus it plays a learning rate role.

The implementation of FACL is as follows: [16]
Suppose that the robot is in time t+1, with and applied action U_t and received reinforcement r_t.The fuzzy q-learning is implemented as follows:

- Current rule strengths are calculated by fuzzyfying current sensor outputs.
- New TD error is computed. (2. 26)
- The state-action vectors are updated with the TD error and the old action eligibility which was calculated at time t. (2. 28)
- New actions are elected with new updated state – action vectors and total action is calculated by taking the weighted average of the elected actions (2. 36).
- New actions and the rule strengths are readjusted by the eligibility for the actions are recalculated by the old eligibility. The new calculated eligibility will be used at time t+2 as old action eligibility.

2.4.10. Proposed Update Mechanism

For the controllers whose output is a linear combination of the values of same kind; it is not always true to apply the update to the last selected actions. Following situation is with seven discrete actions:

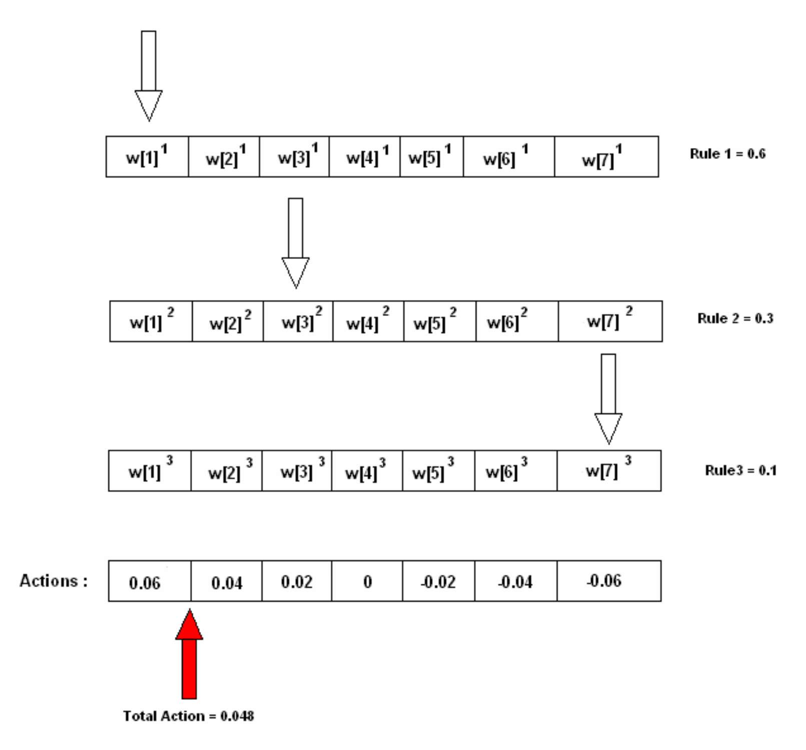

w[1]^1 w[2]^1 w[3]^1 w[4]^1 w[5]^1 w[6]^1 w[7]^1
Rule 1 = 0.6
w[1]^2 w[2]^2 w[3]^2 w[4]^2 w[5]^2 w[6]^2 w[7]^2
Rule 2 = 0.3
w[1]^3 w[2]^3 w[3]^3 w[4]^3 w[5]^3 w[6]^3 w[7]^3
Rule3 = 0.1
Actions :
0.06
0.04
0.02
0
-0.02
-0.04
-0.06
Total Action = 0.048

Figure 2.25. Selected Actions for Rules.

In the figure above, three rules are active; and the marked actions are selected for each rule. The output is a weighted linear combination of the selected actions of the rules which appears to be 0.048 which is marked with red in the figure.

*Total Action = 0.6 * 0.06 + 0.3 * 0.02 + 0.1 * -0.06 = 0.048*

In the situation above rule 3's action is conflicting with the total output action. If the total selected 0.048 action was good, it is not wise to straighten the rule 3's conflicting action which itself probably is not a good action or vise versa if the total selected action was wrong, it is not wise to weaken the rule 3's conflicting action which is a good action.

An update technique which applies the updates to the neighbor actions instead of elected actions will be proposed. Suppose the same situation where the total output was 0.06; the neighbor actions are 0.06 and the 0.04.

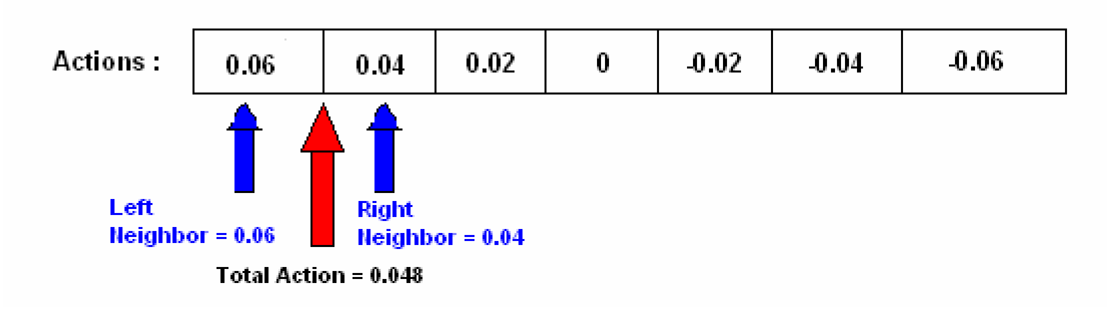


Figure 2.26. Action Neighbors

Total action is a linear combination of its neighbors. The weight of the neighbors to produce the total output determined the strength of the neighbors:

Right neighbor strength = (total action – left neighbor) / (right neighbor – left neighbor)

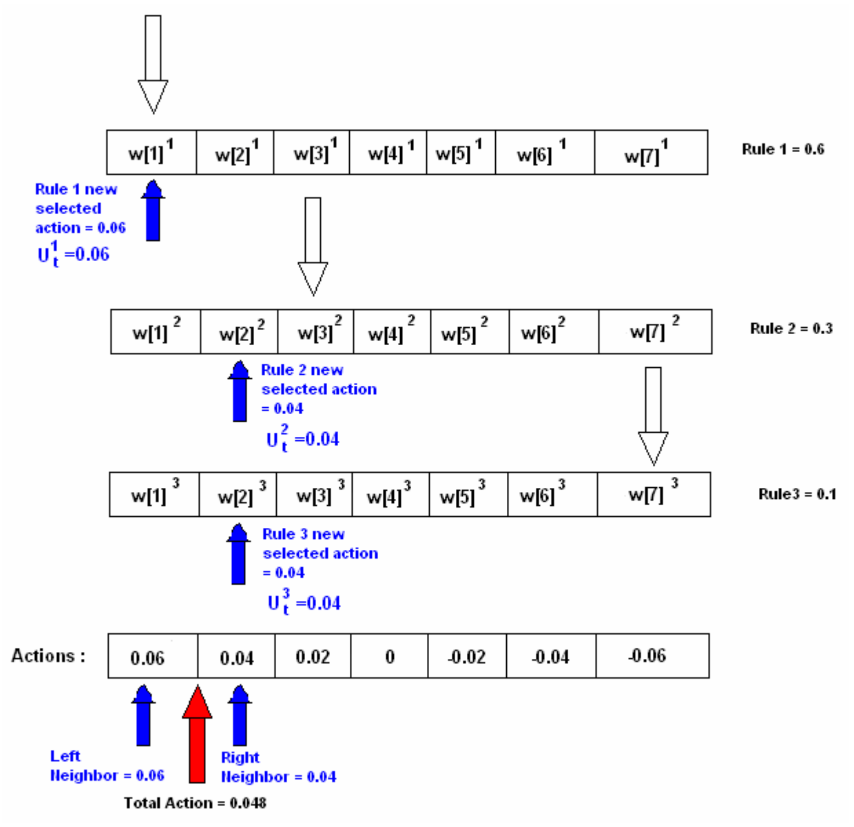
Left neighbor strength = 1 – right neighbor strength (2. 37)

In the case above:

Right neighbor strength = 0.6

Left neighbor strength = 0.4

All selected actions of the active rules will be reset to the neighbor, which is nearer to themselves.



w[1]^1 w[2]^1 w[3]^1 w[4]^1 w[5]^1 w[6]^1 w[7]^1
Rule 1 = 0.6
Rule 1 new selected action = 0.06
U_t^1 =0.06
w[1]^2 w[2]^2 w[3]^2 w[4]^2 w[5]^2 w[6]^2 w[7]^2
Rule 2 = 0.3
Rule 2 new selected action = 0.04
U_t^2 =0.04
w[1]^3 w[2]^3 w[3]^3 w[4]^3 w[5]^3 w[6]^3 w[7]^3
Rule3 = 0.1
Rule 3 new selected action = 0.04
U_t^3 =0.04
Actions :
0.06 0.04 0.02 0 -0.02 -0.04 -0.06
Left Neighbor = 0.06
Right Neighbor = 0.04
Total Action = 0.048

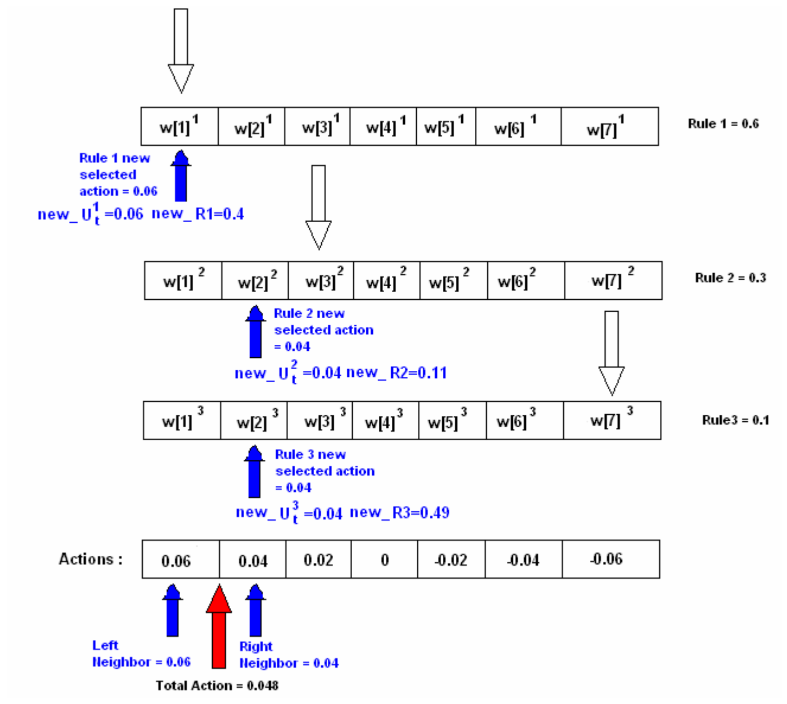
Figure 2.27. Actions Member to the corresponding Neighbors

The strength of the neighbor actions will be shared among their members. The sharing will be done by

$$new\_R_i^t = \left|\text{Total Action - old\_}U_t^i\right| * R_i^t \Big/ \sum_{\substack{k \in \text{all rules having the same} \\ \text{neighbor membership}}} \left|\text{Total Action - old\_}U_t^k\right| * R_k^t \qquad (2.38)$$

Note that rules with more radical actions (which are away from the total rule) are weighted more during the neighbor strength sharing.

For the example, new rule strengths for new selected actions:



w[1]¹ w[2]¹ w[3]¹ w[4]¹ w[5]¹ w[6]¹ w[7]¹
Rule 1 = 0.6
Rule 1 new selected action = 0.06
new_U$_t^1$ =0.06 new_R1=0.4
w[1]² w[2]² w[3]² w[4]² w[5]² w[6]² w[7]²
Rule 2 = 0.3
Rule 2 new selected action = 0.04
new_U$_t^2$ =0.04 new_R2=0.11
w[1]³ w[2]³ w[3]³ w[4]³ w[5]³ w[6]³ w[7]³
Rule3 = 0.1
Rule 3 new selected action = 0.04
new_U$_t^3$ =0.04 new_R3=0.49
Actions :
0.06 0.04 0.02 0 -0.02 -0.04 -0.06
Left Neighbor = 0.06
Right Neighbor = 0.04
Total Action = 0.048

Figure 2.28. New adjusted Actions and Shared Strengths

From this point on the eligibility trace for the state – action vector update can be calculated by the new rule strengths and new selected actions. Note that new proposed action selection and rule strength calculation does not disturb the total action and updates the state – action vectors smoothly. With the smooth updating the controller will not seek all linear combinations of the action set which won't make any output difference. With truncated action set search, the learning will speed up.

In the succeeding chapter, a controller hierarchy composed of fuzzy logic controllers utilizing FACL learning scheme will be constructed based on the theoretical background explained in this chapter. Also if applicable, the proposed update scheme (section 2.4.10) will be integrated to the update mechanism of controllers, but a performance analysis will not be done for the new update mechanism. Each controller in the hierarchy will be trained on the simulation model of the implemented hardware robot.

CHAPTER 3

CONTROLLER ARCHITECTURE ILLUSTRATED WITH SIMULATION RESULTS

A simulation model of the mechanical snake robot is developed based on MATLAB – SIMMECHANICS equipped with more additional features than those of the hardware robot prototype, to overcome some mechanical limitations and to be able perform further gaits. The simulation model possesses both kinematic and dynamic properties overcoming any limitations that the hardware has and is able to perform extra gaits than the ones the hardware has.

3.1. Robot Simulation Model

Due to consistency with the hardware snake robot, its simulation is also composed of interconnected bodies in series but to have better visualization of the control capabilities of our architecture the number of segments is doubled to a number eight (figure 3.1).

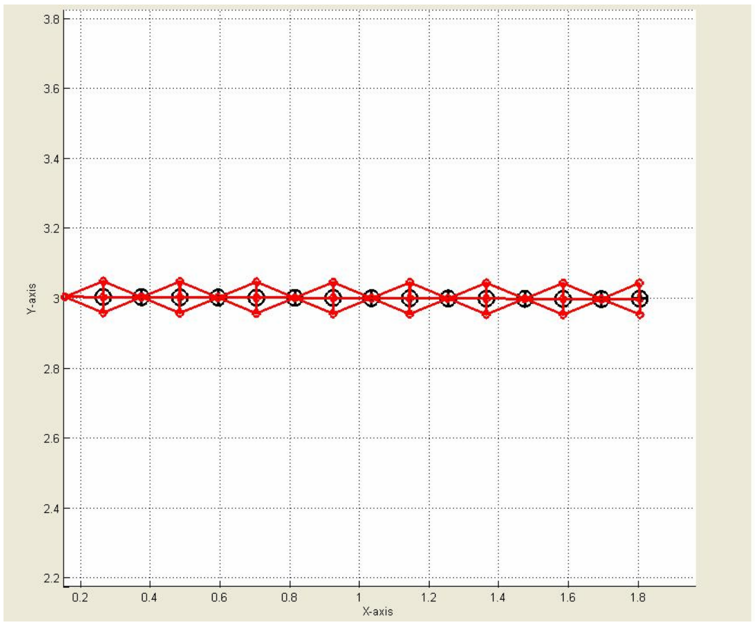


Figure 3.1. Robot model view from –z direction with SimMechanics visualization.

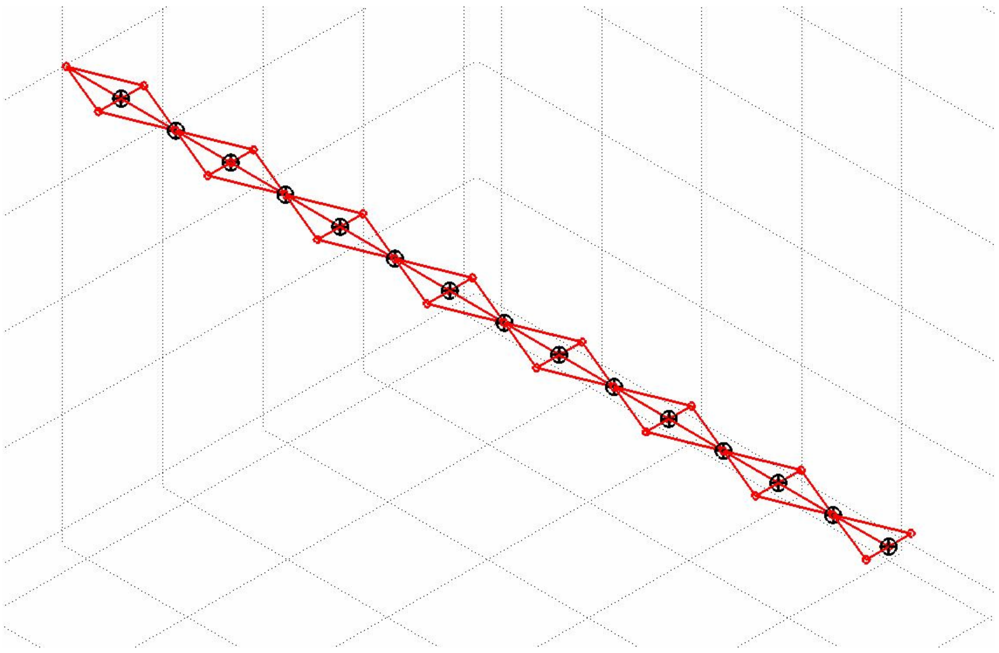


Figure 3.2. 3-D view of robot model with SimMechanics visualization.

3.1.1. Body

Each of the body segments is identical to each other with same inertia having a mass of 100g, a length of 22 cm and 9 cm for width.

3.1.2. Joints & Actuators

Body segments are connected with two degree of freedom joints that can rotate around a local z axis (marked with blue lines in figure 3.3); translate along a local x axis. The rotation angle of the joints about the z axis is limited with +/- 70 degrees; the translation length is limited to 8 cm. Joints are actuated actively and independently in all degree of freedoms. There are no constraints on the actuators of the joints; it is assumed that each actuator in the joints has sufficient force or torque to actuate the joints at all conditions.

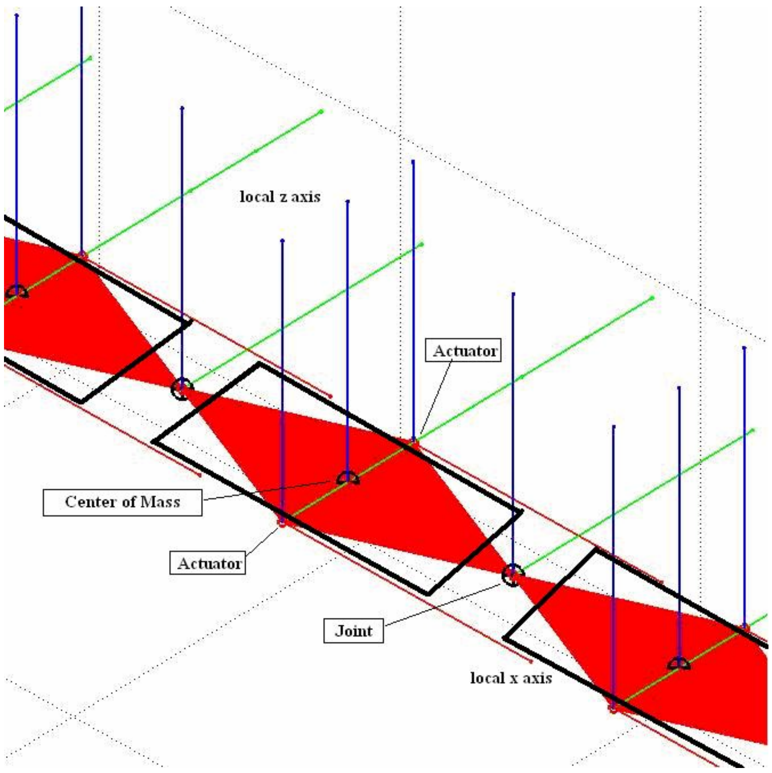


Figure 3.3. Close view of a body segment with SimMechanics visualization.

The dual actuators as clearly labeled in figure 3.3 can supply forces up to the friction force of the environment in positive or negative local x axis direction.

These dual actuators will be referred to as “tip end actuators” in the successive sections. In the y axis of the tip end actuators which is marked green in figure 3.3, the segments are directly under the effect of environmental friction. The “tip end actuators” in the model are to simulate the effects of robot segment palettes of the hardware robot in variable friction environments. The friction force magnitude is modeled in the environment which will be dealt in detail in section 3.1.3.

All of the actuators of the simulated robot are controlled separately. The positions of the actuators in the joints are controlled via constant angular or linear speeds, and the speed of the body segments are controlled via the tip end actuators (figure 3.4) in each body segments. All low level control is assumed to perform perfectly. The inputs to the specified low level controllers are derived from the upper level controllers which will be introduced in sections 3.3.1 and 3.3.2.

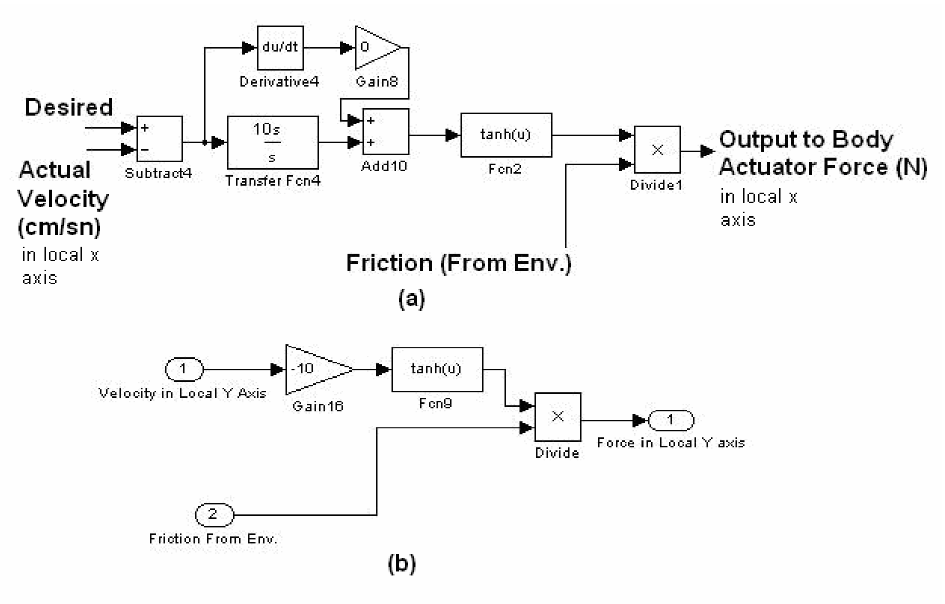


Figure 3.4. a). Low level controller of the tip end actuators at local x axis. (b) Low level controller of the tip end actuators at local y axis.

The low level controller in figure 3.4a responds correctively to any deviations between the desired and the actual speed along the local x axis by generating a force which is equal to the friction value obtained from the interaction between the palette and the environment surface. The controller in figure 3.4b directly responds to motions along the local y axis with a value in reverse direction, whose magnitude is equal to the friction force. Hyperbolic tangent is used to scale the output of the controllers in figure 3.4. The rationale of using a hyperbolic tangent as a limiter is to avoid use of discontinuous functions, such sign function, thus improves the performance of the SIMULINK solver.

The proportional coefficients of the PID controllers in figure 3.4, which were 10, 0, 0 for P, I, D respectively, are determined intuitively in order to approximate a sign function. The rationale of approximating the sign function comes from the friction model of simulations which will be introduced in detail in section 3.1.3.1. Higher coefficients better approximate to sign function but in cost of oscillations at high frequencies. High frequency oscillations degrade the performance of the SIMULINK solver, so the proportional coefficient selected due to the performance of the solver. These controllers in figure 3.4 all together simulate the behavior of a robot segment palette due to the environmental friction. The control of the actuator (dc motor) of a palette is not simulated and assumed to be performing perfectly. The details of the friction model and the rationale of palette simulation will be discussed in 3.1.3.1 section.

Although the model is seemingly planar; any body segment can be modeled as lifting by setting all friction forces acting on the particular segment to zero. Lifting scheme will be used for some gaits (refer to 3.2.).

3.1.3. Environment

For simulation, 1021 by 702 pixels colored bitmap environments are used where a pixel corresponds to two cm. These environments have regions of different frictions indicated by the color intensity of that domain. Full bright (white) region corresponds to maximum available friction, whereas the decrease in intensity means a lower scale friction value. Environments also contain obstacles indicated by pure black color (refer to figure 3.5). The number, shape, size of the obstacles and friction domains can be totally arbitrary.

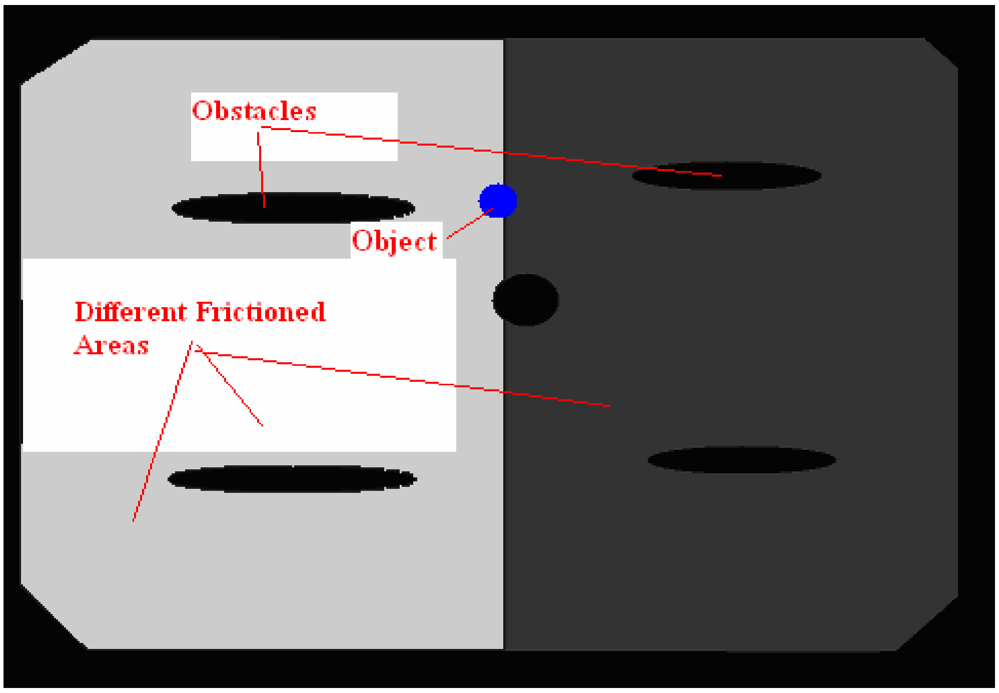


Figure 3.5. The environment. The friction is determined by the intensity of the domains. The total black areas are considered to be obstacles.

The maximum friction force exerting on the model is set to be 5 N (Newton) which is almost half the weight of the hardware model. The color intensity of a domain is normalized by the maximum friction force which is applied to the robot model segments. During simulations each robot model segments faces the friction of the corresponding domain which they are in. Also an object is located in the environment which is indicated with pure blue color.

The IR range detector positions and orientations are modeled identical to the implemented hardware snake robot so as to have a range of 200 pixels; and ten degree of beam width. The accelerometers are simply modeled by body sensor attached to each body segments measuring the planar accelerations in local x and y axes.

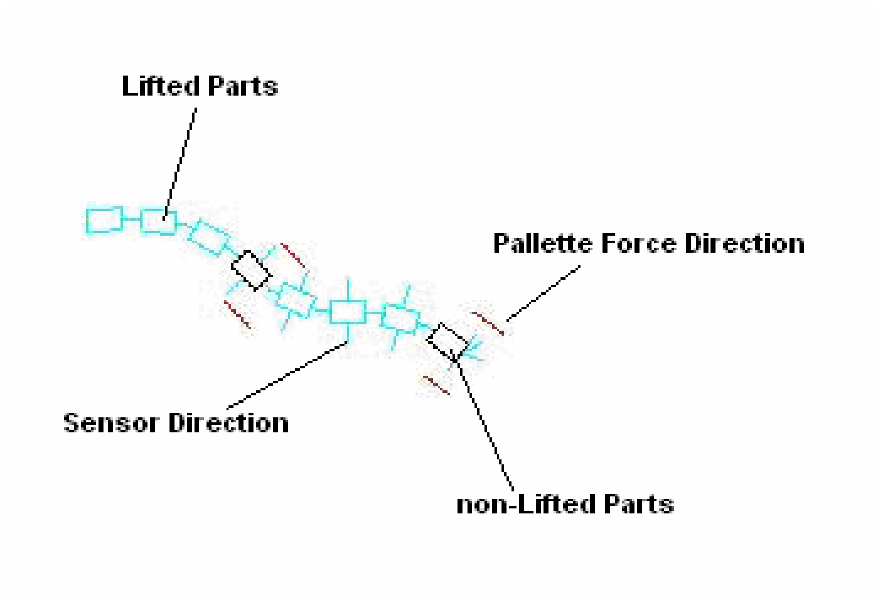


Figure 3.6. Custom Visualization of the Robot model in an environment.

The visualization of the simulated model is represented in figure 3.6. The lifted parts of the body are drawn with light blue, where the parts touching on the ground are drawn with solid black color. The IR sensors orientations are represented by blue lines on the segments. The red lines at the sides of the segments represent the palette force direction. The red line's length is determined by its corresponding palette's desired speed, 0.5 pixel length for one cm/sec.

3.1.3.1. Friction Model

Coulomb friction model is used in the model as shown in figure 3.7. [20]

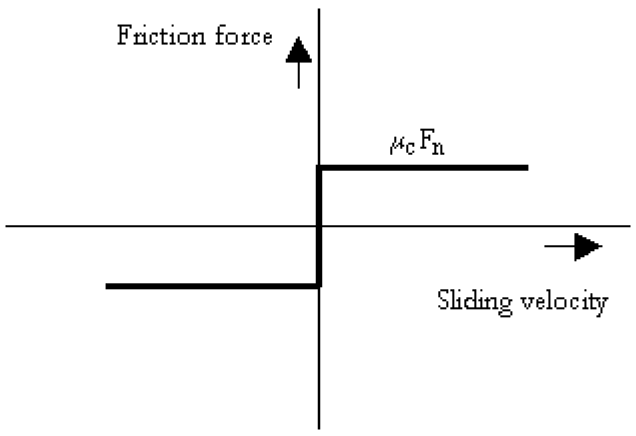


Figure 3.7. Coulomb Friction.

The magnitude of the friction force is scaled by the environment and is an input to the tip end actuators (refer to 3.1.2). The magnitude of friction force is assumed to be coming from the environment directly so it is not calculated through the weight of the robot segments.

The nature of the friction model in figure 3.7 is a sign function scaled by the magnitude of environment friction. Thus the response of a segment palette to motions which is not equal to its turning speed is simply the magnitude of the environment friction force.

3.2. Application of Snake Gaits

The proposed snake robot model can perform the snake gaits introduced in section 1.6.2 with proper inputs to the model's low level controllers. The gaits are performed by making the snake robot model imitate biological snakes' overall movements. The proposed snake-like robot model differs from biological snakes, so its resultant gaits show difference in dynamical and frictional characteristics. The detailed simulation result of the implemented snake gaits' characteristics can be found in section 3.4.1. We will now concentrate on models for creation of the gaits.

3.2.1. Lateral Undulation

Lateral undulation is performed by the coordinated control of the speed and orientation in steering body segments; no translation motion is used. For lateral undulation linear velocity of the segments are identical; but steering depends on body curvature. Tip end actuators are also involved in the steering of the body segments.

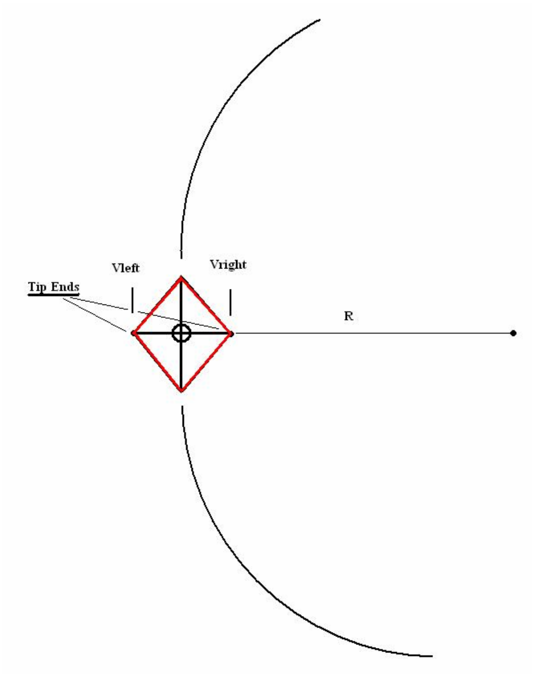


Figure 3.8. The steering of a body segment.

Any differences between the speeds (V_{left}, V_{right}) at the tip end actuators causes the segment to rotate in a circle as shown in figure 3.8. The mentioned circle's radius is determined by the amount of the difference between tip end speeds and the width of the segment (figure 3.8). If no slippage occurs, the two following conditions are always held:

$$V_{linear} = {(V_{left} + V_{right})}/{2} \quad (3.1) \qquad\qquad {V_{left}}/{R+a} = {V_{right}}/{R} \quad (3.2)$$

Where

R : Radius of the circle.

a : Width of the segment.

V_{linear} : The linear speed of the segment.

After determining linear speed and steering of a segment, and using the two conditions of (3. 1), (3. 2); the necessary speeds of the tip ends (V_{left}, V_{right}) actuators can easily be calculated.

For lateral undulation it is assumed that each body segments has its own turning radius when chasing predecessor segment's radius as shown schematically in figure 3.9.

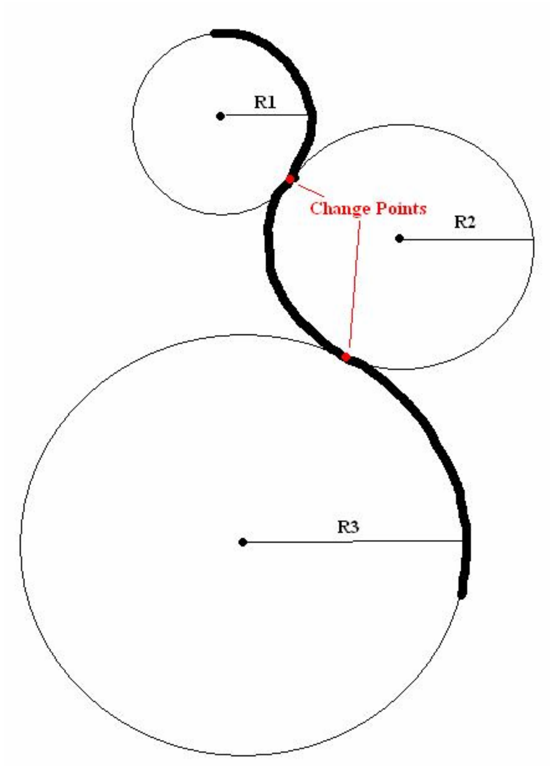


Figure 3.9. The path of the segments during lateral undulation.

Each segment moves as closely matching as possible to its corresponding radius. After reaching the change points which are infact the touching points of the circles; the corresponding circle changes, thus its radius, yielding the steering of the body segment through the adaptation changes to the radius of its predecessor (figure 3.9).

Joint angles between the segments are calculated according to the radius of the circle that segments are affected at that instant. If two consecutive segments are under the affect of the same circle (having same radius) the joint angle is kept at a fixed value which can by calculated from a simple trigonometric analysis in figure 3.10.

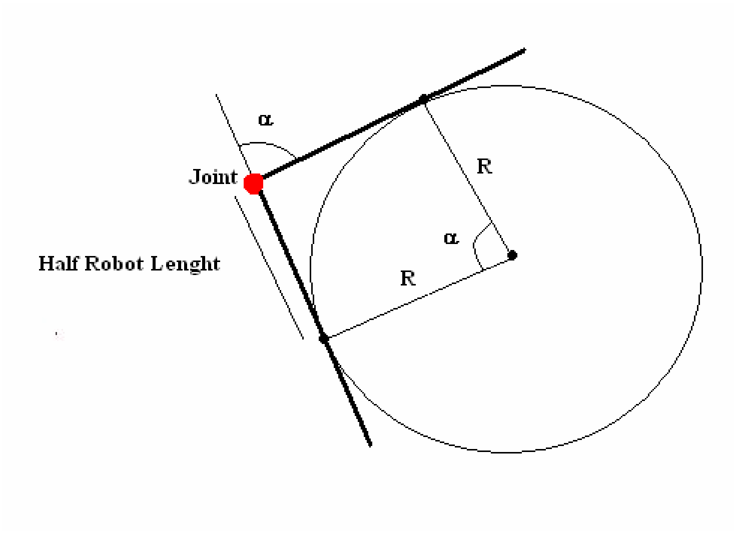


Figure 3.10. The angle between the consecutive segments.

If two consecutive segments are in different circles, the target angle is calculated as if they were in the same (the leading segment's) circle and the joint is turned to meet this target angle with a constant speed which is derived to be:

$$\dot{\theta} = \frac{V}{r} - \frac{V}{r'} \tag{3. 3}$$

Where

V : Linear velocity of the segments.

r : The radius of the leading segment.

r' : The radius of the lagging segment.

$\dot{\theta}$: Joint angle speed in rad/sec.

The induced lateral undulation scheme is for constant speed motion. But this scheme will also hold if same acceleration is applied to all of the segments and the angle speeds changes ($\dot{\theta}$) are also accelerated according to linear acceleration.

The steering is done for the head segment, down to the successive body segments which follows the head. The head steering is determined independently since it has no predecessor. If the head segment is steered by alternatively changing its turn radius of motion, the resulting motion will be the most known "s" shaped lateral undulation.

Lateral undulation begins from straight body. So before lateral undulation gait, the body is reformed by setting all servo angles (rotational axis along z axis) and prism joint lengths to zero as shown in figure 3.11.

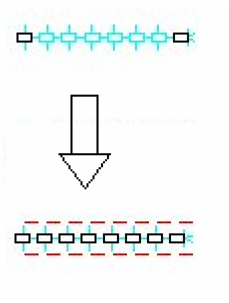


Figure 3.11. Reformation move for Lateral Undulation

3.2.2. Accordion

This gait is performed by induced translations from the joints along the local x axis direction. The skin friction properties of biological snakes exhibit a low friction in the forward motion while it has a high friction in the backward direction. This characteristic results in forward propagation. The skin characteristics of the biological snakes are simulated by actively control of the tip end actuators so as to exert a force in the forward direction.

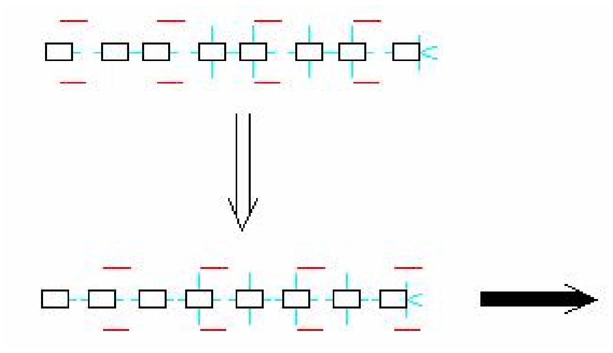


Figure 3.12. Accordion Gait. The red lines indicates the forward forces applied by the body tip actuators. The forward force exerted on the translating body segments results in forward body motion.

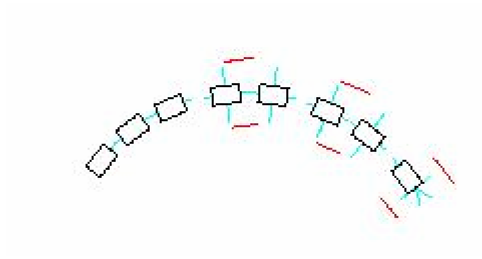


Figure 3.13. Steering for Accordion Gait. The steering is accomplished by setting all the joint angles according to the desired turn radius at the same time equally. The tip end actuators are driven according to the turn radius which was discussed in 3.2.1 lateral undulation.

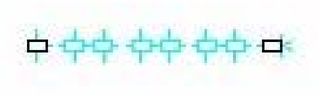


Figure 3.14. Reformation move for accordion. The joint servo angles are set to zero, but prismatic joints are set to alternating 0 and 8 cm.

3.2.3. Rectilinear

This gait is similar to the accordion gait; but this time the forward propagating segments are lifted by properly setting the friction values effecting at the corresponding segment to zero. For rectilinear gait no special skin friction is necessary; so tip end actuators are not used for this gait.

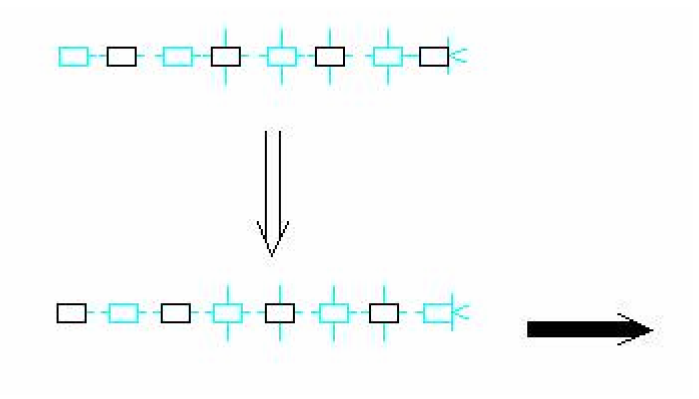


Figure 3.15. Rectilinear Gait. The body segments being translated are lifted as shown by blue segments.

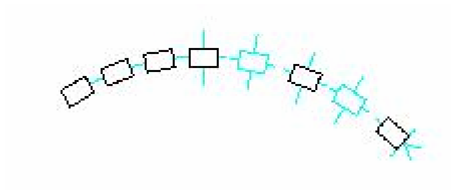


Figure 3.16. Steering for Rectilinear Gait. The steering is similar to the accordion gait.

For rectilinear gait, same reformation scheme is used than the one used for the accordion gait which was given in figure 3.14.

3.2.4. Sidewinding

Sidewinding is similar to the lateral undulation but, the body only touches the ground at two segments; other segments are lifted. The segments which have the largest heading angle difference with respect to the head direction touch the ground, other segments are lifted. The body can be moved to either left or right while navigating in the forward direction. The rotation motions in two different directions are illustrated in figure 3.17 and 3.18.

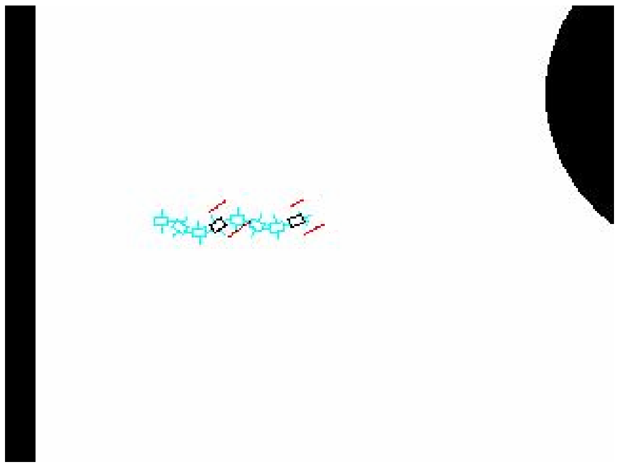


Figure 3.17. Sidewinding to left side. Segments having positive slope with respect to head position touch the ground.

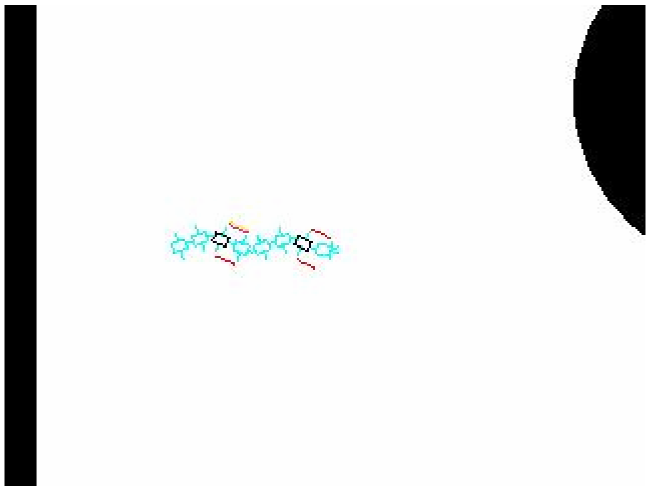


Figure 3.18. Sidewinding to right side. Segments having negative slope with respect to head position touch the ground.

The resulted motion is expected to be in the resultant direction of the touching segments (marked with red lines in figure 3.17 and figure 3.18.). No steering strategy is used for side winding.

For reformation, a priori curvature is given to the body as shown on figure 3.19.

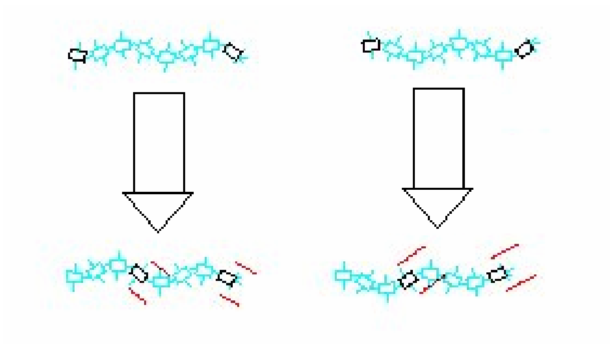


Figure 3.19. Reformation schemes for sidewinding.

3.3. Controller Network

The rectilinear, accordion and lateral undulation gaits have steering capability that can as well be used for obstacle avoidance or object reaching. The main purpose of the snake robot is to combine obstacle avoidance and object reaching behaviors while selecting the best applicable gait for the local environmental conditions. These tasks are carried out by a 2 layer controller architecture. The first layer selects the gait to be performed; while controllers in the second layer conduct obstacle avoidance behavior and object reaching. (Figure 3.20)

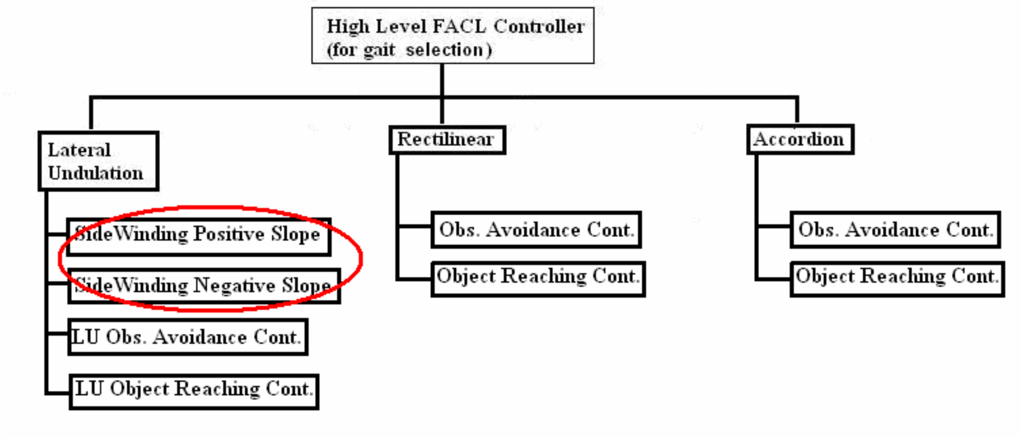


Figure 3.20. Controller Network.

The controllers responsible for obstacle avoidance and object reaching will called as "middle level controllers" throughout this thesis and the gait selector controller will be referred as "high level controller" throughout this thesis. For each of the lateral undulation, rectilinear and accordion gaits, there exist two middle level controllers for obstacle avoidance and object reaching. One behavior, thus one middle level controller, is activated at a time for each gait, due to object proximity and sensed obstacle distances. The activation of a behavior is accomplished by a bi-stable switch to avoid oscillations between behaviors (refer to 3.3.1.1.). The main structure of the middle level controllers of each gait is similar to [21].

It should be noted that Sidewinding (marked with red in figure 3.15) to either left or right side (with positive or negative slope) is an extension of lateral undulation and does have an explicit steering. This gait does not need any middle level controllers, thus it is added to lateral undulation for being selected as a preset gait by the high level controller as a derivative of lateral undulation as shown on figure 3.20.

The middle level controllers determine the steering of the mode due to their purposes. The total linear speed of the model is not explicitly controlled; instead it is taken as a constant input considering the limitations of the hardware mechanical robot. In the simulations a speed of 15cm/sec is used which corresponds to 7.5 pixel/sec.

3.3.1. Middle Level Controllers

3.3.1.1. Obstacle Avoidance Controllers

All three of the obstacle avoidance controllers are fuzzy logic controllers emphasizing FACL learning. All obstacle avoidance controllers for each gait are identical but tuned to handle their corresponding gaits for obstacle avoidance (Marked with red in figure 3.21.). Obstacle avoidance controllers operate with a frequency of 1Hz. With the 7.5 pixel/sec speed, 1Hz operation is long enough in the application of the selected steering.

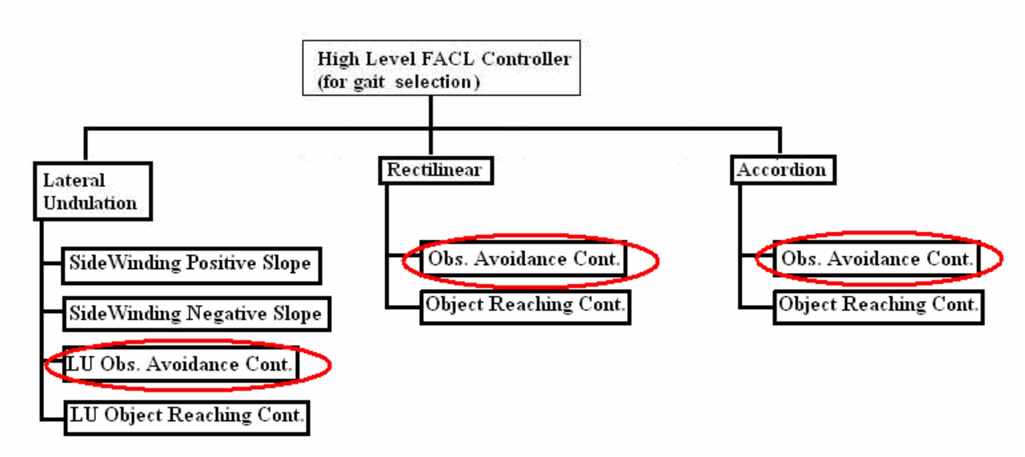


Figure 3.21. Controller Network.

Each controller has first four of the IR distance sensors as an input (two sensors looking forward, the minimum one of left sensors and the minimum of right sensors); which are fuzzified by 5 fuzzy sets leading to 625 states (figure 3.21). The reason of selecting only first four of the sensors is that the head's sensory information is sufficient to represent the states of the robot for navigation and to keep the input state space small enough to shorten training time. The fuzzyfication of the sensors is done based on triangular membership functions uniformly distributed between 0 to 200 pixels where only two of fuzzy label memberships are possible to occur simultaneously. The selection of the number of sensory fuzzy sets is manually done as stated in section 2.3.9.

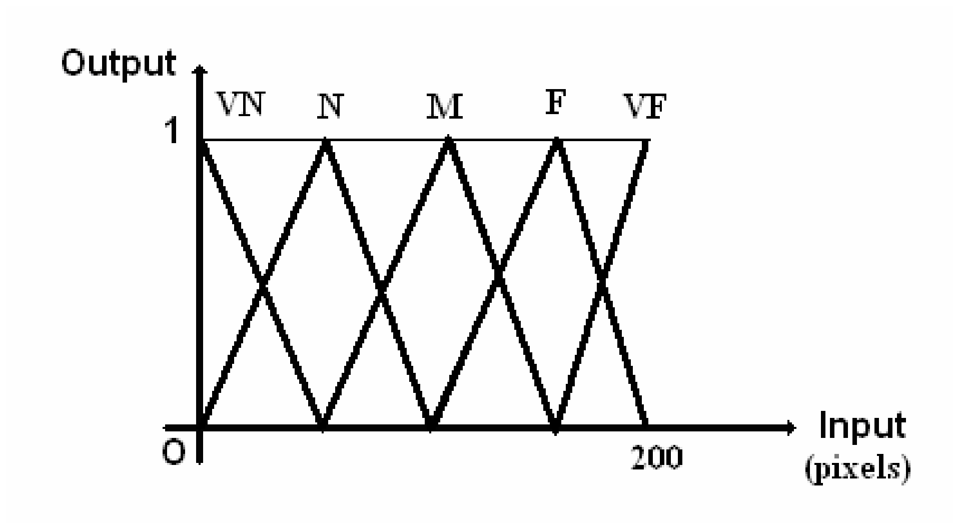


Figure 3.22. Fuzzy Membership Functions of IR Sensors

Each of the middle level controllers is based on seven identical steering radii vectors such that they output a combination of them.

$$r \rightarrow [-33 \quad -22 \quad -11 \quad 0 \quad 11 \quad 22 \quad 33] \qquad (3.4)$$

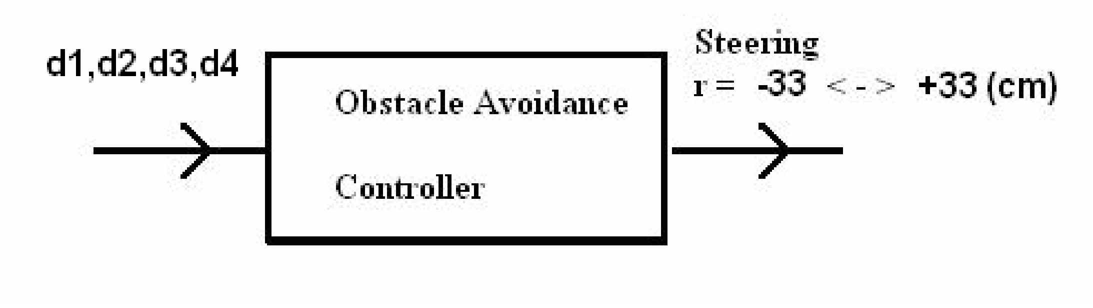


Figure 3.23. Middle Lever Obstacle Avoidance Controller

The actions set of controller in figure 3.23 represents the radius of the circle which the head segment is on. With 15cm/sec speed, a radius of 33cm corresponds to 26 degree/sec heading change speed. With maximum steering, the model can make a180 degree turn in 7 seconds.

3.3.1.1.1. Training of Obstacle Avoidance Controllers

Each of the obstacle avoidance controllers (marked with red in figure 3.21) utilizes the FACL learning architecture.

Table 3.1. FACL Parameters

FACL Parameters for Obstacle Avoidance Controller [$\gamma\lambda\lambda_a\beta_0$]
[0.1 0.3 0.3 0.1]

The following reinforcement scheme is used:

$$\text{reinforcement} = \left| \begin{array}{c} +1\,\text{if }(d_1, d_2 > 100\,\text{and}\,\min(d_3,\ldots,d_4) > 40) \\ -1\,\text{if }(d_1, d_2 < 40\,\text{or}\,\min(d_3,\ldots,d_4) < 20) \\ 0\ \text{otherwise} \end{array} \right. \qquad (3.\,4)$$

If a collision occurs the model is taken back to a previous “safe” location, and the navigation is maintained while giving a “-1“reward for the colliding action. The trainings are done in a 5 N friction environment which supplies 5N friction to each of the robot segments that is a force threshold enough for properly achieving any of the gaits available.

During the training phase, the update mechanism induced in 2.2.1 is used. The trainings of each obstacle avoidance controllers are continued in arbitrary environments until 100 successive good reinforcements are received.

3.3.1.2. Object Reaching Controller

Object reaching controllers for any selected gait are fuzzy logic controllers with preset LUT (look-up table) where no tuning is necessary (figure 3.21 and 3.22). Reaching of the object in the environment is quite an easy task when compared to obstacle avoidance so a simple controller is used with a predetermined LUT for each gait.

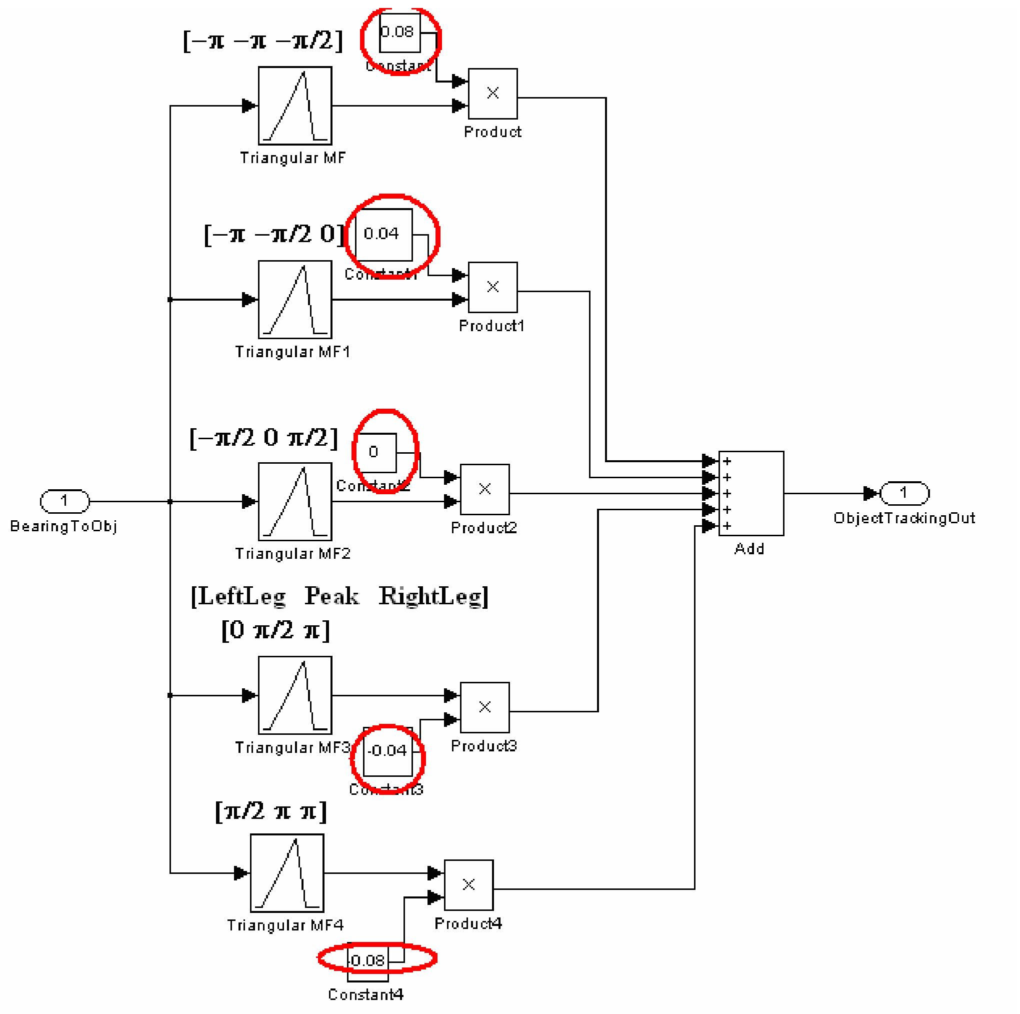


Figure 3.24. Object Reaching Controller for Lateral Undulation. The LUT ,visible in the constant value boxes marked with red, are populated by 1/r values.

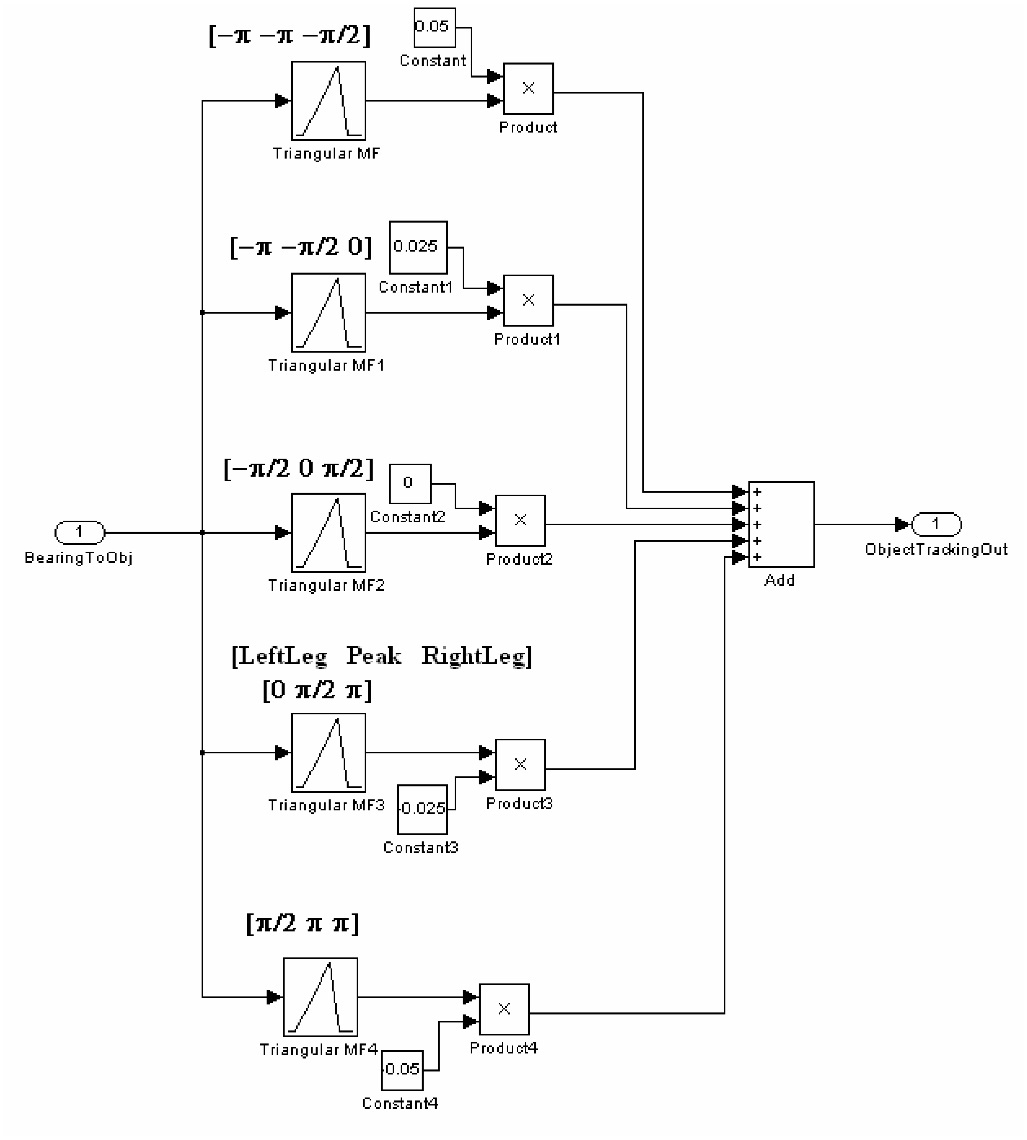


Figure 3.25. Object Reaching Controller for Accordion and Rectilinear

3.3.1.3. Selection of Behavior

The selection among obstacle avoidance and object reaching behaviors is accomplished by a bi-stable switch which works based on object heading, distance and the distance to the nearest sensed obstacle. The switching parameters are determined intuitively in this thesis.

$$\text{behaviour} \longrightarrow \begin{cases} \text{object reaching; if } \min(d_1,d_2) > 80 \ \& \ obj.dist < 150 \ \& \ abs(obj.bear) < 40 \\ \text{obs. avoidance; if } \min(d_1,d_2) < 30 \ or \ obj.dist > 200 \ or \ abs(obj.bear) > 60 \\ \text{else maintain previous selection} \end{cases}$$

(3. 5)

Switches identical in all controllers determine the behaviors of the gaits. When "obstacle avoidance" behavior is active; the explicit selection of sidewinding or lateral undulation gaits are done by high level controller. Even though sidewinding gaits can not steer, they can still be used to avoid the obstacles in some situations. But when "object reaching" behavior is selected, sidewinding gaits are suppressed and are replaced automatically by "lateral undulation with object reaching" regardless of the high lever controller's output since sidewinding gait can not reach an object.

3.3.2. High Level Controller

High level controller has four IR sensor inputs; two for head, other two represent data from right and left side sensors sensing the nearest obstacles. The sensors are fuzzified generating three fuzzy membership functions instead of five. The robot segments are assumed to sense the friction of the environment and the controller also has an input the average of the friction forces affecting the robot segments. Each input is fuzzified by three fuzzy sets generating 243 input states. The main reason for the generation of three fuzzy set for each sensory input is to limit the total number of the states which guaranties to shorten the training time.

The high level controller selects five actions (gaits) at each phase which are lateral undulation, sidewinding, accordion and rectilinear. The combination of actions is not applicable because they can not be super-positioned. The controller selects the most preferred action at a time. Thus the output is the gait which has the greatest support from the valid states at a time.

3.3.2.1 Training of High Level Controller

The high level controller is also a FACL controller which is tuned throughout the reinforcements. The training of high level controller is done with already trained middle level controllers which were induced in section 3.3.1.1. High level controller aims to select the best gait which stays away from the sensed obstacles without slippage. The best gait is chosen based on maximizing the reinforcement which is calculated as:

$$\text{reinforcement} = \begin{bmatrix} +1 \text{ if } \min(d_3,...d_{18}) > 20 \text{ and } \min(d_1, d_2) > 50 \text{ and not skidding} \\ -1 \text{ if } \min(d_3,...d_{18}) < 10 \text{ or } \min(d_1, d_2) < 30 \text{ or skidding} \\ \text{else } 0 \end{bmatrix}$$

(3. 6)

The slippage is determined by monitoring the instantaneous linear velocity by the body sensors attached to each body segment of each segment along local y axis (perpendicular to the segment direction), where the linear velocity should be zero when proper traction is achieved. If the linear velocity along local y axis exceeds a predetermined threshold, the robot is assumed to be slippage. The threshold will be derived in section 3.4.1.5 while observing the friction characteristics of the gaits. In section 3.4.1.5 also a minor modification will be done to the (3.6) reinforcement. The training is conducted until 50 successive high reinforcements are received in an arbitrary environment where selection of 50 gaits is sufficient to visit almost every state of the controller.

The following FACL parameter set is used during the trainings:

Table 3.2. FACL Parameters of High Level Controller

FACL Parameters for Obstacle Avoidance Controller [$\gamma\lambda\lambda_a\beta_0$]
[0.5 0.1 0.1 0.5]

3.3.3. Object Dragging

The object is modeled as a massless 7 pixel diameter circle in the environment. The robot can sense the distance and the direction of the object with respect to the each segment center. The object can be pushed and translated by any part of the robot.

When the robot reaches the object, it drags the object by pushing with its head while continuing with the object reaching behavior. The pushing continues until the robot senses a nearby obstacle. Robot can sometimes loose the object from the front of its head while performing lateral undulation due to the undulations. The robot ceases "object pushing" behavior for also these cases.

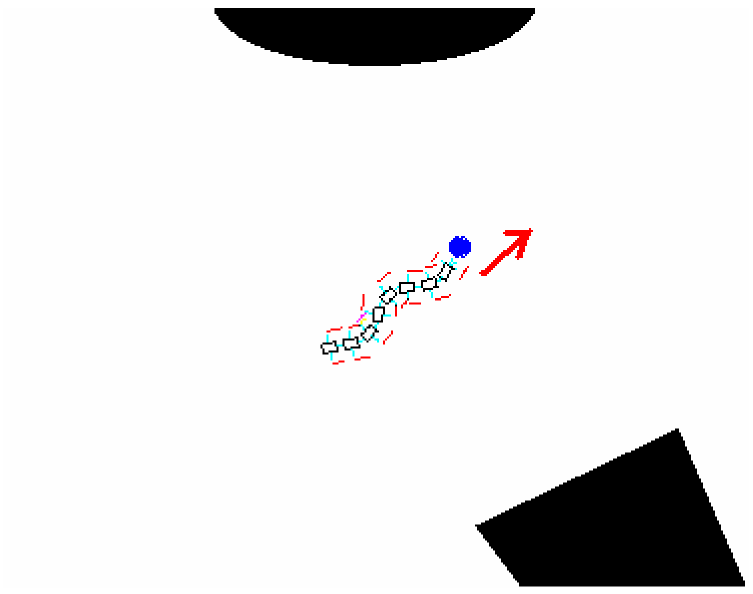


Figure 3.26. Object Push by head.

The manipulation of the object is still possible by using the snake body when the head misses the object. For manipulation with body; the following preset additions, which are rule based modules, are made in the control architecture.

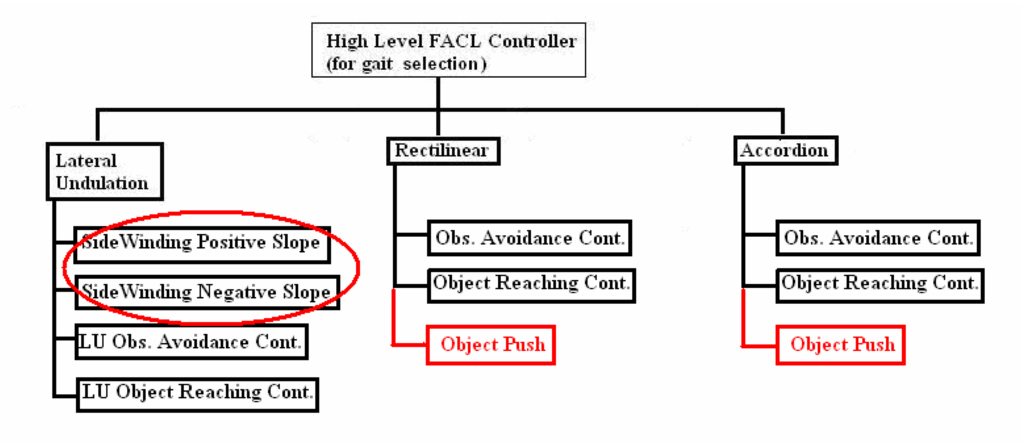


Figure 3.27. Body Push Extensions

For lateral undulation, an object on the body side can be manipulated with the body curvatures along the motion path by performing sidewinding gait in the corresponding direction as shown on the left of the figure 3.28. For rectilinear and accordion gaits it is not possible to manipulate the object along the robot motion direction, but the object can be pushed away by the body in order to clear a passage for the robot as shown on the left side of figure 3.28. The details of object push methods will be demonstrated and discussed in section 3.4.4.

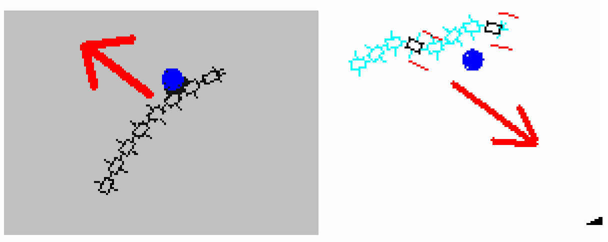


Figure 3.28. Body Push methods.

These schemes are triggered when rule based criteria is met. For simulations this criterion is considered to be 90 degrees or higher object direction angles with respect to the head segment (the object is not in front of the body) and a maximum of 5 pixels from the nearest body segment. This criterion is determined by intuition but it is obvious that the object should be on the side and close enough to the body.

Table 3.3. Object Push Activation Criteria

Distance to the nearest segment	**Bearing w.r.p. to head**
< 5 pixels (10 cm)	> 90 degrees

Object interaction is utilized only on the simulation model. Implemented mechanical robot can not perform interaction with an object because it lacks the sensors for detecting an object and determining its distance and orientation.

3.4. Simulation Results

3.4.1. Gait Friction Characteristics

In this section, implemented gaits will be introduced and their corresponding friction characteristics will be observed. Each gait requires some amount of friction force from the environment to be conducted properly. In absence of the required friction a particular gait will may not manage to navigate the robot. Due to insufficient friction, the robot may begin slipping resulting in loss of stability and ineffective navigation.

The friction susceptibility of each gait is determined by measuring the scalar sums of linear speeds of the segments along the local y axis, which is perpendicular to the palette direction, for environments with different frictions. When the segments are not slipping, due to the nature of the palettes, the linear speeds along local y axis of the each body segments should be zero. A nonzero linear speed along local y axes indicates slippage and results in unwanted heading change. Nevertheless body segment's inertia will always cause slippage. In the end of this section, a threshold for the determination of the slippage will be determined; this threshold will be the main criteria for reinforcement of the high lever controller.

The scalar sums of all body segments' perpendicular speeds (along local y direction) may show noisy characteristics. To have better determination of slippage, perpendicular speeds can be integrated to calculate the amount of slippage in distance quantity. The sum of the all segments' slippage distances for a period of time instance gives a brief idea about the gaits friction characteristics.

Slippage is very closely related to the linear speed and steering of the body, where sharp steering may cause slippage; so the worst case scenario (figure 3.29) is used for friction analysis. The speed is fixed at 15cm/sec.

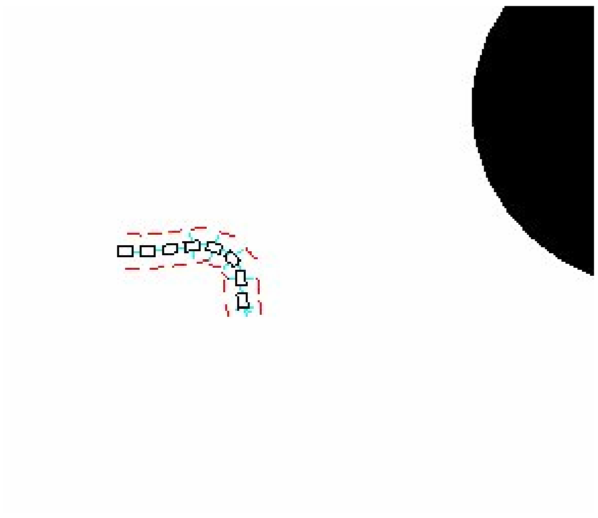


Figure 3.29. The scenario for lateral undulation where the model makes a sharp turn.

Rectilinear and accordion gaits are steered by curving the whole body as stated in 3.2.2 and 3.2.3. The curving of the body yields to local y axis speeds on each body segment as shown on the figure 3.30. So for proper friction analysis, the local y axis speeds are not taken into consideration while rectilinear and accordion gaits are steering. During steering of rectilinear and accordion gaits, local y axis speeds are simply not monitored for slippage. Since the steering of accordion and rectilinear is not continuous, ignoring local y axis speeds does not degrade detection of slippage.

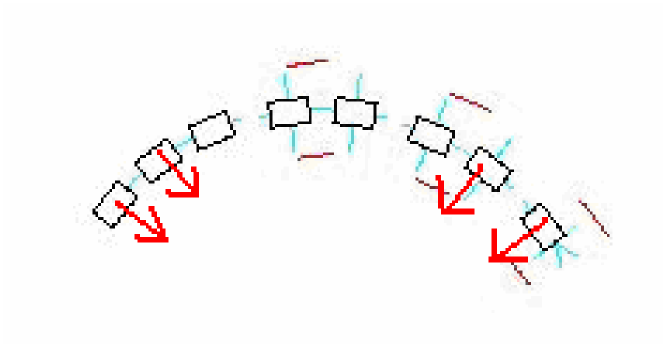


Figure 3.30.The steering for rectilinear gait, same scheme is also done for accordion.

The lifted part's perpendicular speeds are disregarded since the lifted parts do not touch the ground thus they do not slip.

3.4.1.1. Accordion

The visualization of the snake-like robot model performing accordion gait over a terrain with 5 N friction without any steering is illustrated in figure 3.31. Forward motion is generated by use of the translation motion of the joints where all segments touches the ground. The segments which are propagating forward are supported by their actuators (indicated by red lines by their corresponding segments in figure 3.31), simulating the frictional effects of a snake skin.

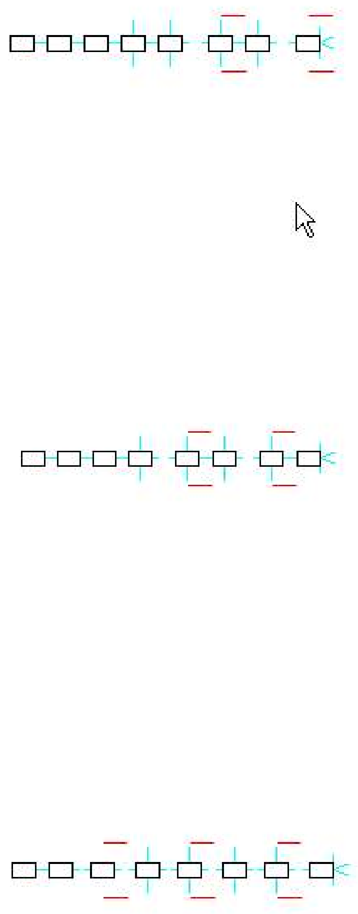


Figure 3.31. Accordion gait resultant motion in environment with 5 Newton friction force. The body propagates forward steadily but the speed is reduced to half.

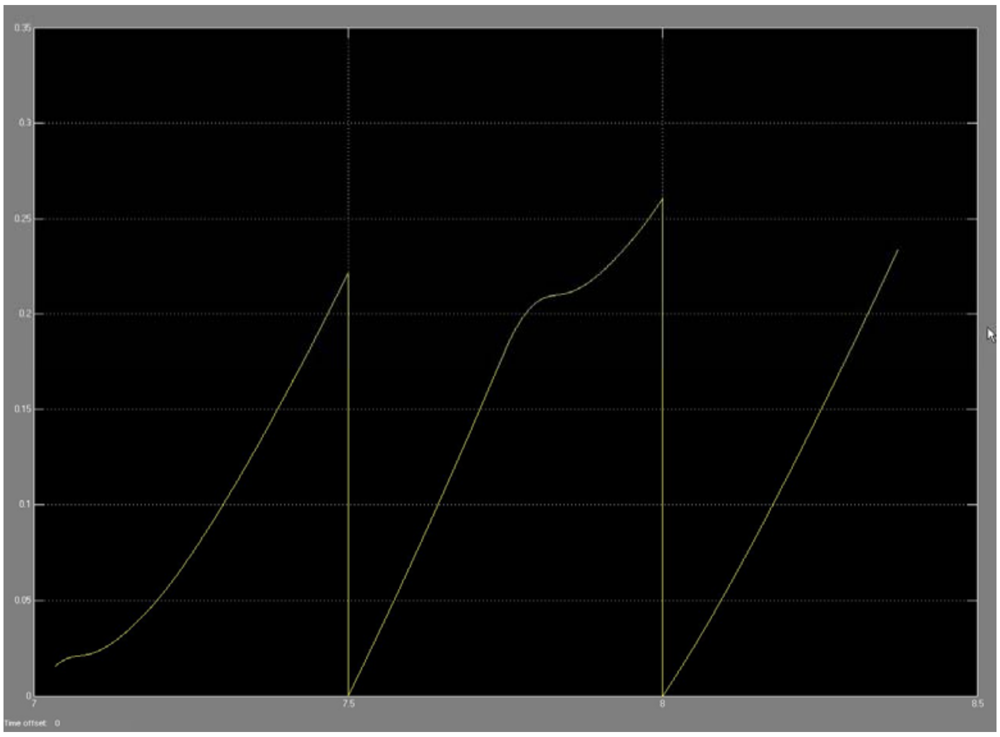


Figure 3.32. Slippage distance of accordion on 20N friction. The amount of slippage does not exceed 0.25cm per half second.

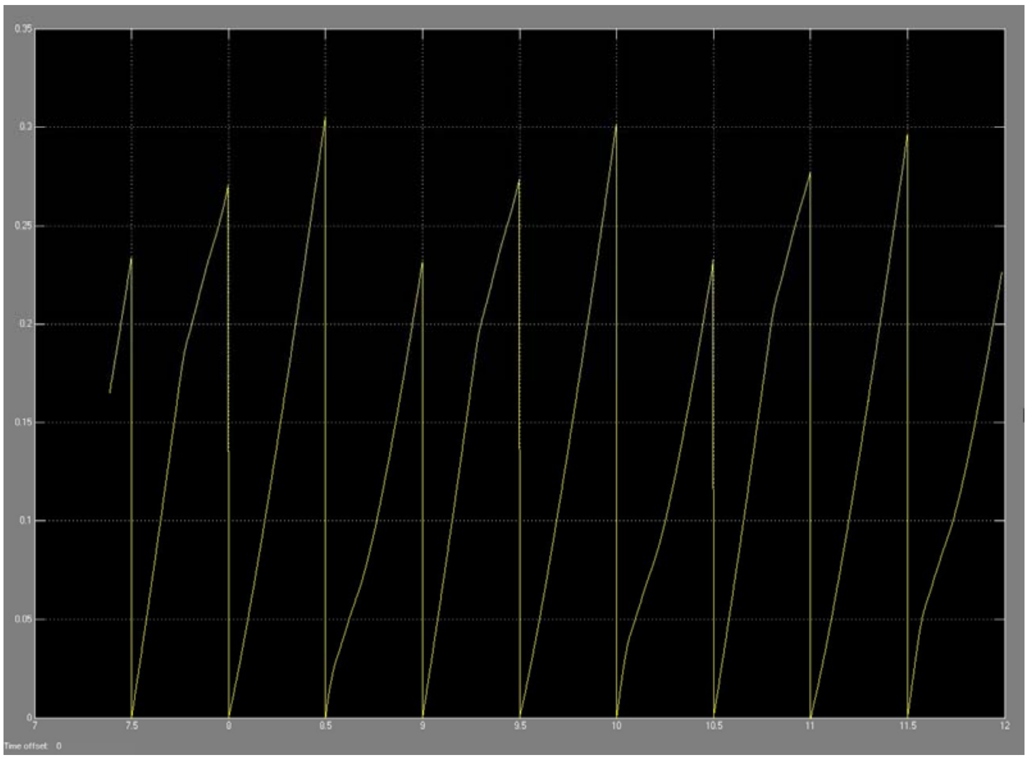


Figure 3.33. Slippage of accordion on 2 N friction. A maximum of 0.3cm slippage occurs per half seconds.

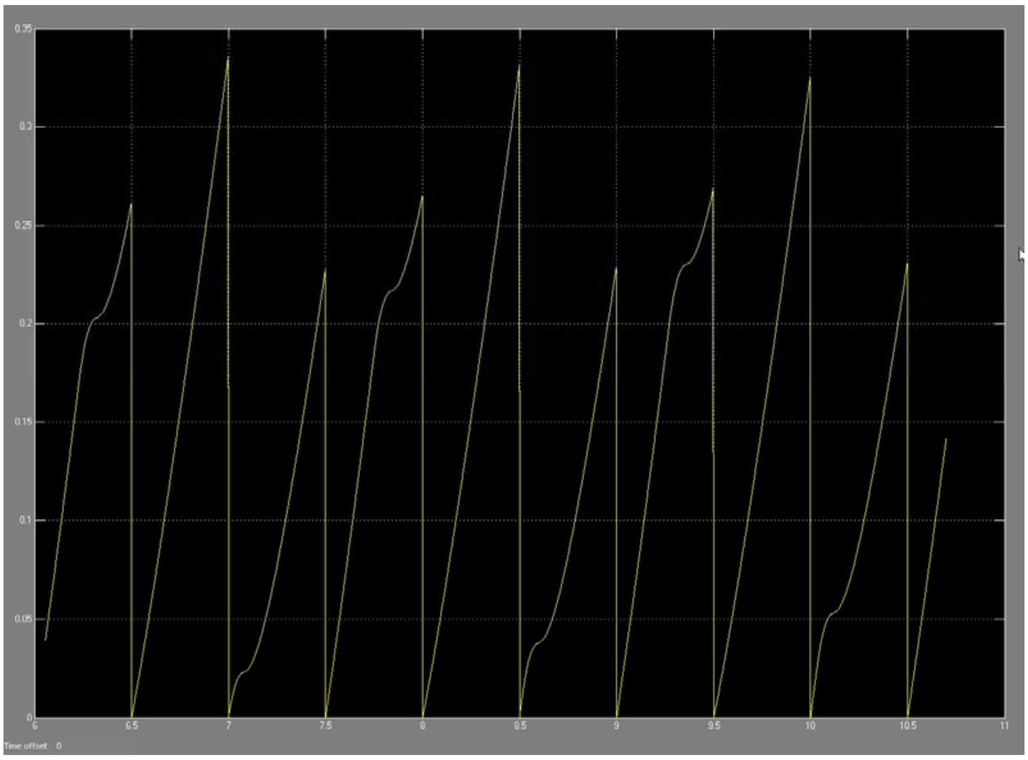


Figure 3.34. Slippage of accordion on 0.02 N friction. A maximum of 0.35cm slippage occurs per half seconds.

Accordion gait's slippage distance per half seconds is illustrated in figures 3.32, 3.33, 3.34 in environments with 20, 2, 0.02 N friction respectively for the scenario identical with that of figure 3.29.

Accordion gait slips 0.35 cm in 0.5 seconds during steering in an environment modeled with 0.02N friction. Under 20N friction the slippage distance is almost the same, yielding an accordion gait that is not significantly affected by friction change. Accordion gait is applicable in environments with low friction without significant slippage.

3.4.1.2. Rectilinear

The visualization of the snake-like robot model performing rectilinear gait over a terrain with 5 N friction without any steering is illustrated in figure 3.35. The forward motion is generated by the translation motions of the joints along body direction. The segments which are propagating forward are lifted shown with light blue borders in figure 3.35. The tip end actuators are not used.

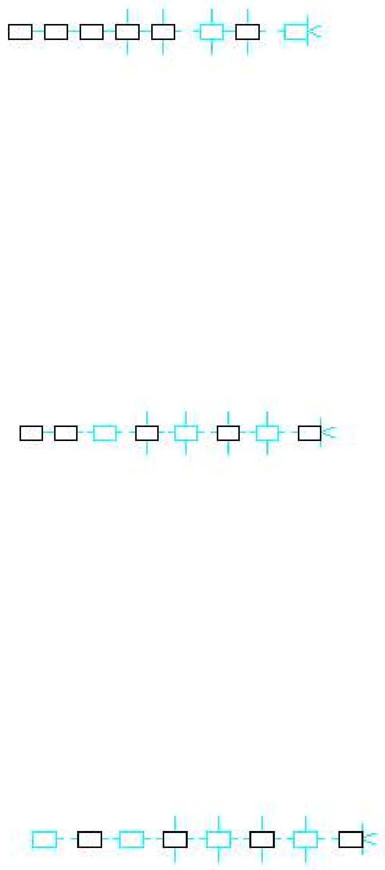


Figure 3.35. Rectilinear gait in environment of 5N. The linear speed of the whole body is half of the segments speed like the rectilinear gait.

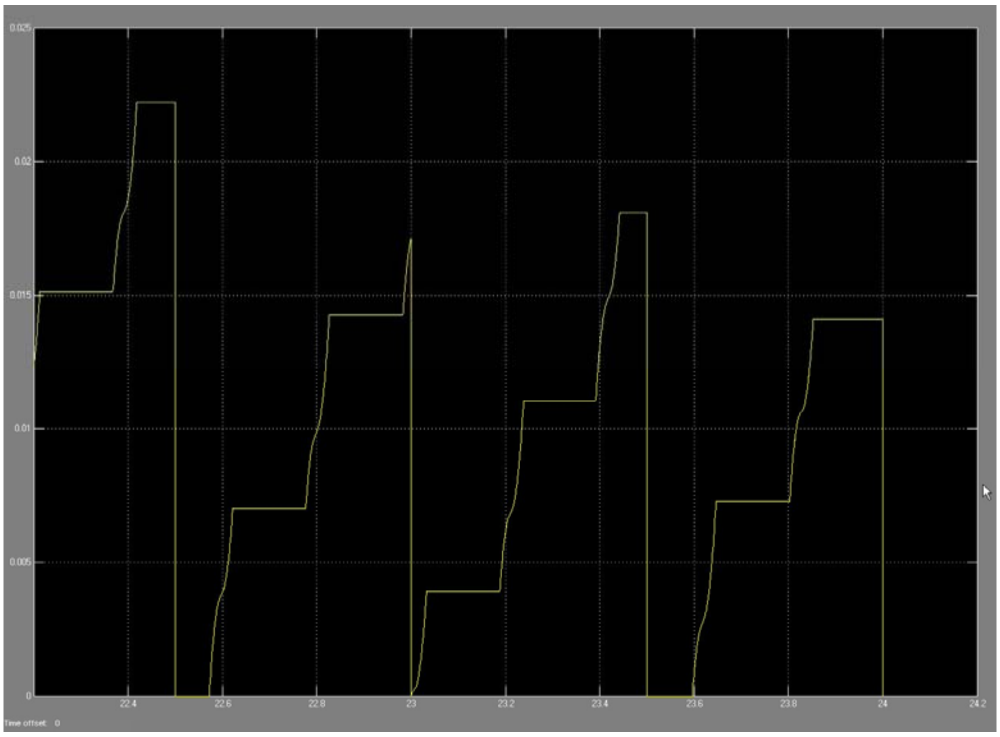


Figure 3.36. The slippage distance on 20N friction environment. The total amount of slippage does not exceed 0.025 cm for 0.5 seconds.

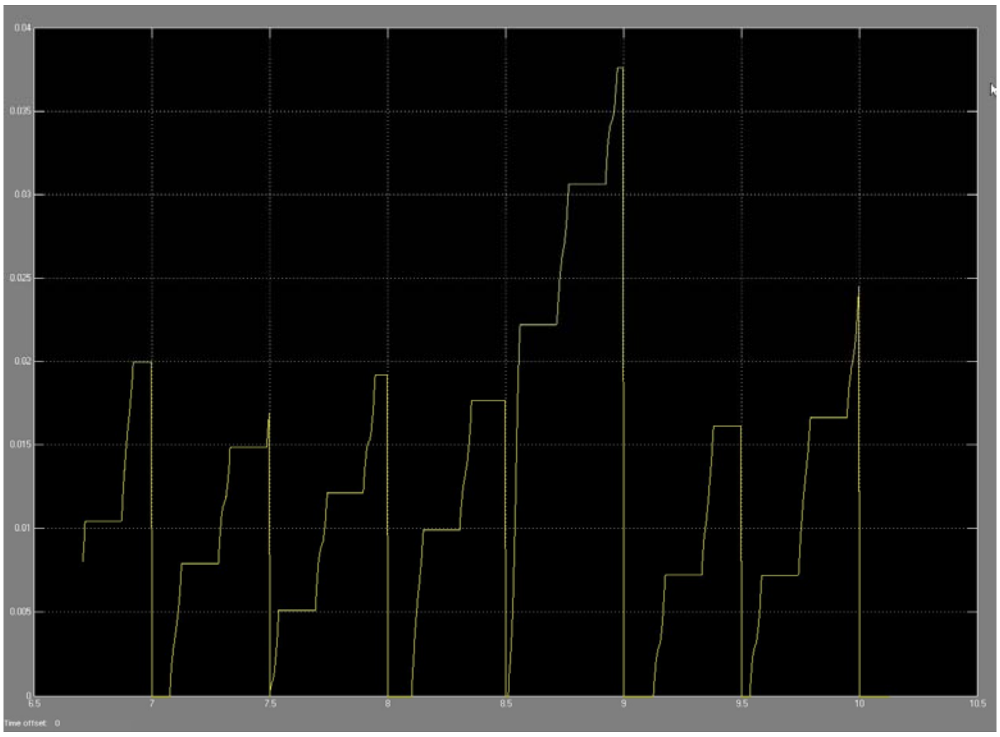


Figure 3.37. The slippage distance on 2N friction environment. The total amount of slippage does not exceed 0.04 cm for 0.5 seconds.

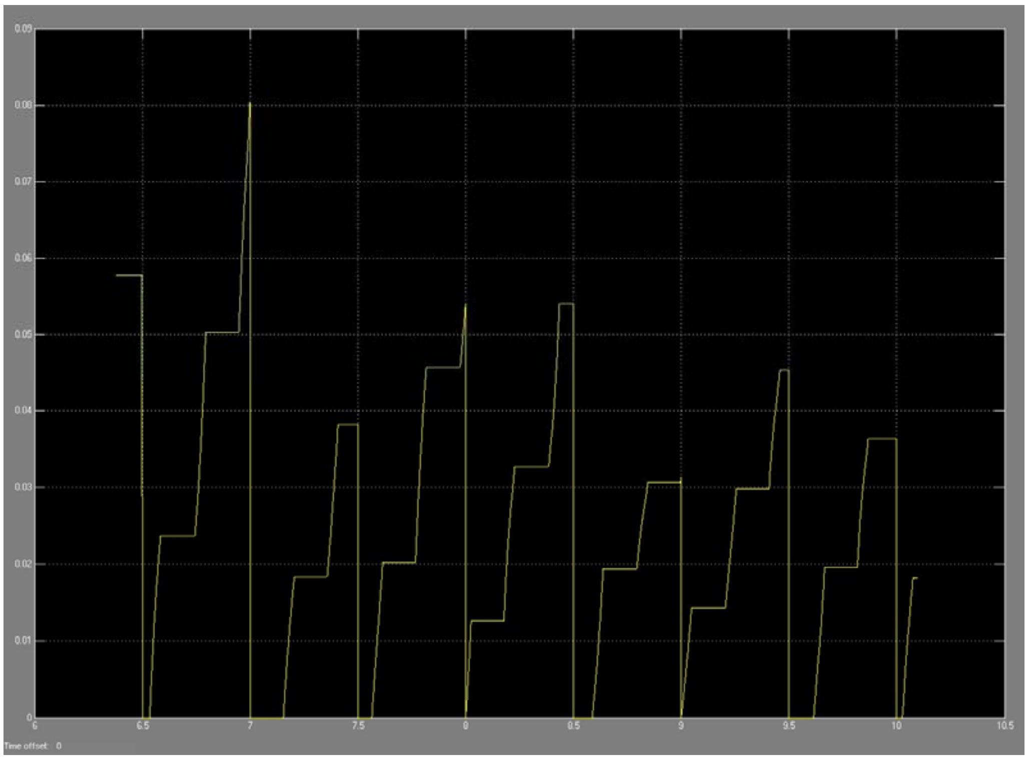


Figure 3.38. The slippage distance on 0.02N friction environment. The total amount of slippage does not exceed 0.08 cm for 0.5 seconds.

Results show that rectilinear gait can be conducted on very slippery surfaces. As the friction force of the environment is decreased from 2 to 0.02N, the slippage distance only doubles. For example, even a 0.02N friction demonstrates a slippage distance of only 0.08 cm which is a negligible distance compared to the size of the robot.

The slippage distance of the rectilinear gait is smaller than the accordion gait because for rectilinear gait, half of the body parts are raised eliminating slippage. The translation of prismatic joints is the main source of slippage. It should be noted that as the friction drops, the forward speed of the rectilinear gait is greatly reduced; due to insufficient friction the stationary segments repel back as the lifted sections are advancing. This can also be considered as slippage but it does not result in unwanted heading angle change and therefore it is not taken into account within the slippage analysis.

3.4.1.3. Lateral Undulation

The visualization of the snake-like robot model performing lateral undulation gait over a terrain with 5 N friction is illustrated in figure 3.39. All segments touch the ground, and forward motion is obtained by use of tip end actuators shown with red lines in figure 3.39.

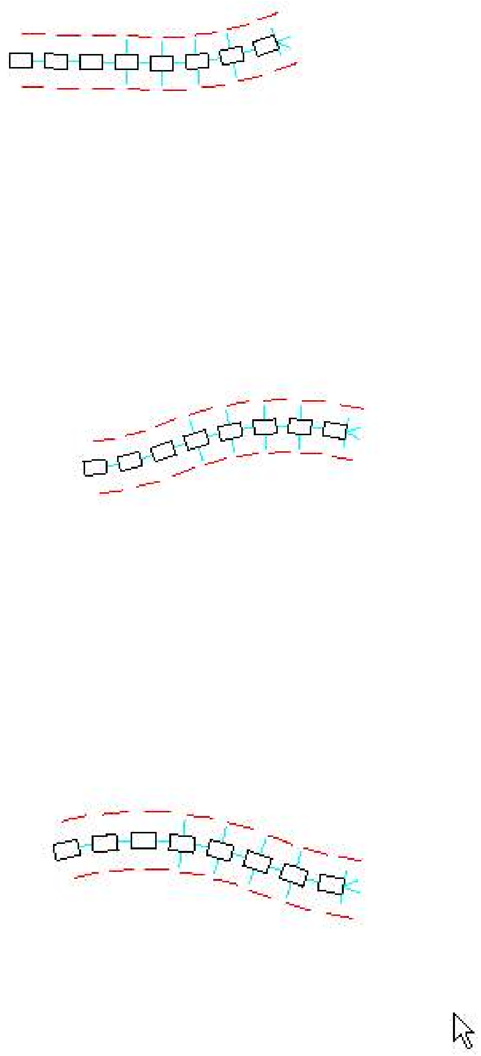


Figure 3.39. Lateral undulation in environment with 5N friction. The undulation amplitude is a simulation parameter. The undulation amplitude and the frequency of the simulation are selected for obtaining the best curvature of the body. Higher amplitude and frequency of undulation requires smaller body segments.

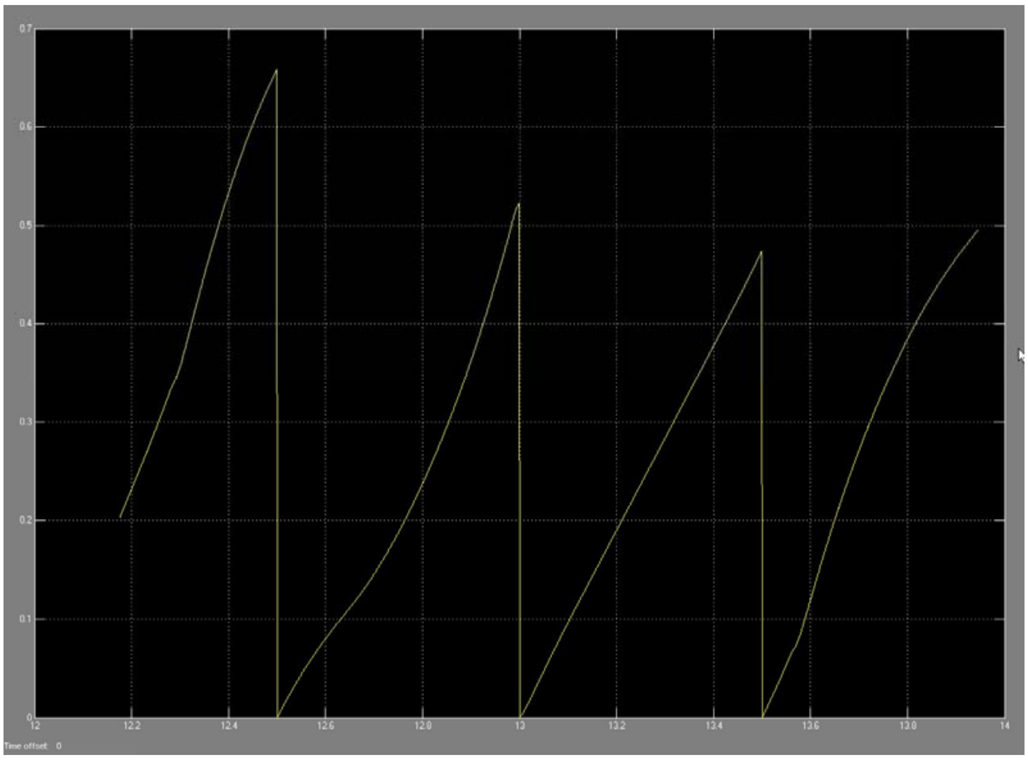


Figure 3.40. Slippage of LU on 20N friction. The peak of slippage distance is about 0.65cm in 0.5 seconds.

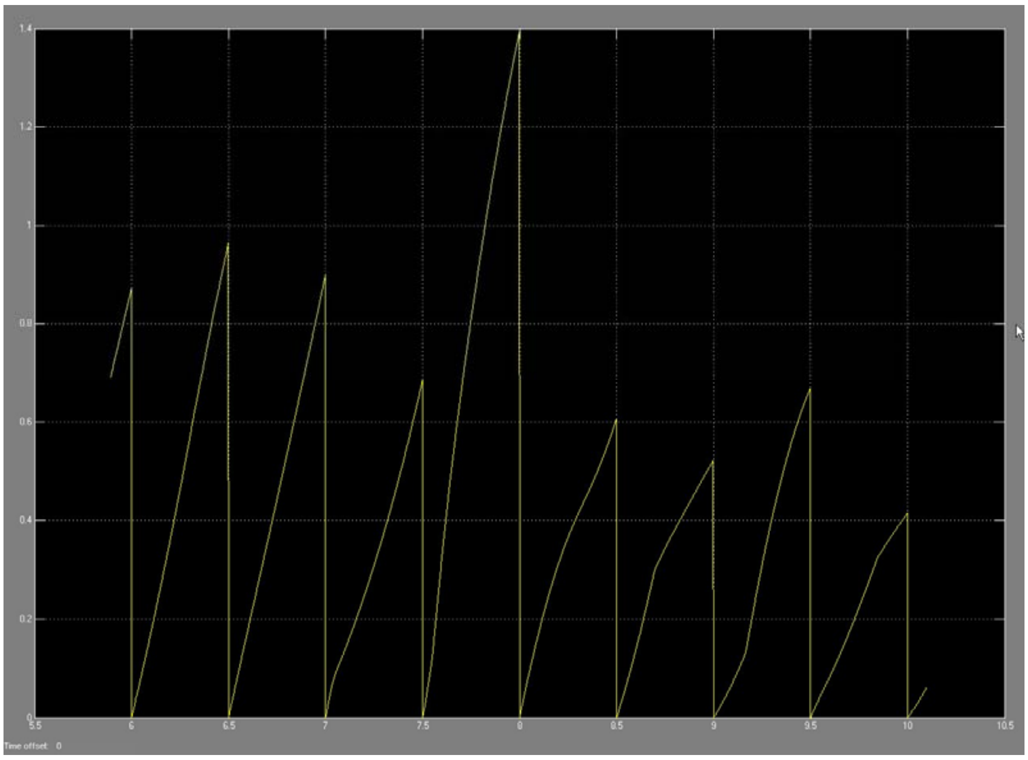


Figure 3.41. Slippage of LU on 2N friction. The peak of slippage distance is about 1.4 cm in 0.5 seconds which occurred during sharp steering.

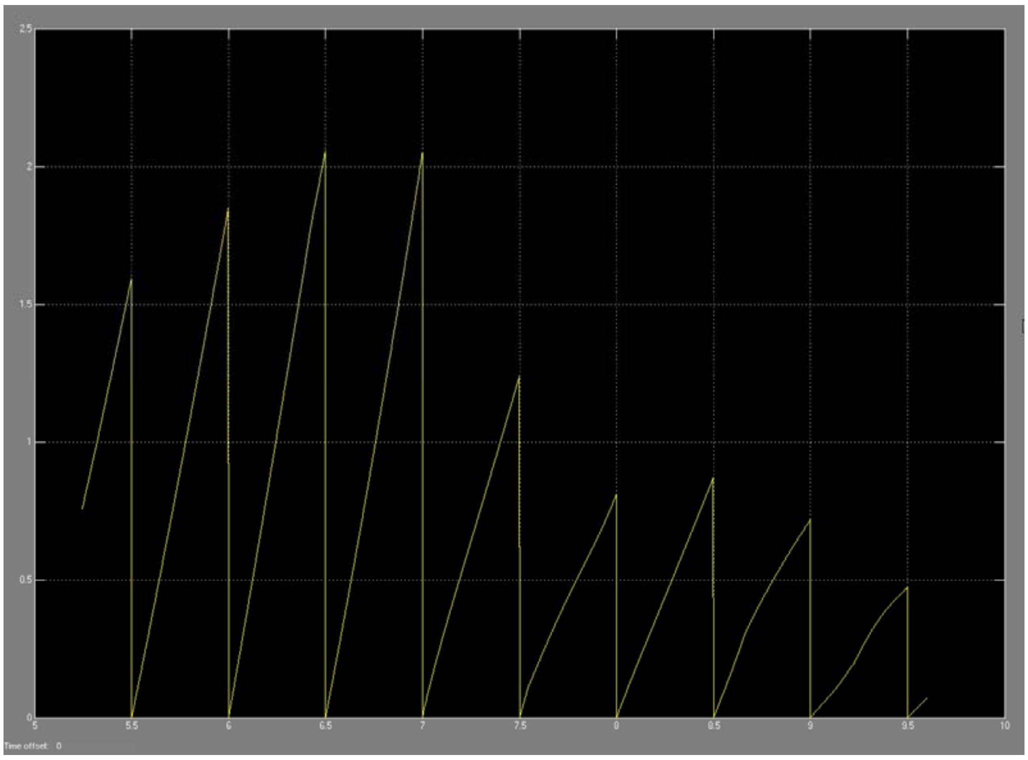


Figure 3.42. Slippage of LU on 0.02N friction. The peak of slippage distance is about 2.1 cm in 0.5 seconds.

During locomotion using lateral undulation gait on a 0.02N frictional areas, the robot slips 2 cm in half second, which is also noticeable with naked eye. This slippage is not unexpected for this gait since the linear speed of the robot is the double than that for the accordion and rectilinear gaits. Also in lateral undulation gait, during sharp steering the body segments may push each other in the y axis direction.

However if the friction is around 20N, the lateral undulation gait results with slippage distances similar to that of the accordion gait. The worst case scenario used in the analysis let to a sharp turn caused a maximum of 0.65cm/0.5sec slippage but it must be noted that less sharp steering will result in less slippage. Also the undulation amplitude and the frequency also effects slippage where undulations will less amplitude and lower frequencies will decrease the amount of slippage.

3.4.1.4. Sidewinding

The visualization of the snake-like robot model performing sidewinding gait to its right side over a terrain with 5 N friction is illustrated in figure 3.43. The body touches the ground only on two segments which are indicated by solid black borders in figure 3.43. Forward motion is obtained by use of the actuators of the segments which are touching the ground.

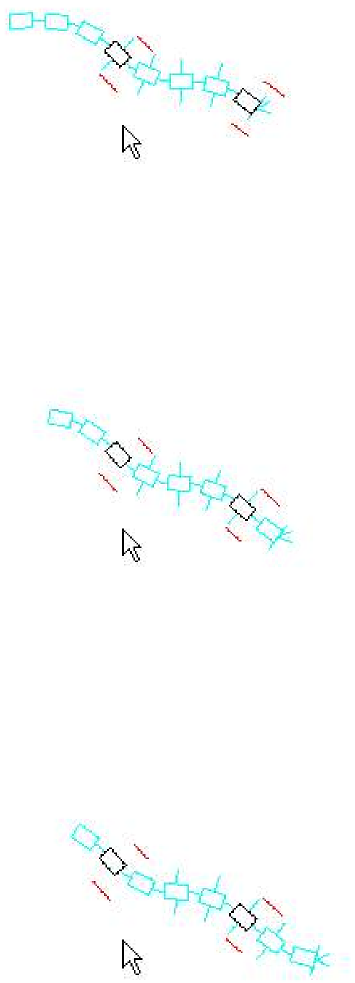


Figure 3.43. Sidewinding in environment with 10N friction. The body moves in the direction of the segments touching the ground.

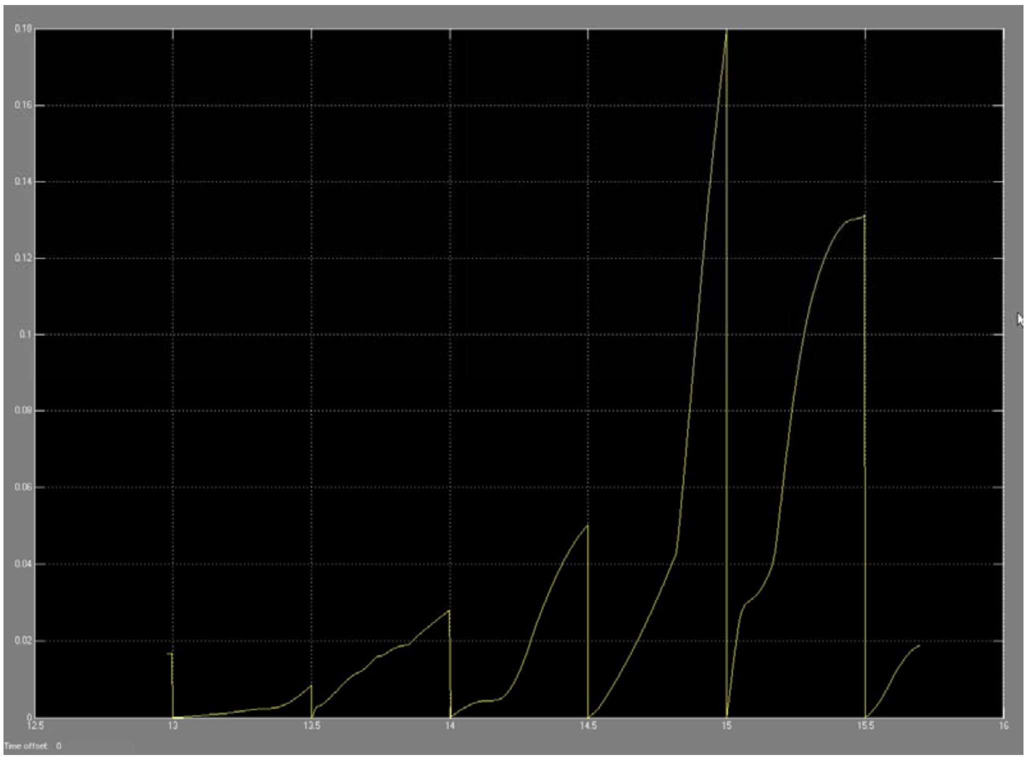


Figure 3.44. Slippage of SW on 20N friction. The peak of slippage distance is about 0.18 cm in 0.5 seconds.

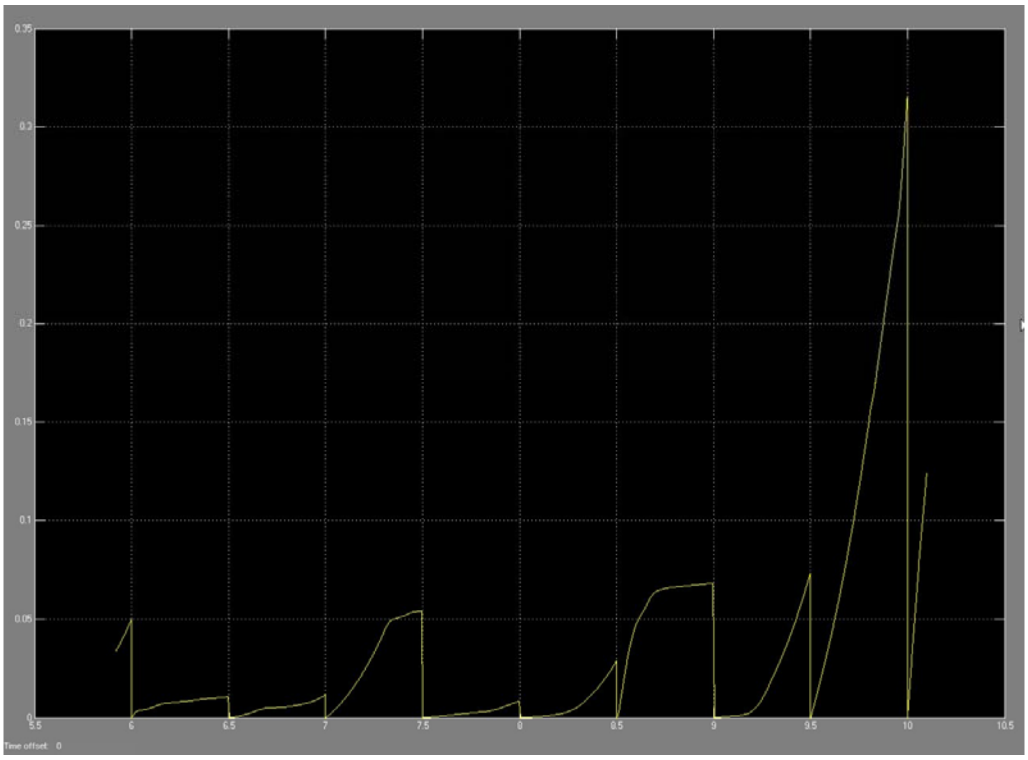


Figure 3.45. Slippage of SW on 2 N friction. The peak of slippage distance is about 0.35 cm in 0.5 seconds.

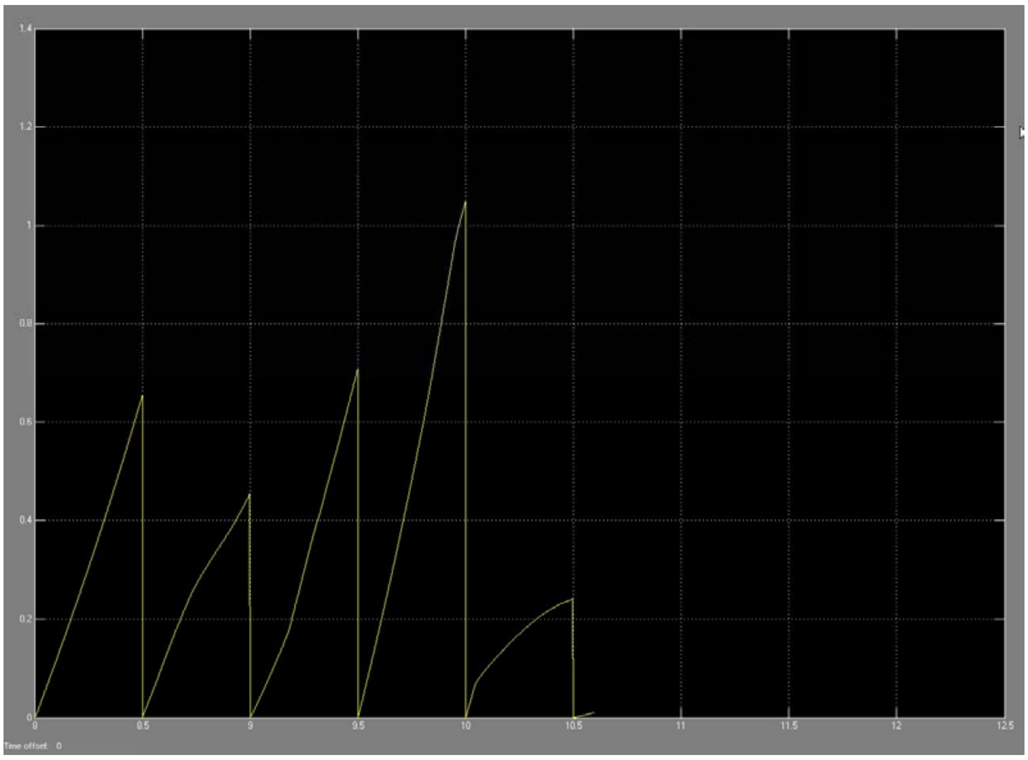


Figure 3.46. Slippage of SW on 0.02 N friction. The peak of slippage distance is about 1 cm in 0.5 seconds.

Since sidewinding gait does not steer, so it has less slippage distance due to the lateral undulation. When the friction is around 20N, the slippage of the sidewinding is negligible. As the friction drops the slippage of the sidewinding increases because only two (sometimes three) of the body segments receives friction from the ground. Even though the snake does not steer, the forces generated by the inertia of the body segments can overcome the friction forces acting only on two of the body segments.

3.4.1.5 Threshold Determination

The maximum slippage distance per 0.5 seconds does not exceed 2cm for lateral undulation whereas it always below 0.3cm for accordion and rectilinear. The threshold for determination of the slippage is considered to be 0.9 cm/0.5 sec. The slippage distances higher than 0.9cm in 0.5 seconds will be considered as slippage and result in a “-1” reinforcement for the high level controller.

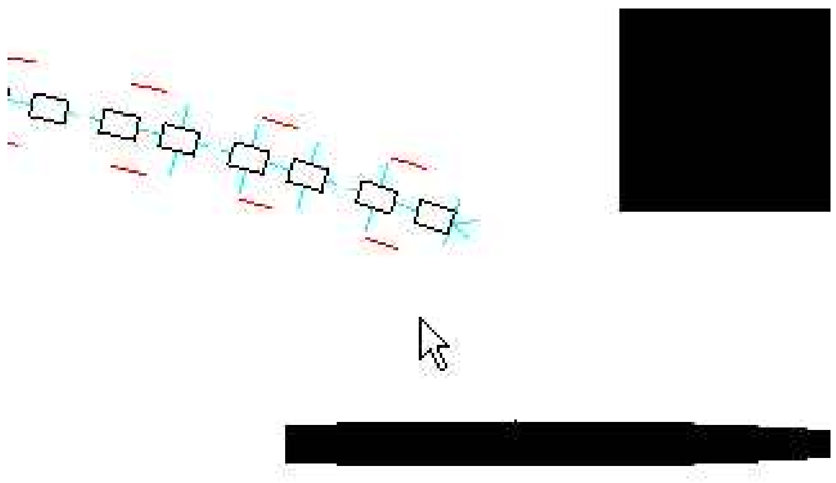
Since rectilinear and accordion gaits have superior friction performance than lateral undulation and sidewinding, these gaits could be performed at every part of the environment. But the speeds of rectilinear and accordion gaits are half of the other gaits; so when applicable performing lateral undulation and sidewinding is preferred. To accomplish this preference, the positive rewards of lateral undulation and sidewinding gaits are doubled to “+2” if they do not skip and stay away from the obstacles.

3.4.2. Simulation Results with Obstacle Avoidance

Accordion, rectilinear and lateral undulations gaits are performed with their respective already trained obstacle avoidance controllers on environments with different obstacle distribution.

3.4.2.1. Accordion

Accordion gait with “obstacle avoidance” behavior is illustrated in figures 3.47 and 3.48. The environment in the figures 3.47 and 3.48 5 provides a constant 5N friction and has obstacles.



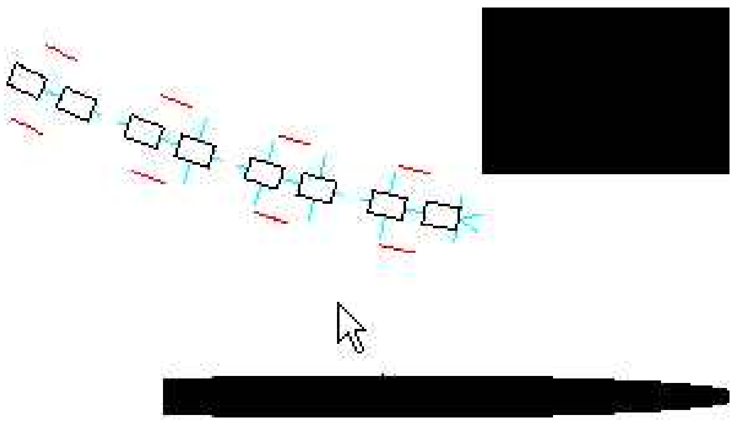


Figure 3.47. Accordion for Obstacle Avoidance.

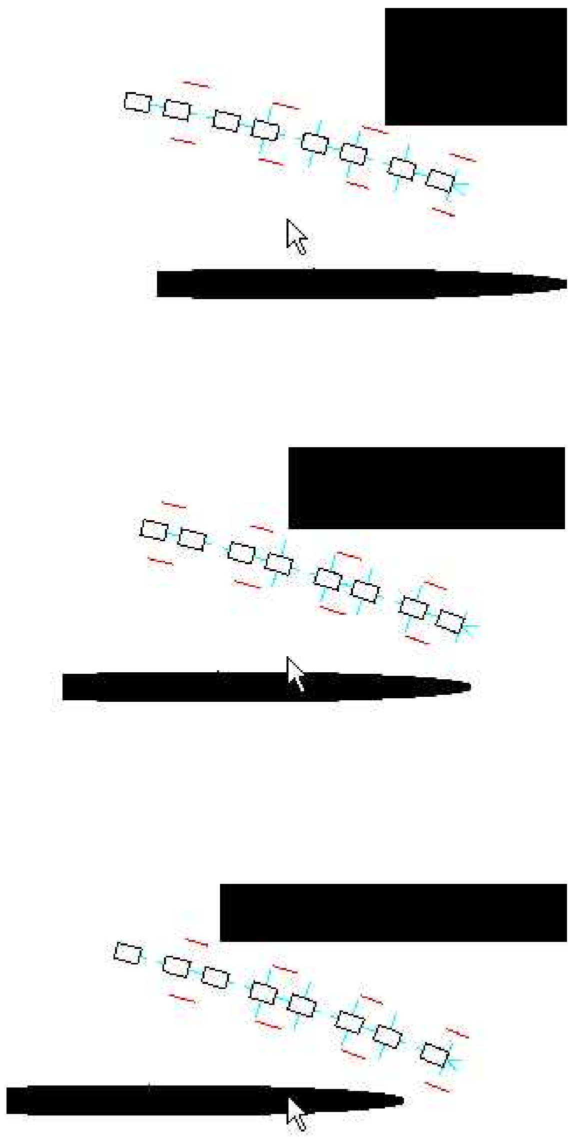


Figure 3.48. Accordion for Obstacle Avoidance.

The simulation model propagates forward by performing accordion gait while avoiding the obstacles on its way. In figures 3.47 and 3.48 the model goes through two nearby obstacles without colliding, which proves the success of the accordion gait's "obstacle avoidance" controller.

3.4.2.2. Rectilinear

Rectilinear gait is performed with its “obstacle avoidance” behavior in figures 3.49 and 3.50. The environment provides a constant 5 N friction and contains obstacles.

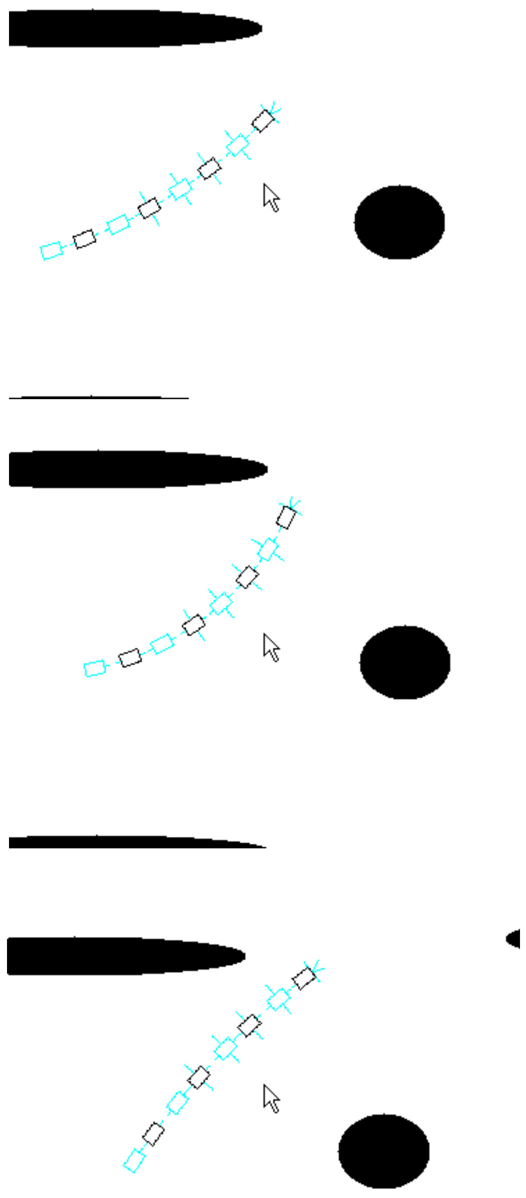


Figure 3.49. Accordion for Obstacle Avoidance

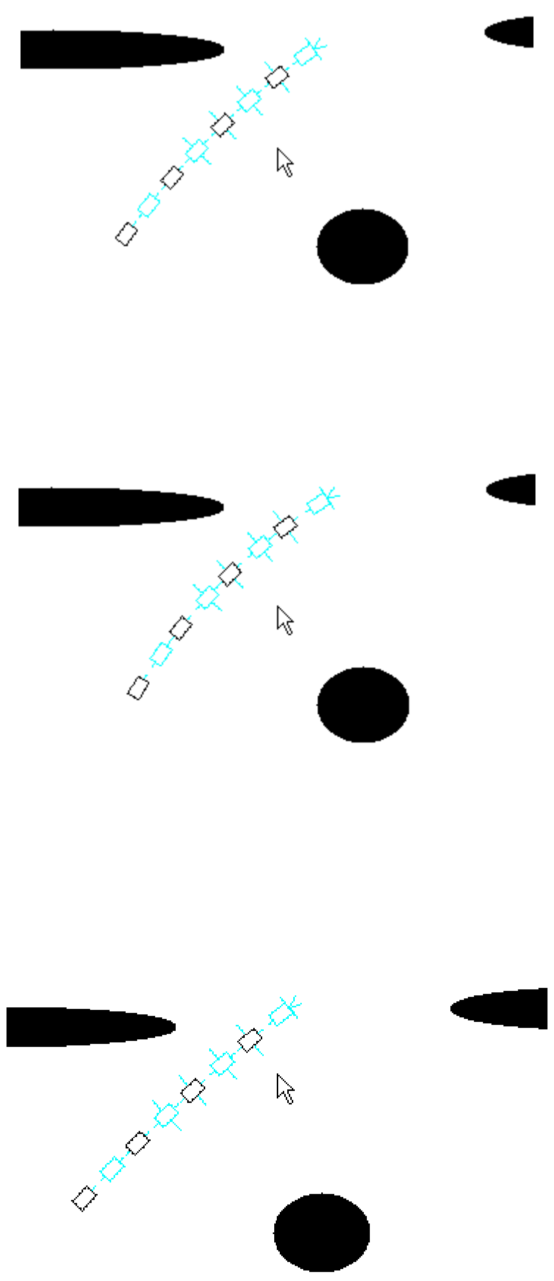


Figure 3.50. Accordion for Obstacle Avoidance (continued)

As seen in figures 3.49 and 3.50, obstacle avoidance controller of rectilinear gait successfully manages to steer the simulation model through an area surrounded by obstacles.

3.4.2.3. Lateral Undulation

Lateral undulation gait with obstacle avoidance behavior is illustrated in figures 3.51 and 3.52. The environment provides a constant 5 N friction and contains randomly distributed obstacles.

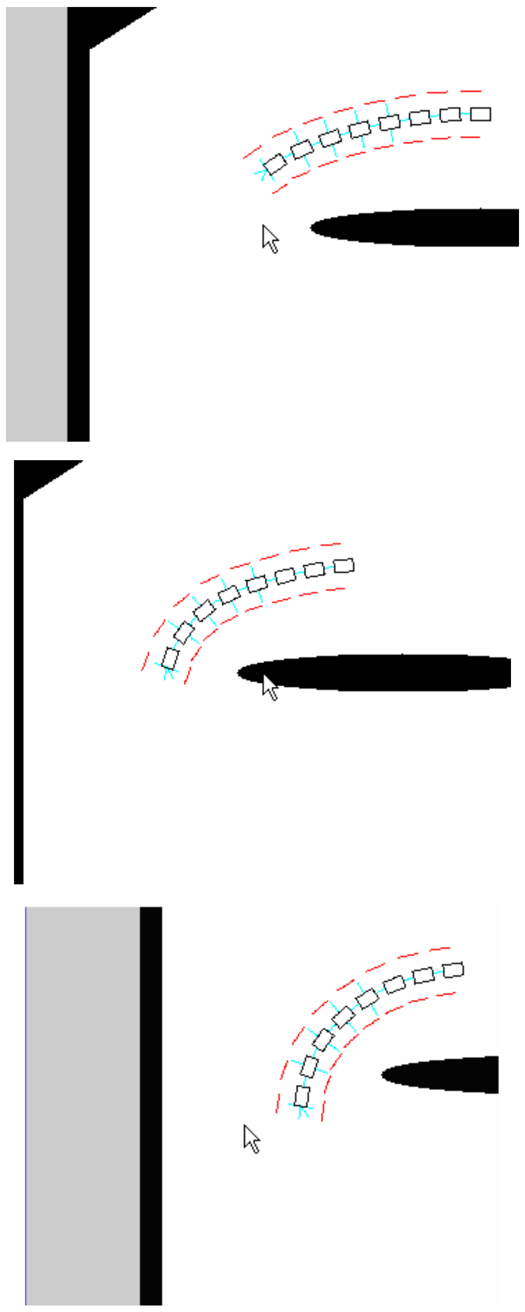


Figure 3.51. Lateral Undulation with obstacle avoidance.

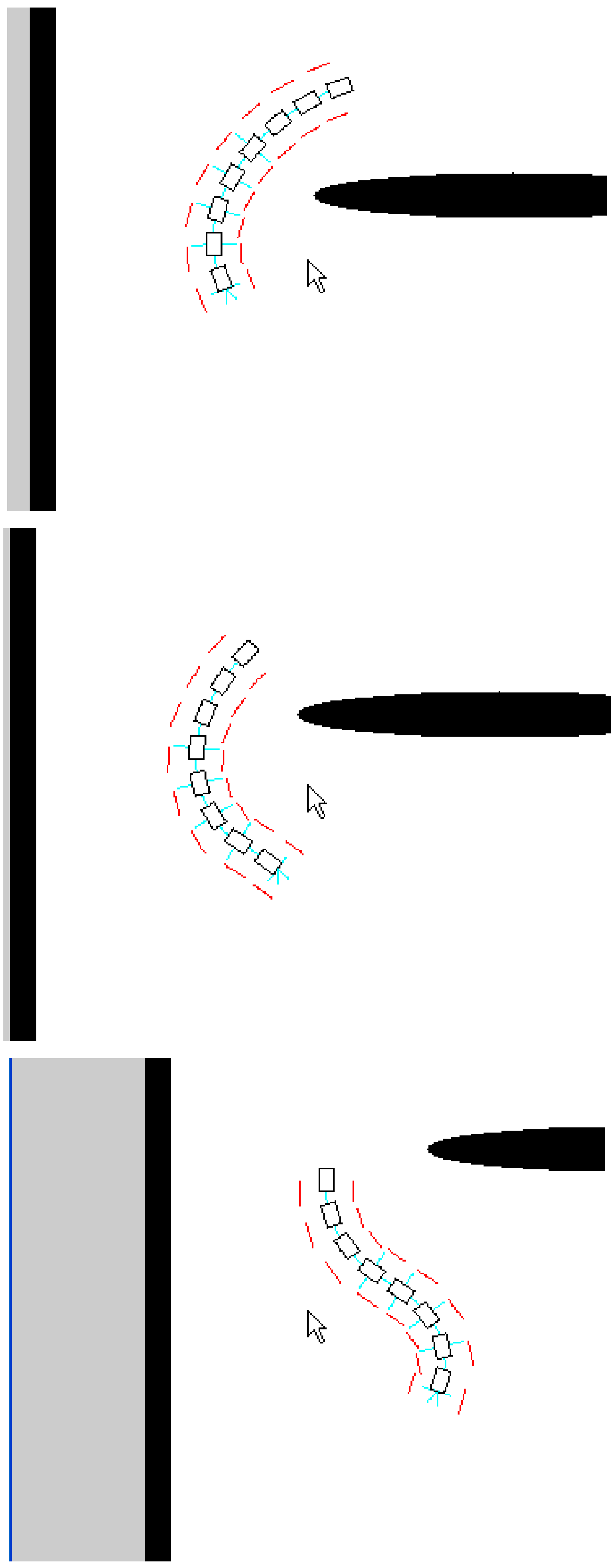


Figure 3.52. Lateral Undulation with obstacle avoidance (continued).

Lateral undulation gait's obstacle avoidance controller manages to steer the body to avoid the obstacles successfully as seen in figures 3.51 and 3.52.

3.4.3. Simulation Results with Arbitrary Environment

Simulation result of the model in an environment which has two regions with different frictions and randomly distributed obstacles is illustrated through figures 3.53 – 3.67. Gray region of the environment has 0.4N friction and white region has 5 N friction. Since the environment does not contain any objects, all controllers and preset actions related with objects are not activated. Simply all gaits with their corresponding obstacle avoidance controllers (except sidewinding) are combined by a high level controller which makes selections among available gaits. Although sidewinding gait does not have an obstacle avoidance controller, it is still available to the high level controller as mentioned in section 3.3.

The reason for using an environment containing two regions with distinct frictions is clearly illustrating gait preference of the high level controller.

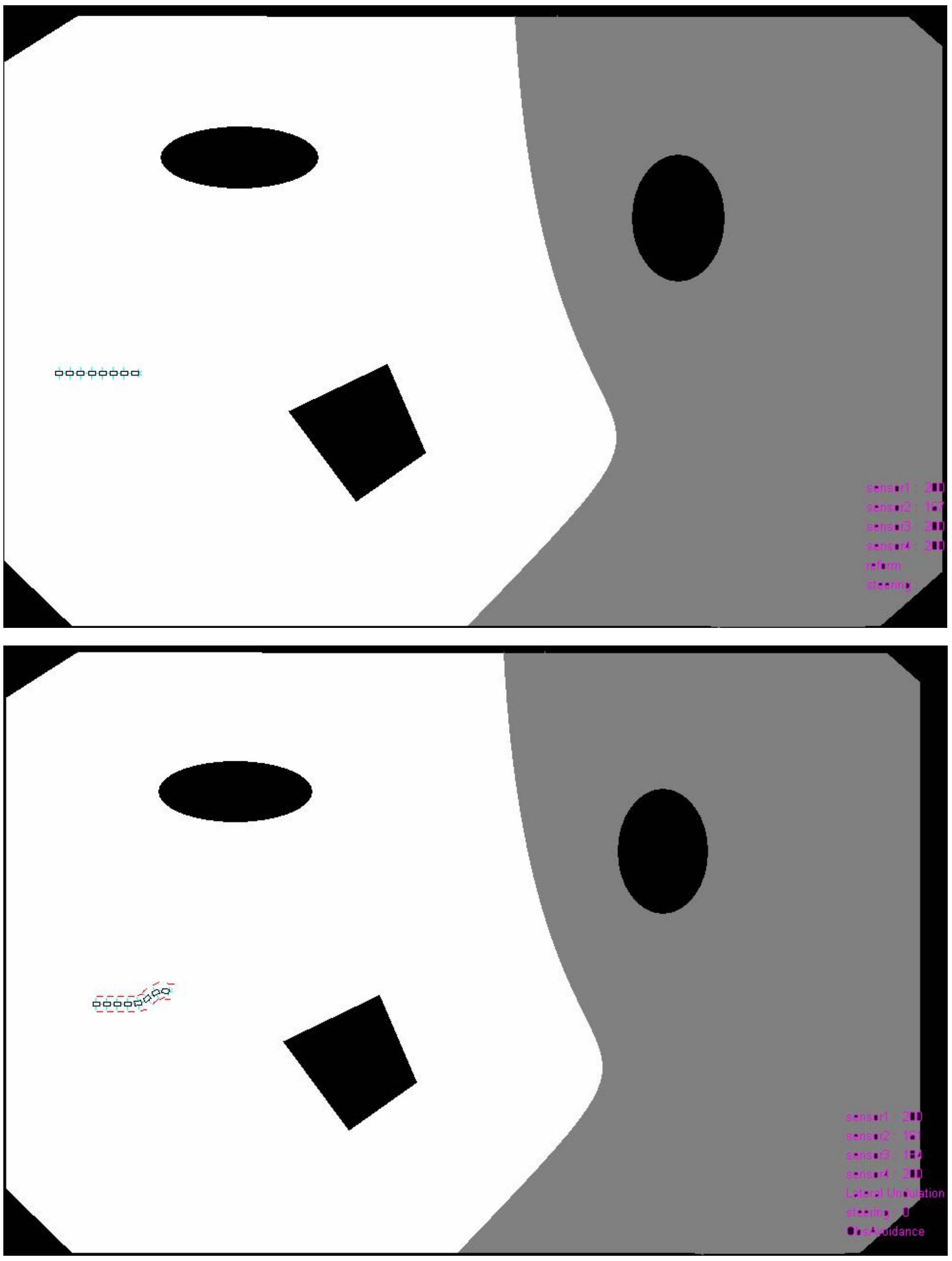
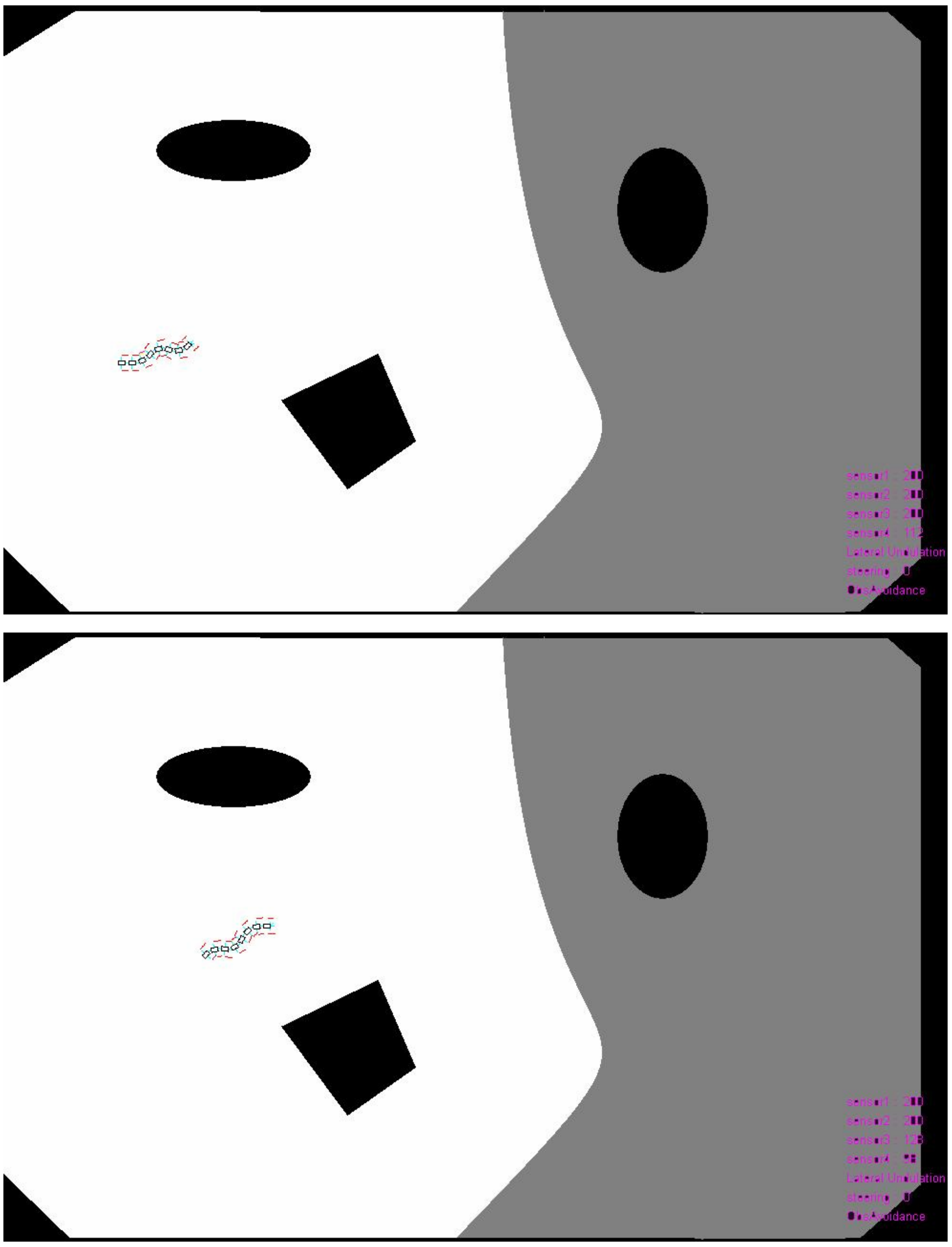


Figure 3.53. Navigation in a variable friction environment. The model begins with performing lateral undulation by avoiding the obstacle on its front.


```
sensor1 : 200
sensor2 : 200
sensor3 : 200
sensor4 : 112
Lateral Undulation
steering : 0
ObsAvoidance
```

```
sensor1 : 200
sensor2 : 200
sensor3 : 128
sensor4 : 98
Lateral Undulation
steering : 0
ObsAvoidance
```

Figure 3.54. The model continues lateral undulation and avoids the frontal obstacle.

In the figures 3.53 and 3.54, the high level controller selects lateral undulation gait and the model avoids the obstacle in the front of the model's head.

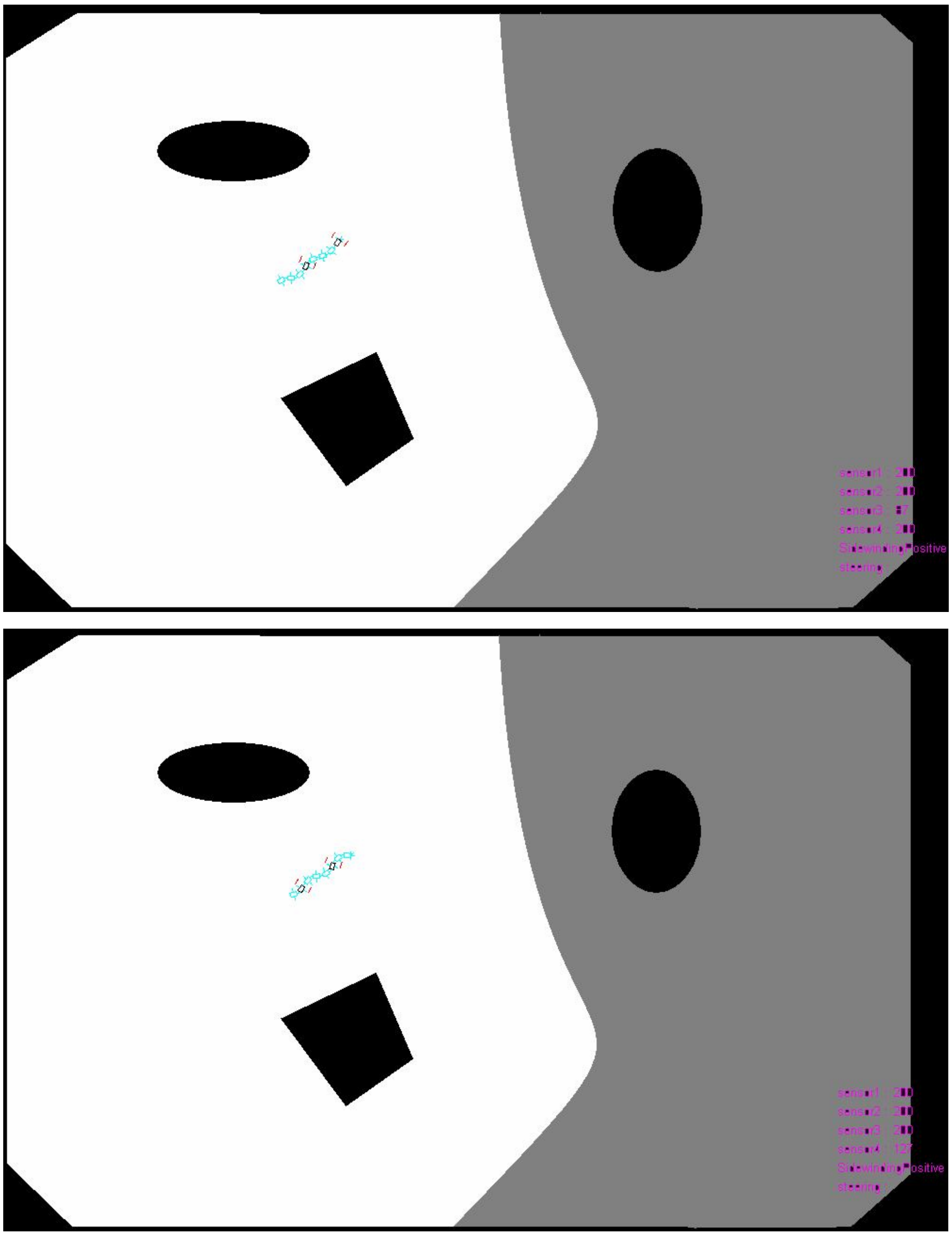


Figure 3.55. The model begins to perform sidewinding gait against its left side (with positive slope), since the head is cleared of obstacles, sidewinding is applicable.

After clearing the head of model from the previously encountered obstacle, model performs sidewinding against its left side (with positive slope) to get away from the previously encountered obstacle while propagating forward as seen in figure 3.55.

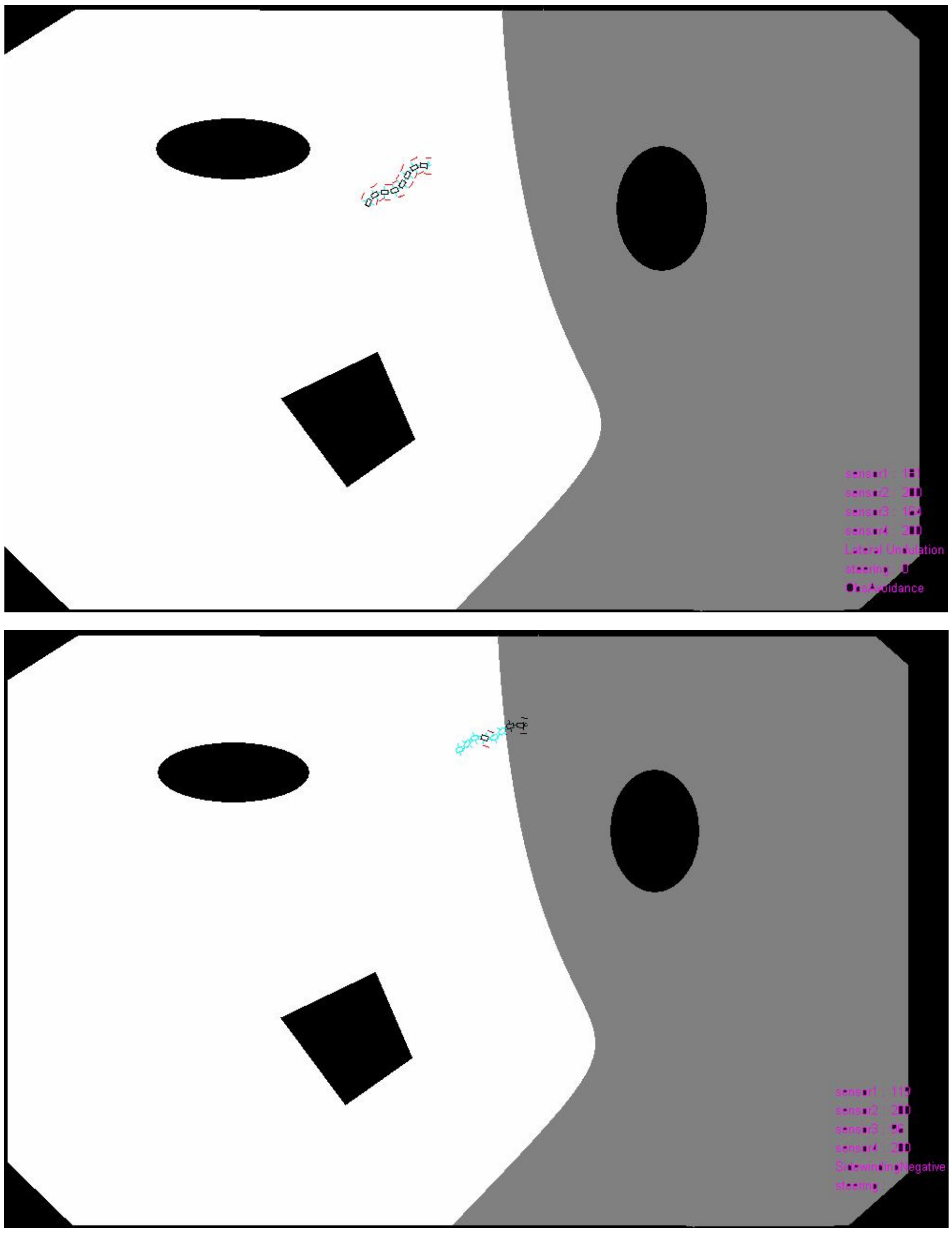
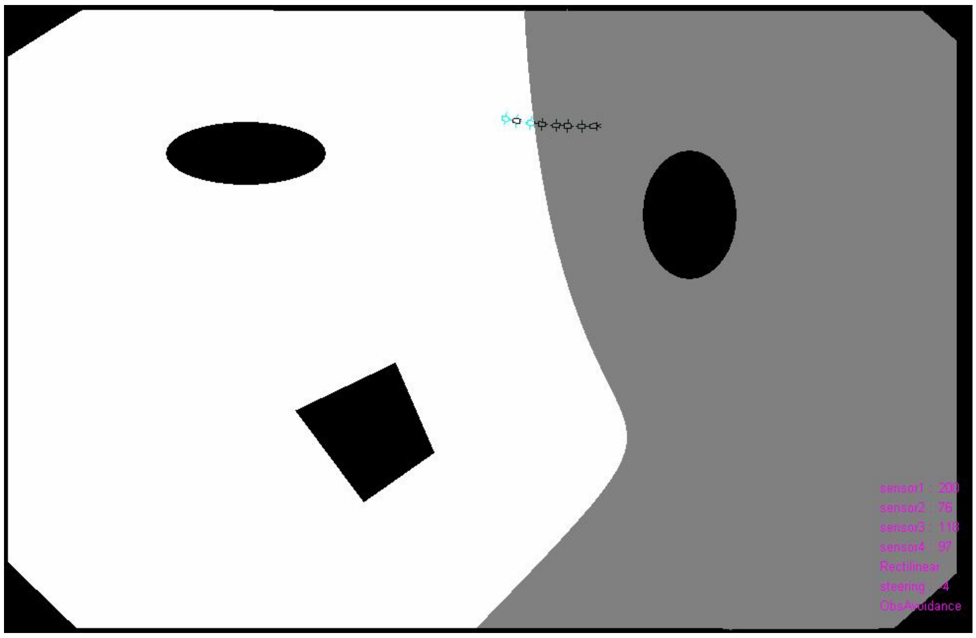


Figure 3.56. The model momentarily switches to lateral undulation and then selects sidewinding again but this time with negative slope.

The model senses the upper obstacle wall and temporarily switches to lateral undulation to avoid it. After the head of the model is clear from the obstacle wall, the model continues to perform sidewinding gait but this time against its right side (negative slope) as shown on figure 3.56. Sidewinding against right side allow the model to get away from the upper obstacle wall.



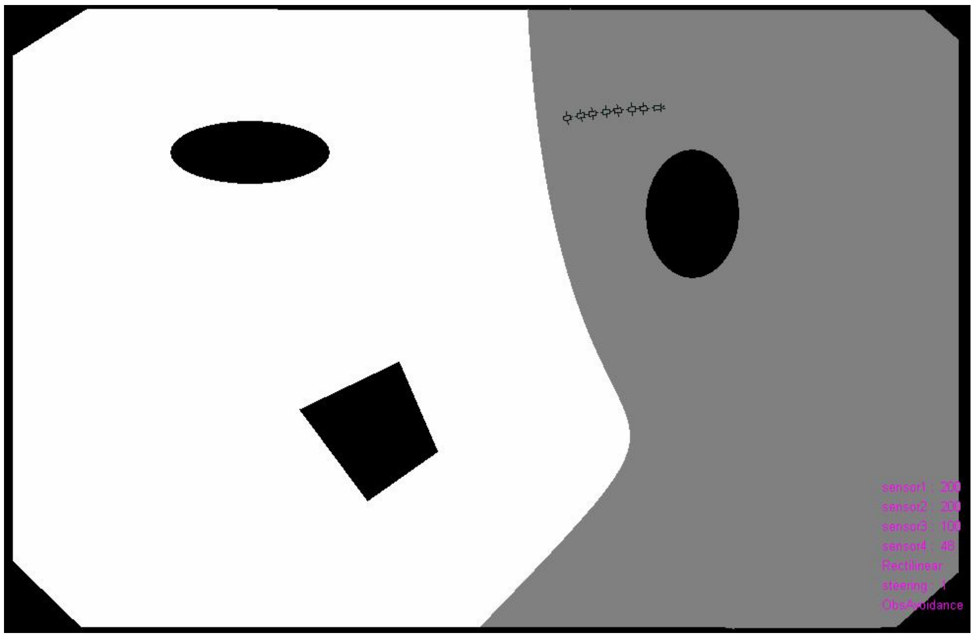


Figure 3.57. The model begins to perform rectilinear gait on the low friction portion of the environment.

As the model passes to the region with low friction (0.4 N), it switches to rectilinear gait. Rectilinear gait's obstacle avoidance controller makes the necessary steering in order to prevent collision with obstacles as seen in the figure 3.57.

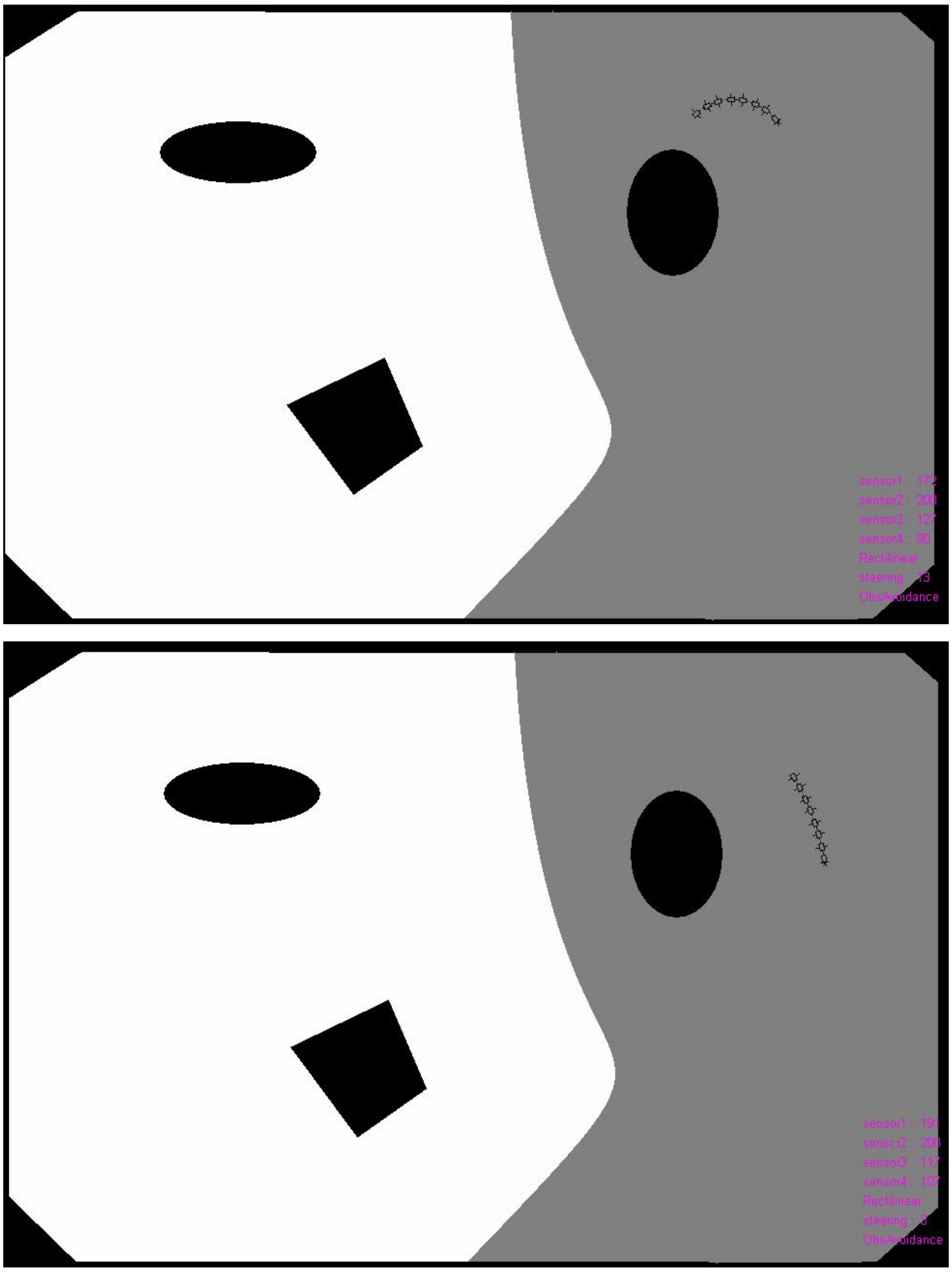
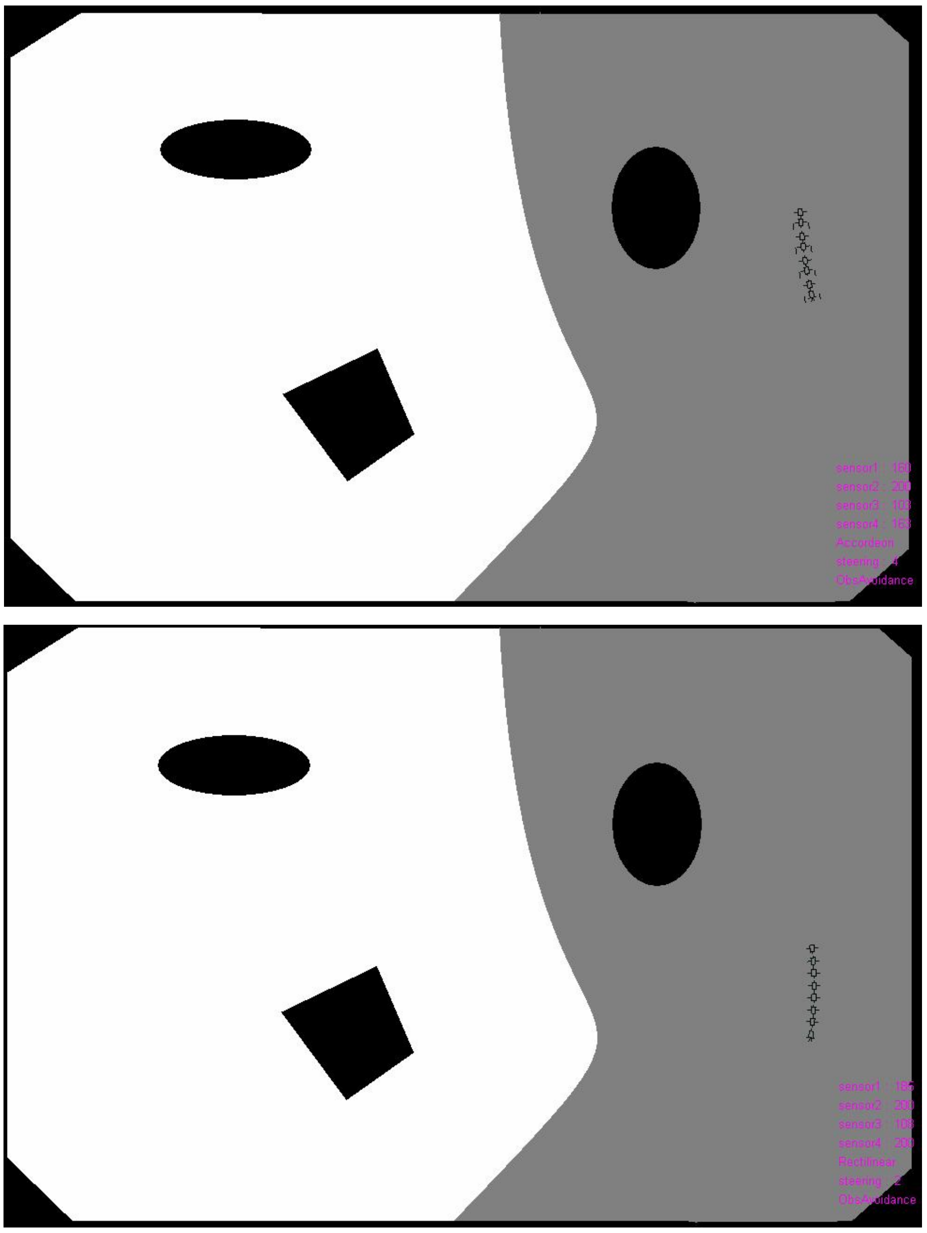


Figure 3.58. The model continues to perform rectilinear gait and avoids the obstacles.

The model continues to perform rectilinear gait while successfully avoiding obstacles as shown in figure 3.58.

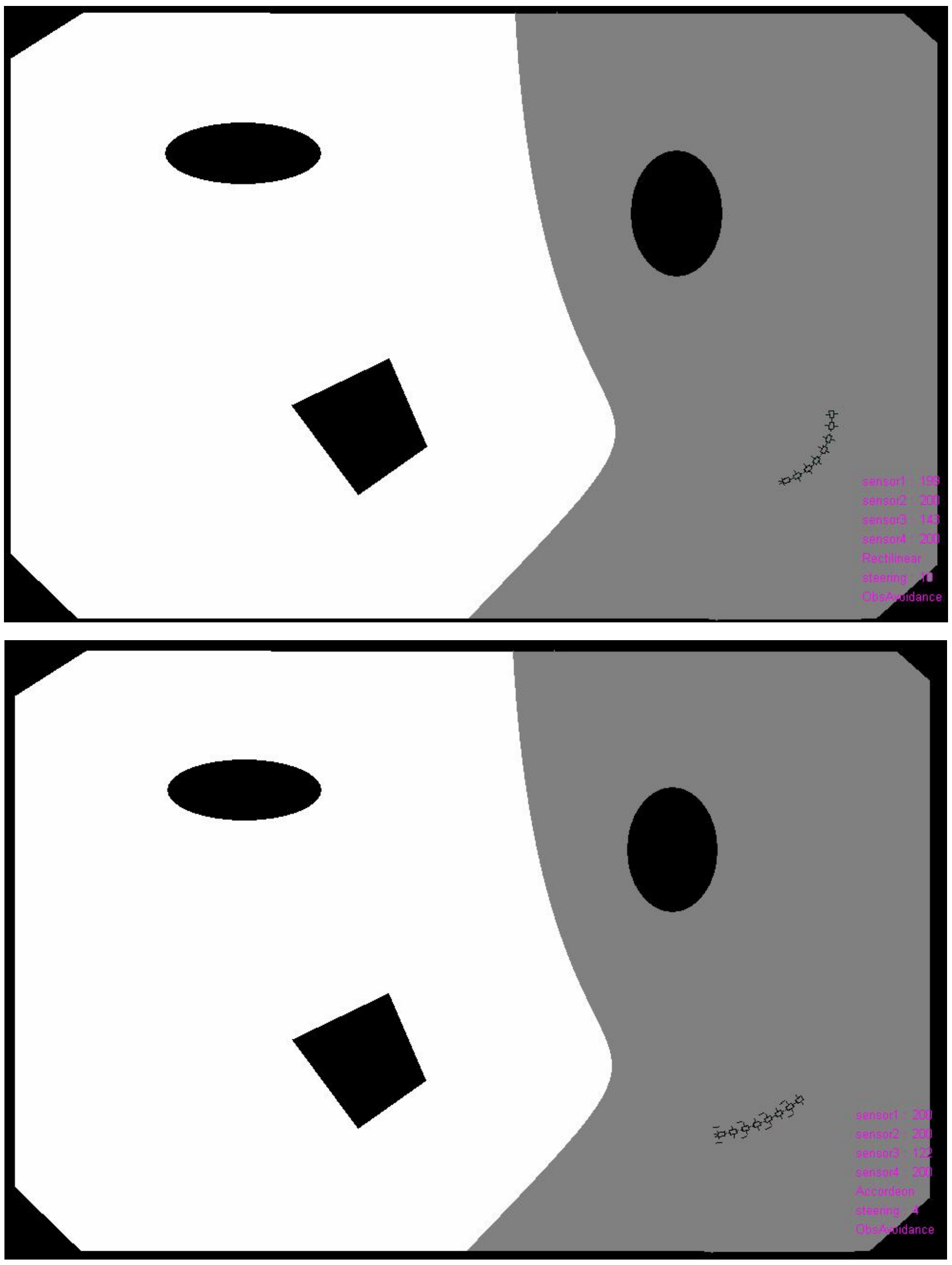


```
sensor1 : 160
sensor2 : 200
sensor3 : 103
sensor4 : 163
Accordeon
steering : 4
ObsAvoidance
```

```
sensor1 : 185
sensor2 : 200
sensor3 : 108
sensor4 : 200
Rectilinear
steering : 2
ObsAvoidance
```

Figure 3.59. The model momentarily switches to accordion gait, and then it continues with rectilinear gait. But high level controller prefers to select rectilinear gait more in slippery portion of the environment. This preference may result from the better obstacle avoidance performance of the rectilinear gait's middle level controller.

As the model advances in the region with low friction, it momentarily switches to accordion gait, but then continues with rectilinear gait as seen in figure 3.59. Both gaits have obstacle avoidance controllers, so the obstacle avoidance behavior is maintained during gait change.



```
sensor1 : 199
sensor2 : 200
sensor3 : 143
sensor4 : 200
Rectilinear
steering : 10
ObsAvoidance
```

```
sensor1 : 200
sensor2 : 200
sensor3 : 122
sensor4 : 200
Accordeon
steering : 4
ObsAvoidance
```

Figure 3.60. Model continues its navigation by rectilinear, on its way it again switches to accordion gait momentarily. On both gaits, the model shows obstacle avoidance behavior.

The oscillations in the selection of the gaits shown on figures 3.59 and 3.60 prove the existence of exploration of the high level controller. But rectilinear gait is preferred more than accordion gait although they have similar properties. This bias may be resulted from insufficient exploration or better obstacle performance of rectilinear gait.

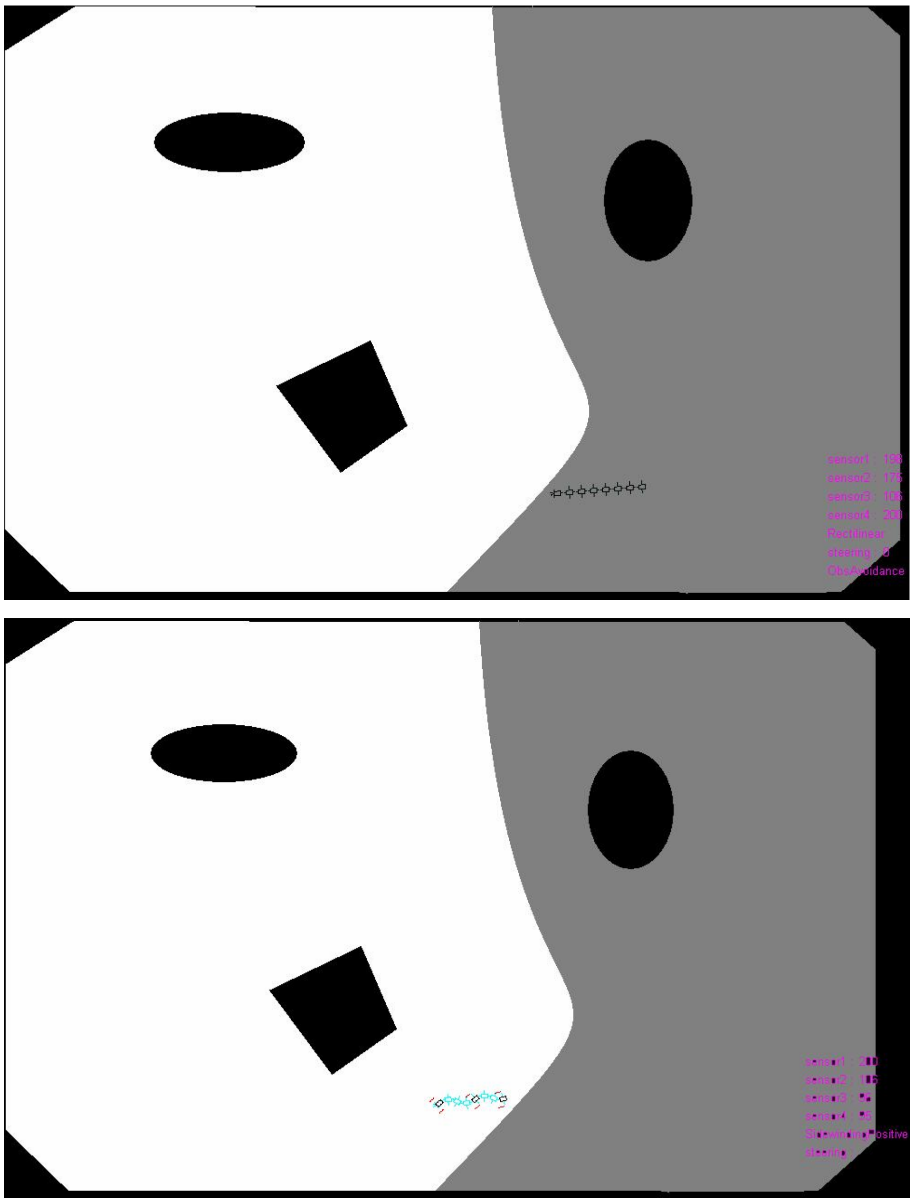
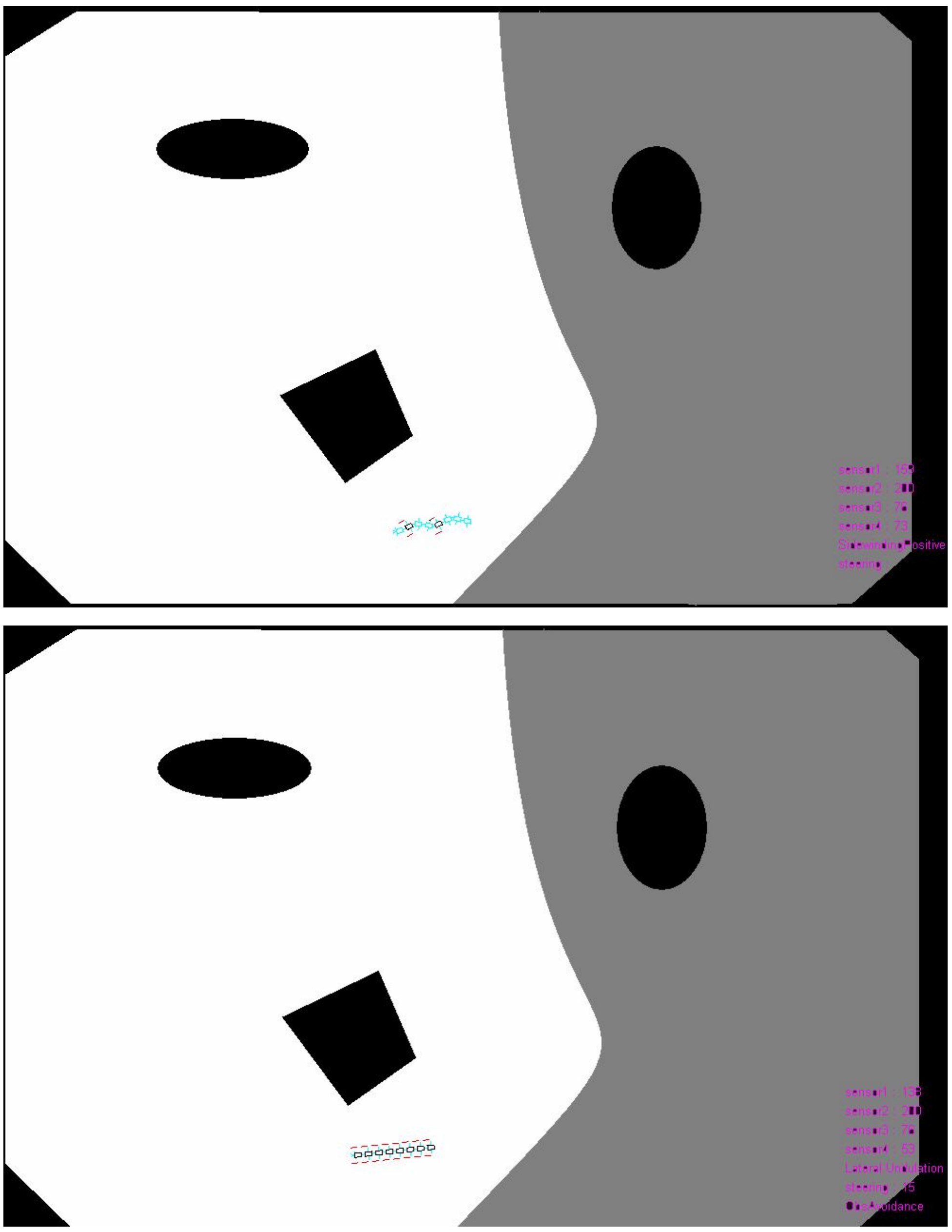


Figure 3.61. Model leaves the slipper portion of the environment with rectilinear gait. Model selects sidewinding on the area with high friction. Although the front of the model is blocked by an obstacle, the high level controller has learned that performing sidewinding may be used to avoid the obstacles at certain orientations.

As the model leaves the region with low friction, it switches to sidewinding gait against its left side to pass between the two frontal obstacles. The selection represented in lower part of figure 3.61 shows that the high level controller found a suitable situation for sidewinding to avoid front obstacles although sidewinding did not posses any obstacle avoidance controller.



```
sensor1 : 159
sensor2 : 200
sensor3 : 76
sensor4 : 73
SidewindingPositive
steering
```

```
sensor1 : 138
sensor2 : 200
sensor3 : 76
sensor4 : 53
Lateral Undulation
steering : 15
ObsAvoidance
```

Figure 3.62. Model enters between the wall (surrounding wall is also considered as an obstacle) and the obstacle. Model performs lateral undulation when nearby obstacle are present on the sides of the robot.

When the model senses obstacles at both sides, it switches to lateral undulation as seen in lower part of the figure 3.62.

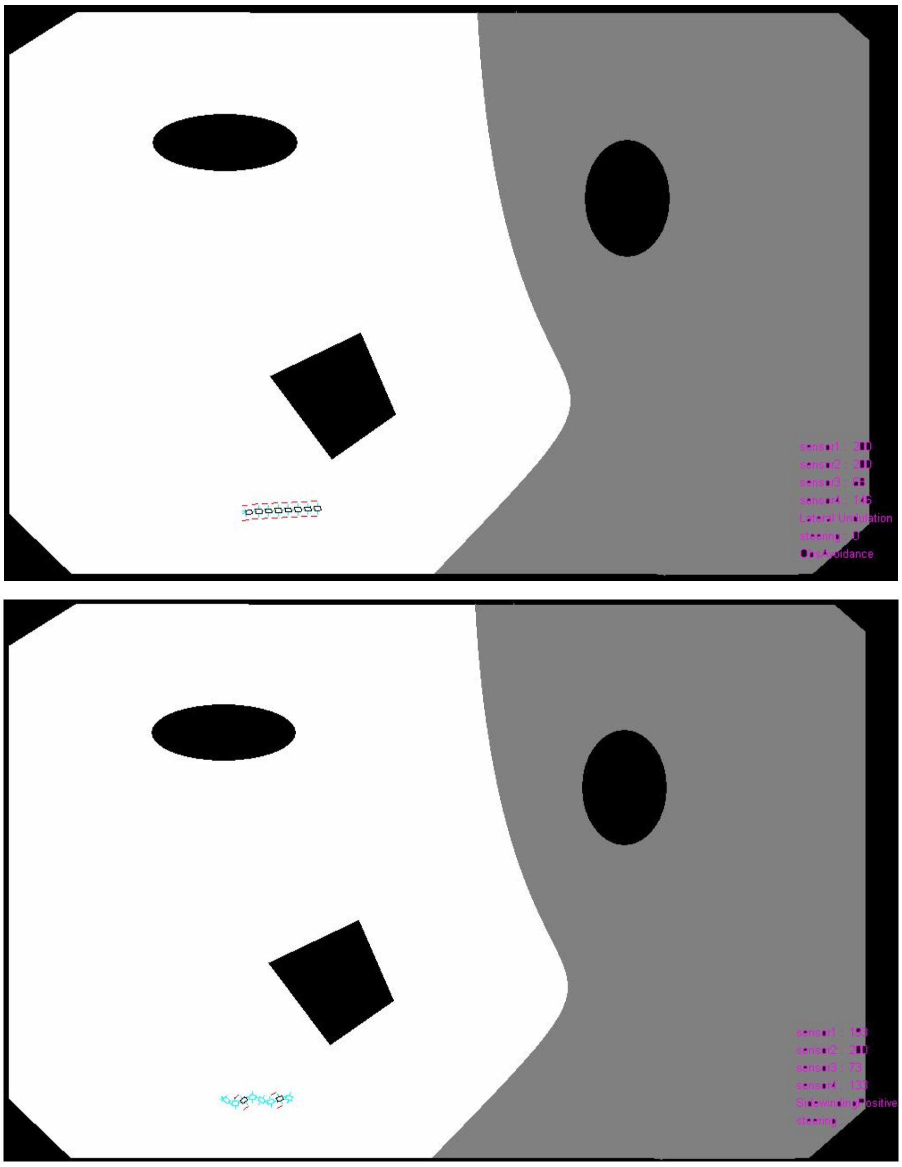


Figure 3.63. The model passes between the obstacles with lateral undulation gait. While passing between the obstacles, the amplitude of the undulation is suppressed by lateral undulation's obstacle avoidance controller. Since undulation moves the model nearer to one of the surrounding obstacles, the avoidance controller gives opposite steering eliminating the undulation. After the passing between the obstacles the model performs sidewinding.

As the model is passing between two obstacles, undulations of lateral undulation gait bring the model nearer to one of the obstacle. The obstacle avoidance controller of lateral undulation gait tires to avoid this situation by steering the model in the reverse direction of the undulation thus suppresses the undulations as seen in the upper part of figure 3.63.

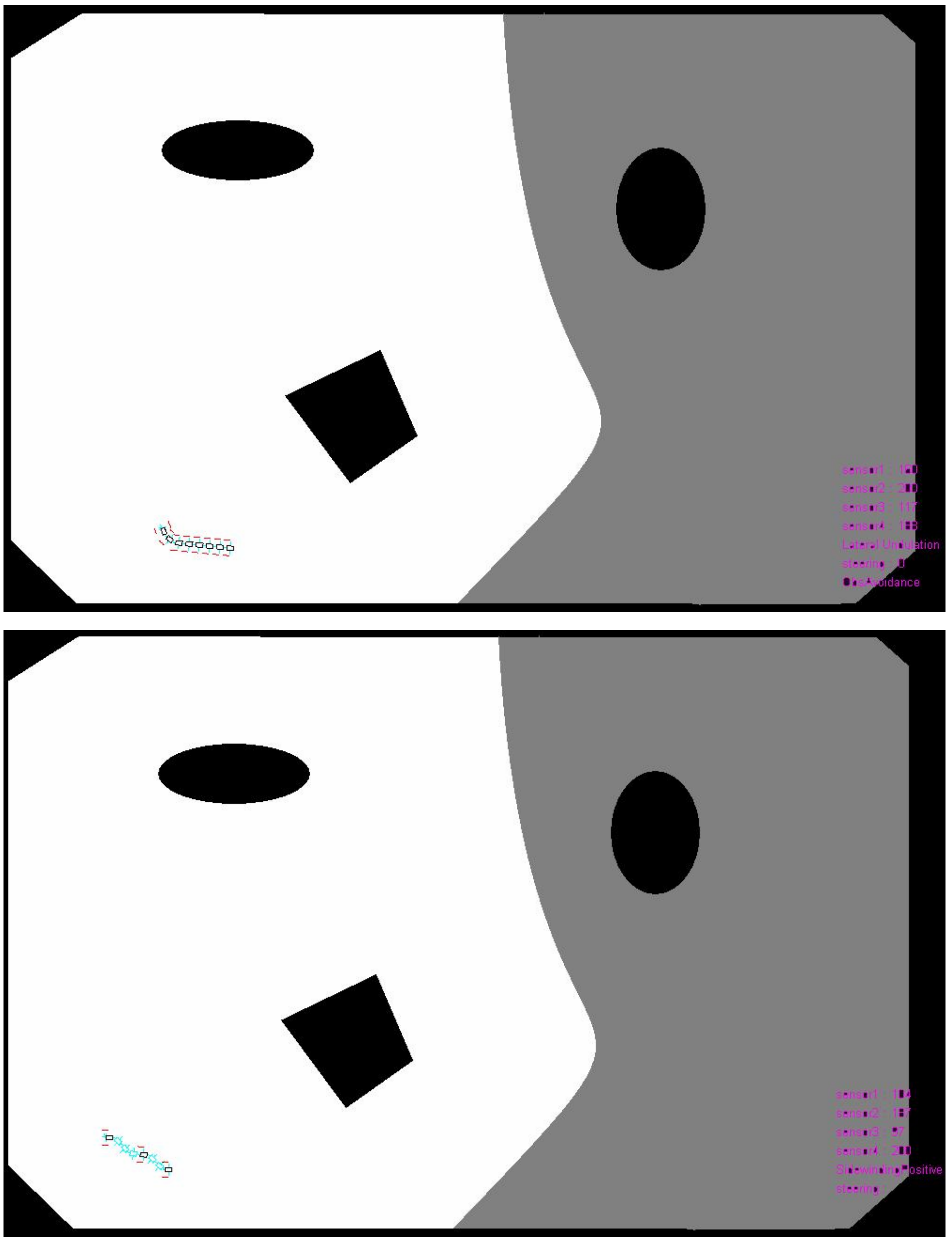


Figure 3.64. The model senses the corner of the environment and switches to lateral undulation to turn the head of the robot. After the head is clear of obstacles, robot continues to perform sidewinding.

The high level controller selects lateral undulation when the model encounters an obstacle in front of its head, sidewinding when only one side of the model encounters an obstacle. In figures 3.63 and 3.64, the model switches back and forth between lateral undulation and sidewinding gaits as the described situations alternates.

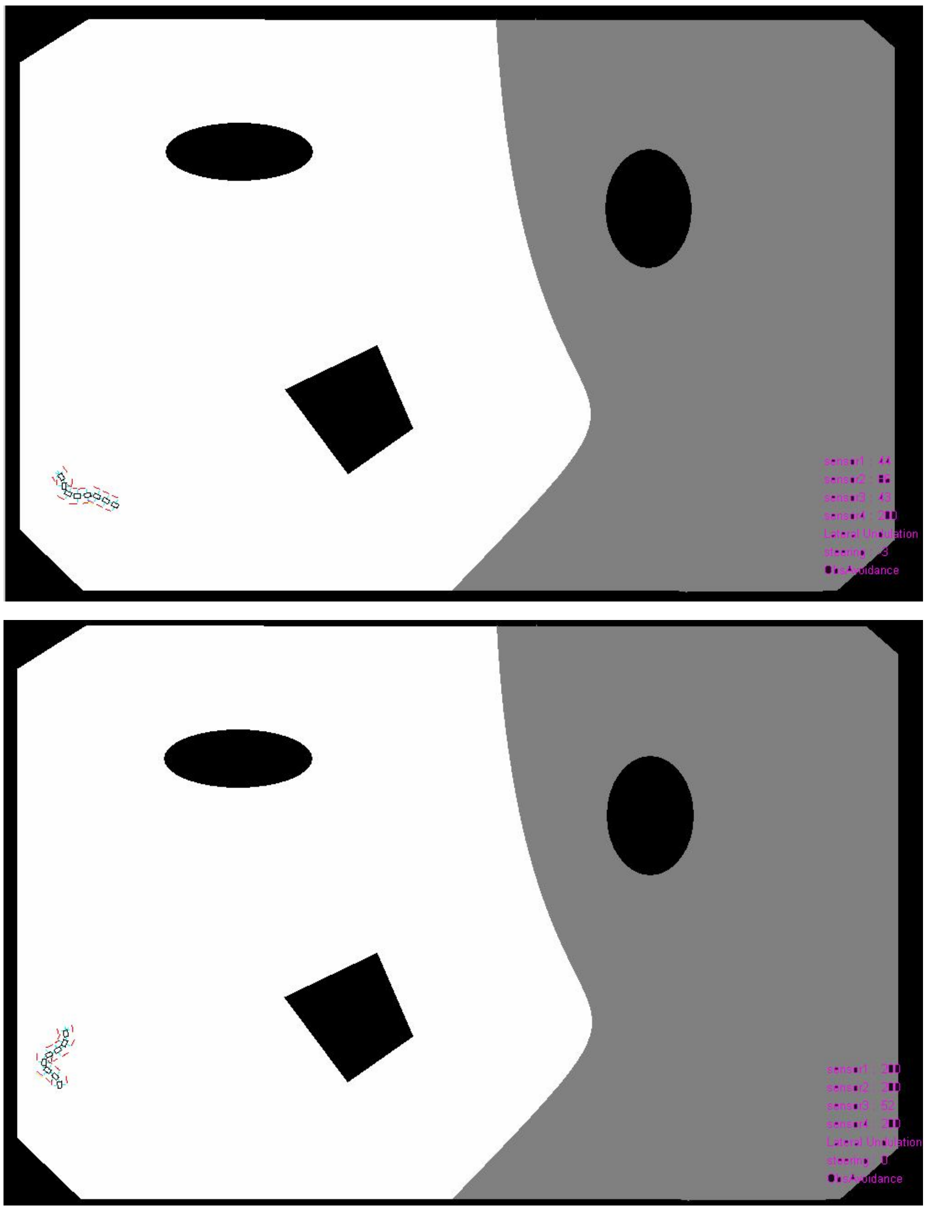
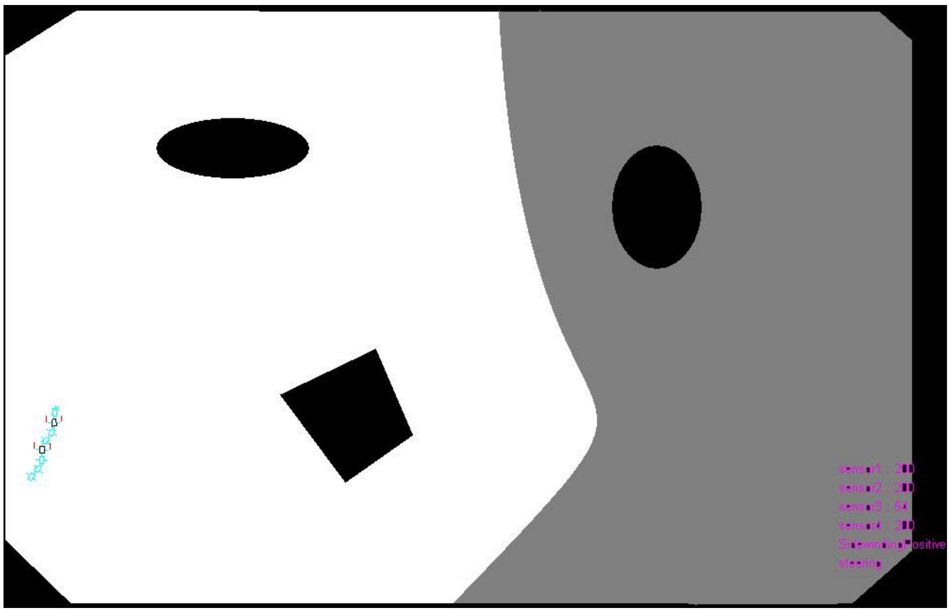


Figure 3.65. The model switches to lateral undulation to avoid the corner..

When the model senses the wall (obstacle) in front of its head, it switches to lateral undulation and makes a very sharp steering in order not to collide into the wall as seen in the figure 3.65. Performing sidewinding without fully avoiding the corner moves the model dangerously near to the corner which is a bad situation. This situation may result from insufficient training, or badly defined rewards.


```
sensor1 : 200
sensor2 : 200
sensor3 : 54
sensor4 : 200
SidewindingPositive
steering
```

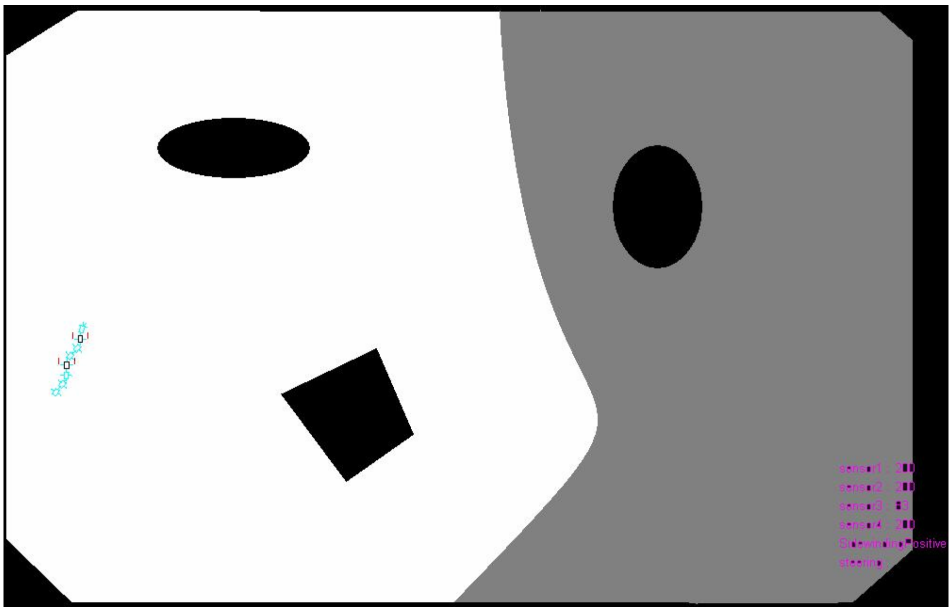
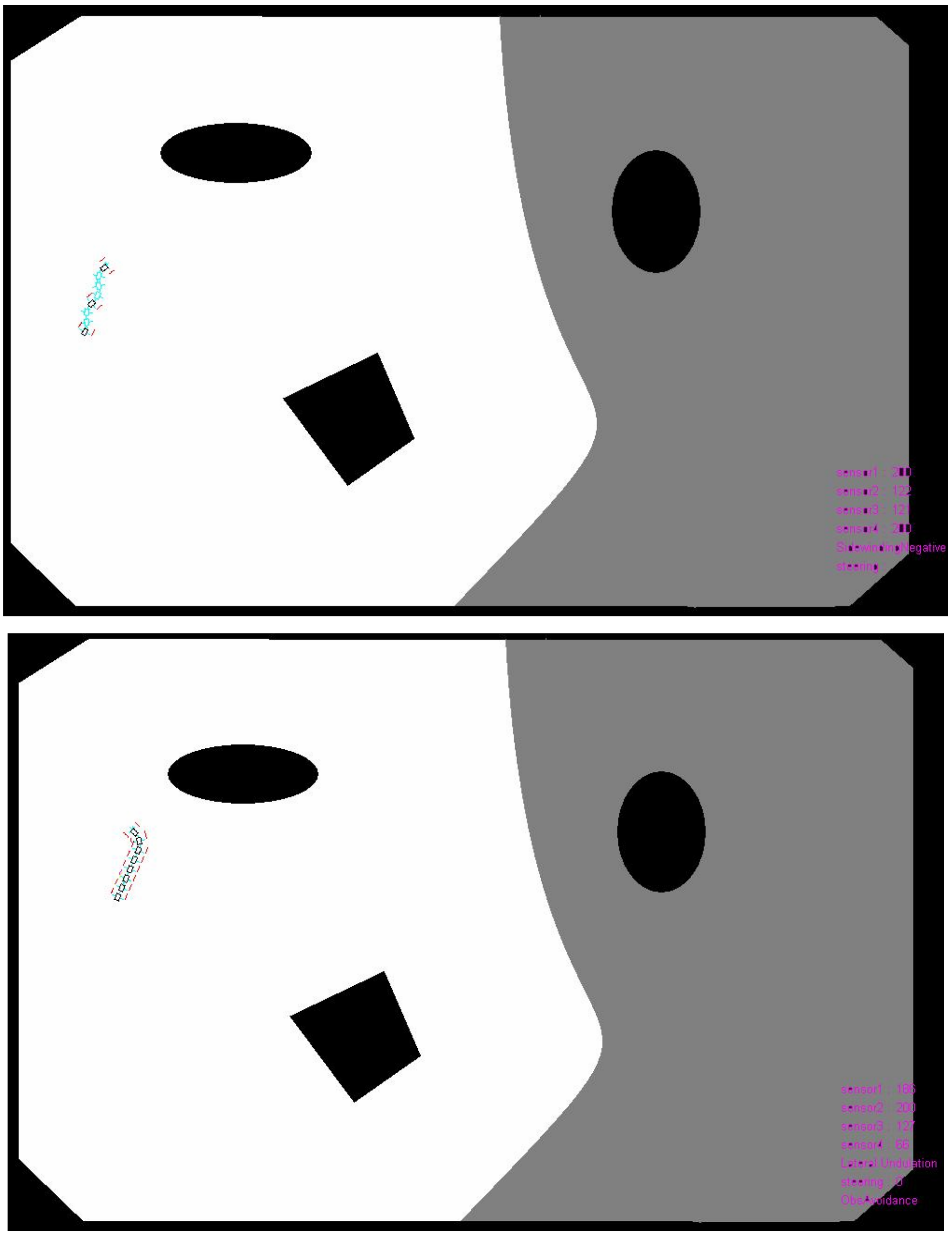

```
sensor1 : 200
sensor2 : 200
sensor3 : [illegible]3
sensor4 : 200
SidewindingPositive
steering
```

Figure 3.66. After the obstacle is avoided, the model performs sidewinding and moves away from the sidewall.

As the head of the robot is cleared from an obstacle and obstacles are near only at the side of the model, model performs sidewinding against its right side as shown in figure 3.66.



```
sensor1 : 200
sensor2 : 122
sensor3 : 121
sensor4 : 200
SidewindingNegative
steering
```

```
sensor1 : 186
sensor2 : 200
sensor3 : 127
sensor4 : 66
Lateral Undulation
steering : 0
ObsAvoidance
```

Figure 3.67. The model continues to perform sidewinding until it confronts an obstacle. The model switches to lateral undulation to avoid the encountered obstacle.

In figure 3.67, the model encounters an obstacle and switches to lateral undulation to avoid the obstacle as shown on figure 3.67.

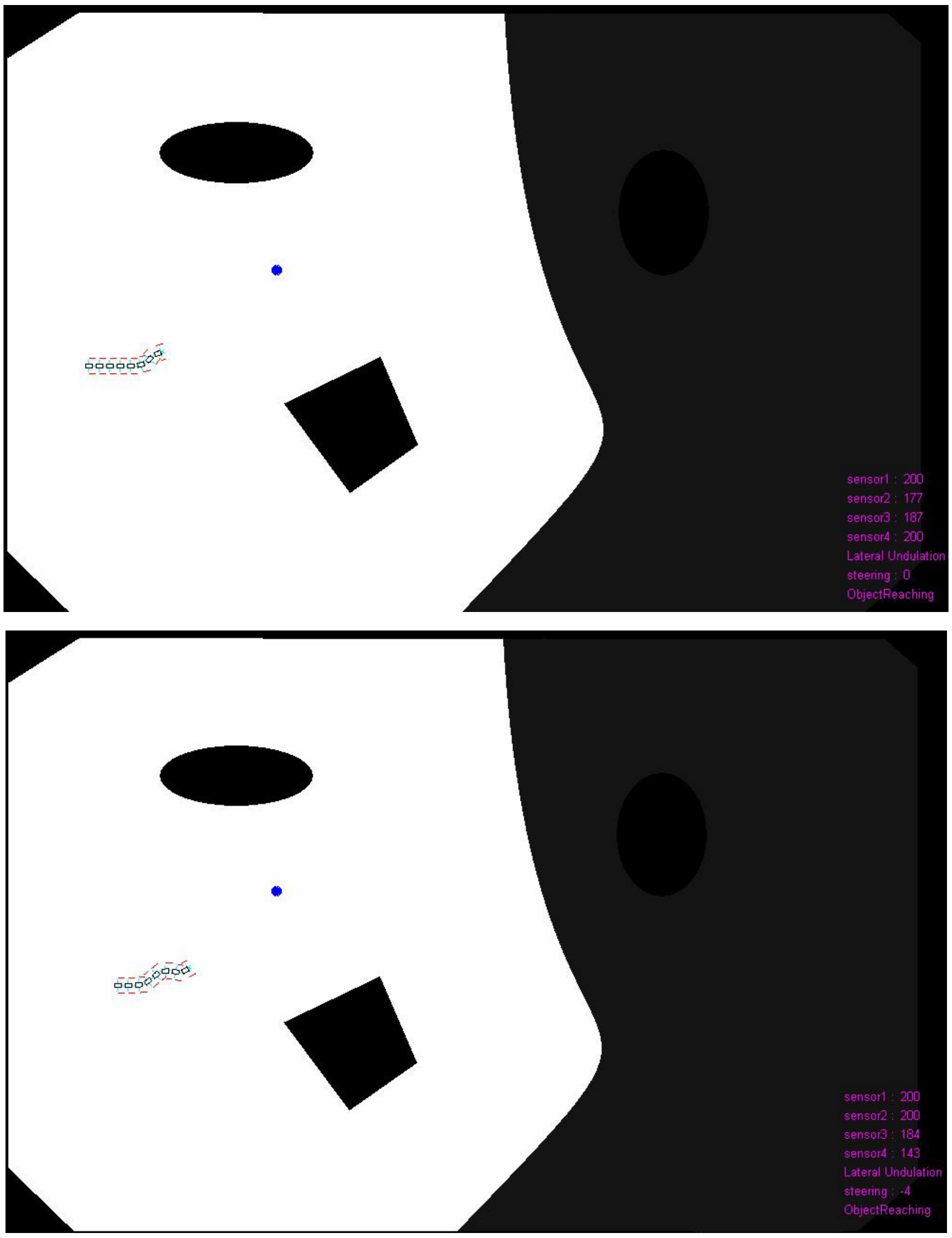
In the region with low friction, the high level controller prefers to select accordion and rectilinear, whereas in the region with high friction, it selects among sidewinding and lateral undulation. This selection is expected due to the friction properties of the gaits, introduced in section 3.4.1.

High level controller makes decisions also due to the obstacle situation of the robot. Since sidewinding gait can not be steered, high level controller selects lateral undulation instead of sidewinding when sharp steering is necessary to avoid a very near obstacle like the case in figure 3.67. High level controller also found suitable situations where sidewinding can be used to avoid obstacles like the case of figure 3.61.

In fact, rectilinear and accordion gaits can also be performed without any slippage and collision where lateral undulation and sidewinding gaits are performed. But since speed of the body is reduced to half for rectilinear and accordion gaits; the selection of sidewinding and lateral undulation gaits are reinforced more when they are applicable as stated in section 3.4.1.5. This bias to the gait selection yields to selection of lateral undulation or sidewinding gaits if the friction is high; rectilinear or accordion gaits if the friction is low.

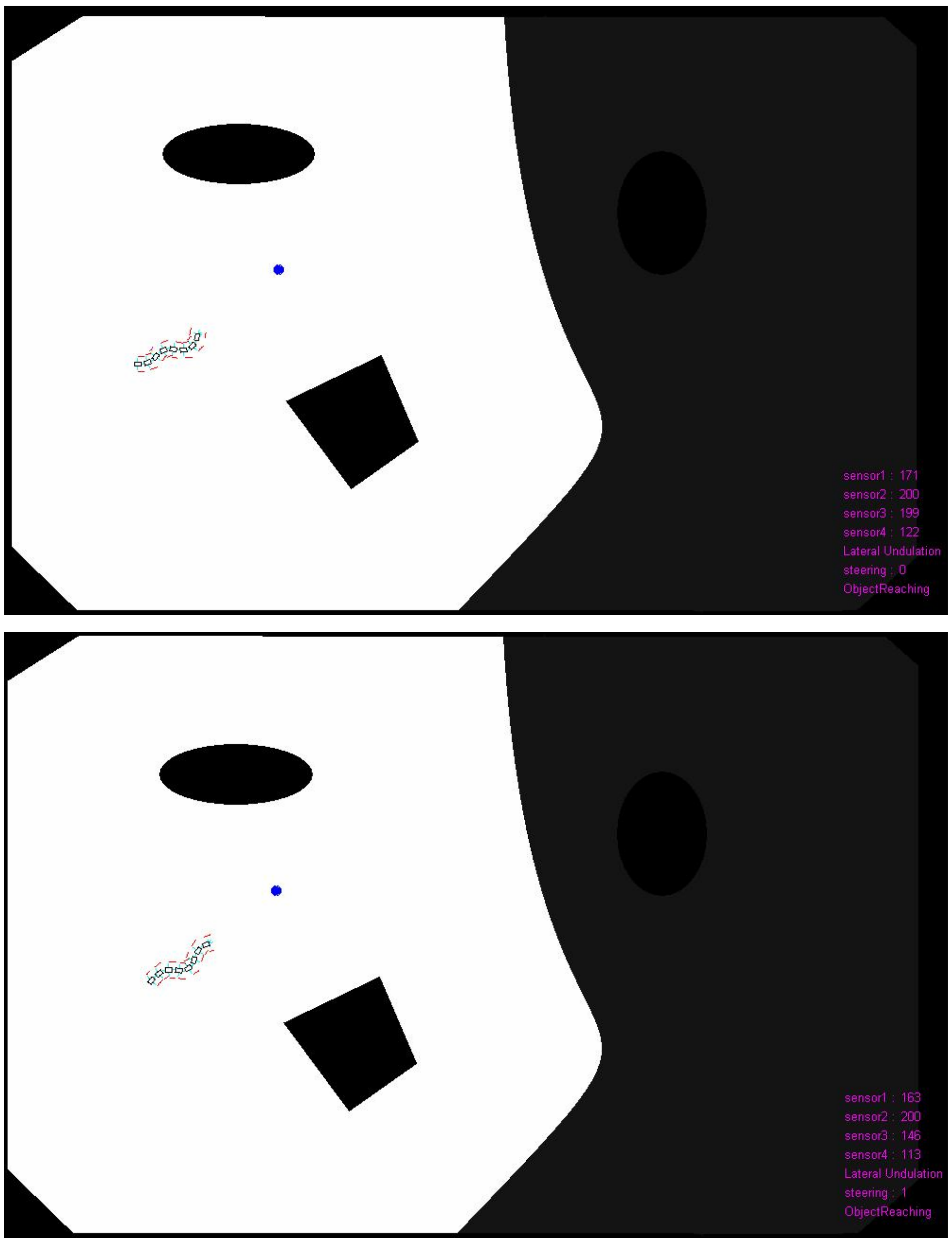
3.4.4. Simulation Results with Object

The simulation result of the model, being controlled with the complete functional controller hierarchy (introduced in figure 3.27) will be illustrated through figures .3.68 – 3.84. Interaction with the object on high friction (5 N) will be shown through figures 3.68 – 3.73, on low friction (0.4 N) will be shown through figures 3.74 – 3.84.


```
sensor1 : 200
sensor2 : 177
sensor3 : 187
sensor4 : 200
Lateral Undulation
steering : 0
ObjectReaching
```

```
sensor1 : 200
sensor2 : 200
sensor3 : 184
sensor4 : 143
Lateral Undulation
steering : -4
ObjectReaching
```

Figure 3.68. The model performs lateral undulation to reach the object. With the scheme described in section 3.3.1.3, selection of sidewinding is suppressed; instead lateral undulation with object reaching is performed.

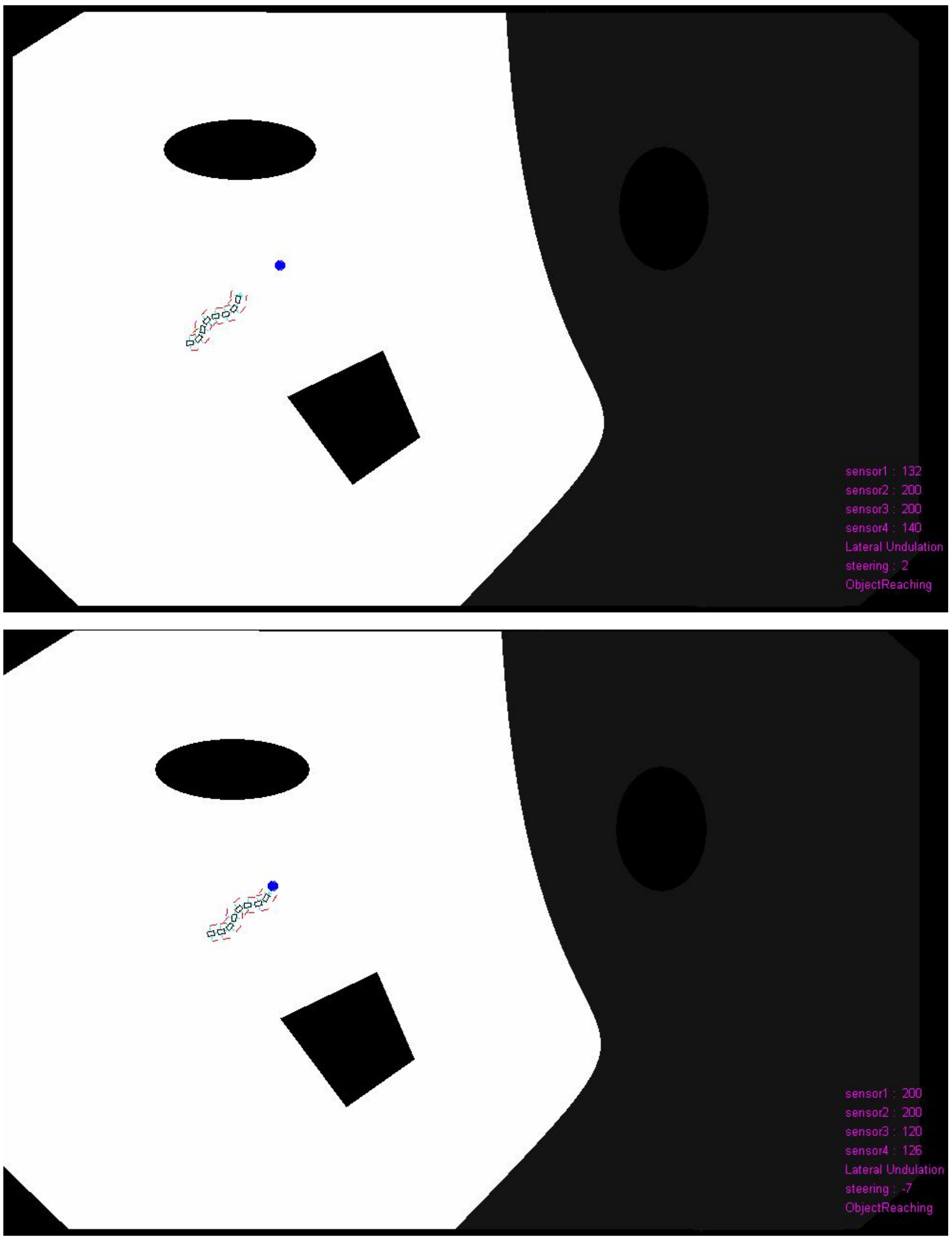


```
sensor1 : 171
sensor2 : 200
sensor3 : 199
sensor4 : 122
Lateral Undulation
steering : 0
ObjectReaching
```

```
sensor1 : 163
sensor2 : 200
sensor3 : 146
sensor4 : 113
Lateral Undulation
steering : 1
ObjectReaching
```

Figure 3.69. The model performs lateral undulation to reach the object.

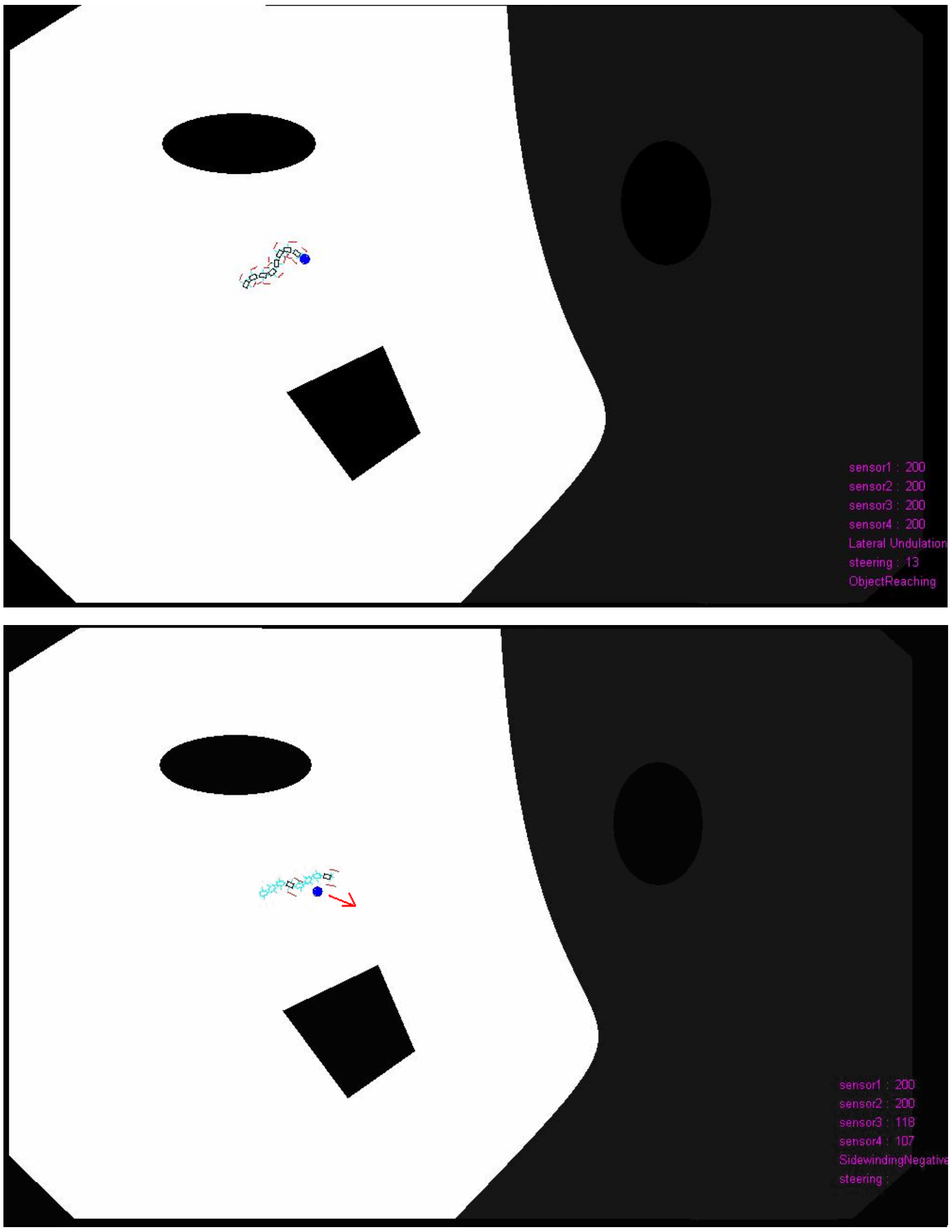
The model performs lateral undulation to reach the object as seen on the figures 3.68 and 3.69. Normally in the absence of any object, the model would prefer sidewinding; but the scheme described in 3.3.1.3 suppresses the selection of sidewinding since no steering can be applied to sidewinding to reach an object.


```
sensor1 : 132
sensor2 : 200
sensor3 : 200
sensor4 : 140
Lateral Undulation
steering : 2
ObjectReaching
```

```
sensor1 : 200
sensor2 : 200
sensor3 : 120
sensor4 : 126
Lateral Undulation
steering : -7
ObjectReaching
```

Figure 3.70. The model performs lateral undulation to reach the object.

The model reaches the object any manipulates it by pushing with its head as seen in figure 3.70. But the undulations of the gait prevent the manipulation by head to take place for a long period of time. The model looses the object from front of its head because of undulations as shown on the upper part of figure 3.71.

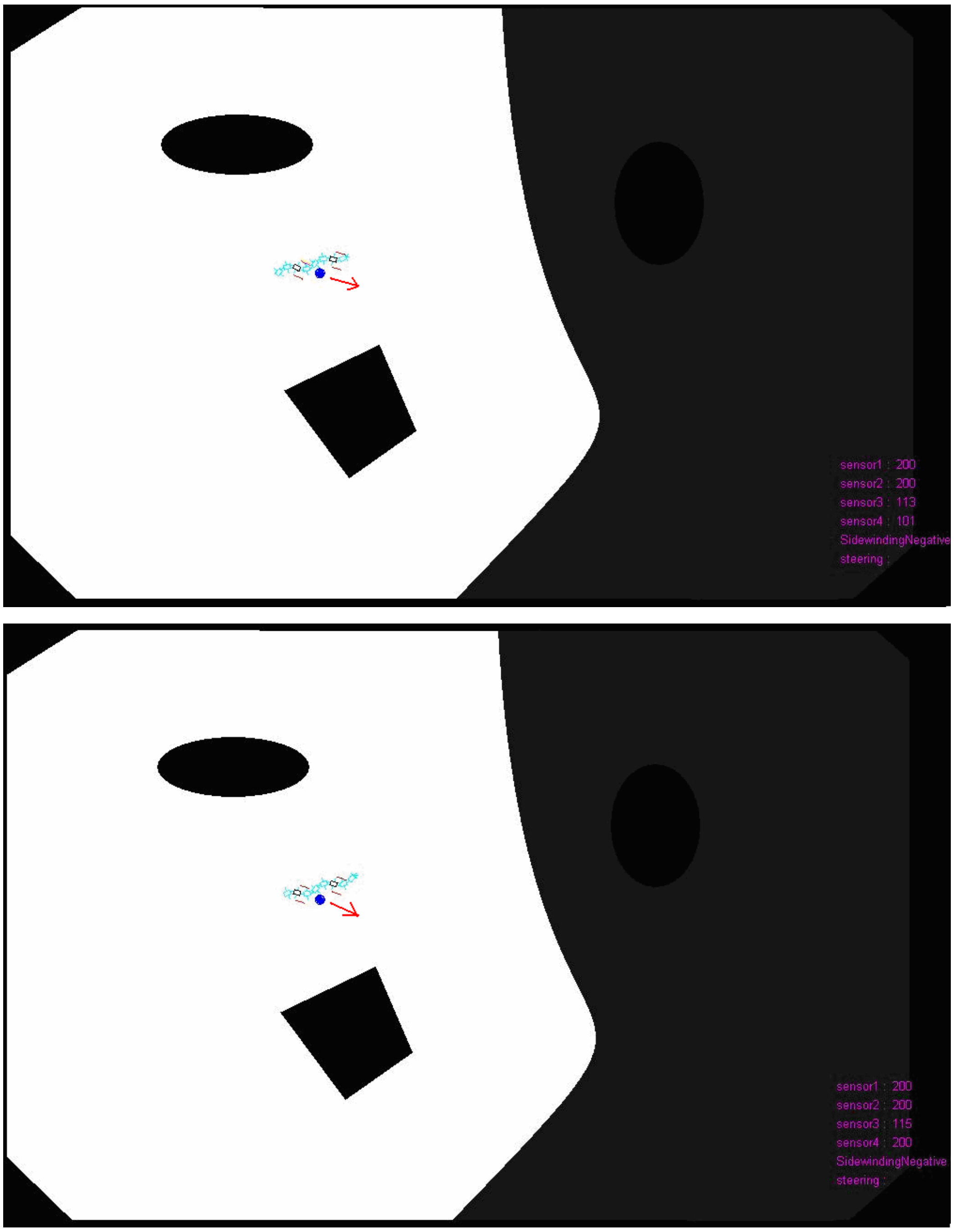


```
sensor1 : 200
sensor2 : 200
sensor3 : 200
sensor4 : 200
Lateral Undulation
steering : 13
ObjectReaching
```

```
sensor1 : 200
sensor2 : 200
sensor3 : 118
sensor4 : 107
SidewindingNegative
steering :
```

Figure 3.71. After the object is reached; the model will manipulate the object by pushing it with its head. But the object is too near and the undulation of the gait will make the head miss the object. After the head misses the object, the criteria induced in 3.3.3 are met, and the model executes sidewinding to manipulate the object with its body along its motion direction.

After the model looses the object from the front of its head, the criterion at section 3.3.3 is met triggering the manipulation by the body scheme. Dragging by the body is illustrated in figures 371, 372 and 3.73.



```
sensor1 : 200
sensor2 : 200
sensor3 : 113
sensor4 : 101
SidewindingNegative
steering :
```

```
sensor1 : 200
sensor2 : 200
sensor3 : 115
sensor4 : 200
SidewindingNegative
steering :
```

Figure 3.72. The object is pushed along the body.

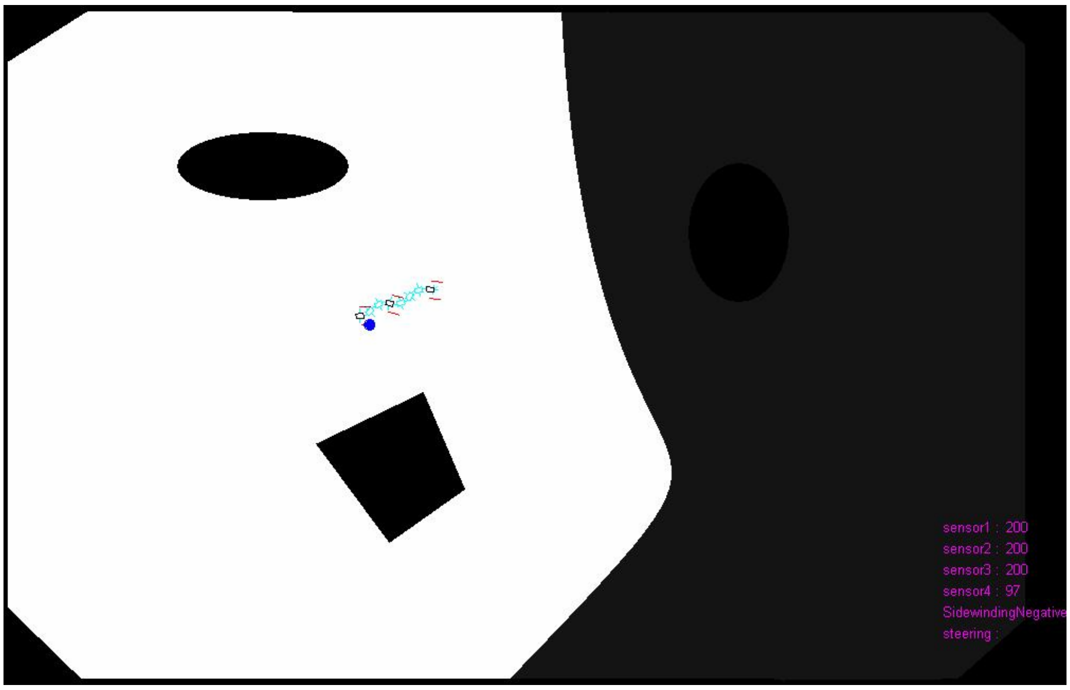


Figure 3.73. Since sidewinding gait does not grasp, the object will be left behind as it reaches the tail of the model. After the object is left, the model continues its ordinary navigation.

The object manipulated by the body can not be manipulated continuously, has to be leaved behind as shown on figure 3.73. But this scheme is useful because at the passes of the model from the object site will bring the object near to a stationary obstacle cleaning the path. Also by using this scheme, the model manipulates the objects in the direction of its body which can take the object away opening passages.

The environment seen in figure 3.74 is used for object interaction in low friction (0.4 N). The model begins at a region with high friction.

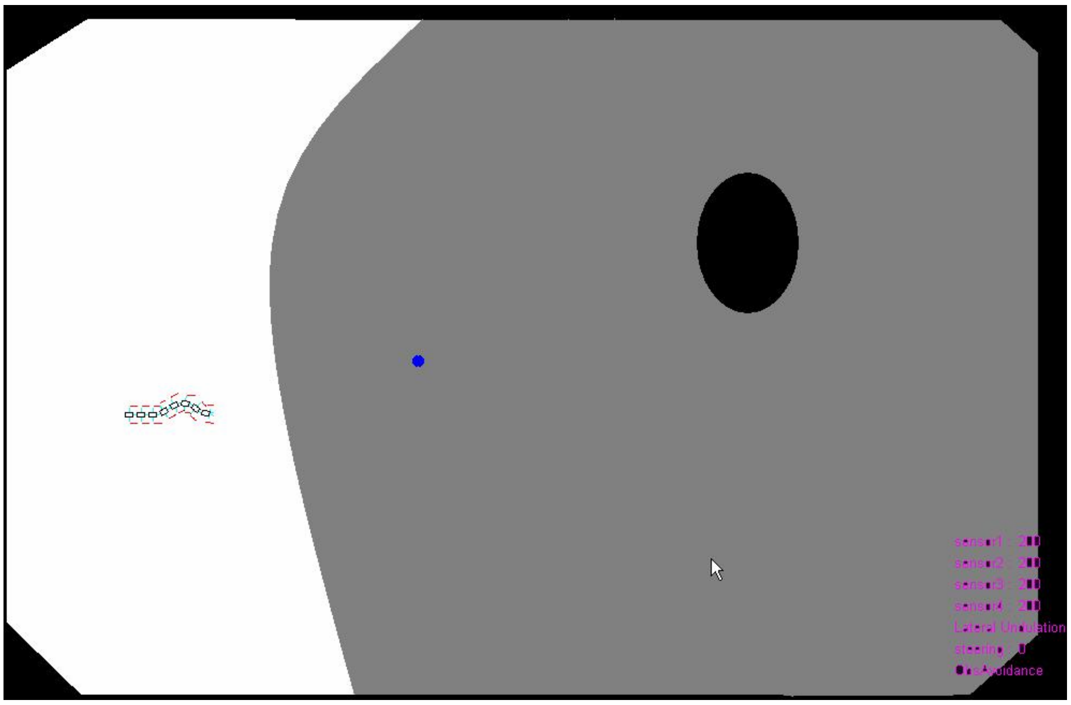


Figure 3.74. The object is in the slippery portion of the environment. The gray part has 0.4N friction while the white part has 5 N friction. The model begins with lateral undulation for object reaching.

The model performs lateral undulation gait with object reaching behavior as shown in figure 3.74, which is expected since lateral undulation gait is preferred in the high friction regions.

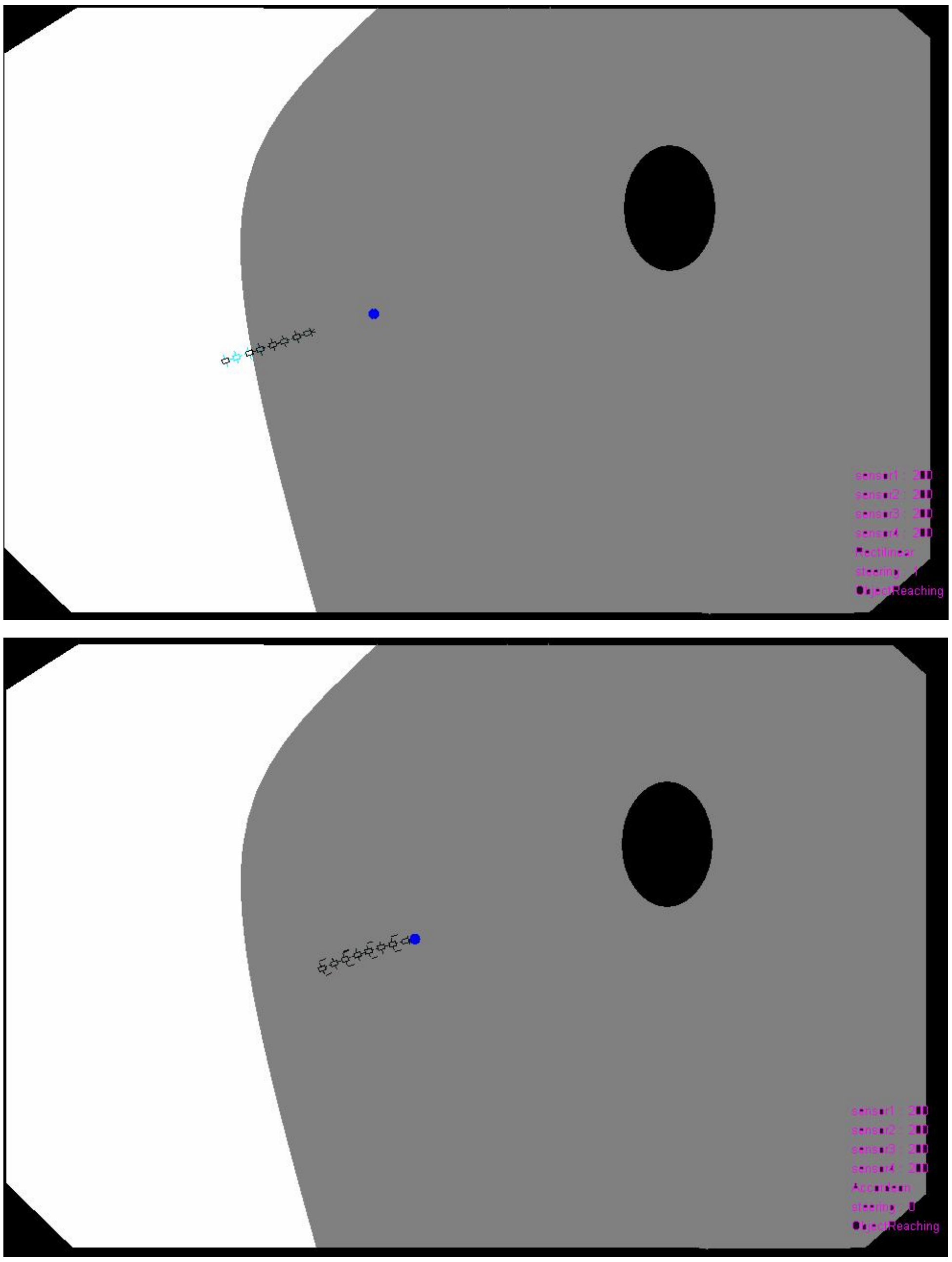


Figure 3.75. In the slippery portion of the environment the model switches to rectilinear, and approaches the object.

The model switches to rectilinear gait as it passes to low friction region of the environment as seen in figure 3.75. Despite of the gait change, object reaching behavior is maintained. The model begins to manipulate the object by pushing with its head as seen in the lower part of the figure 3.75.

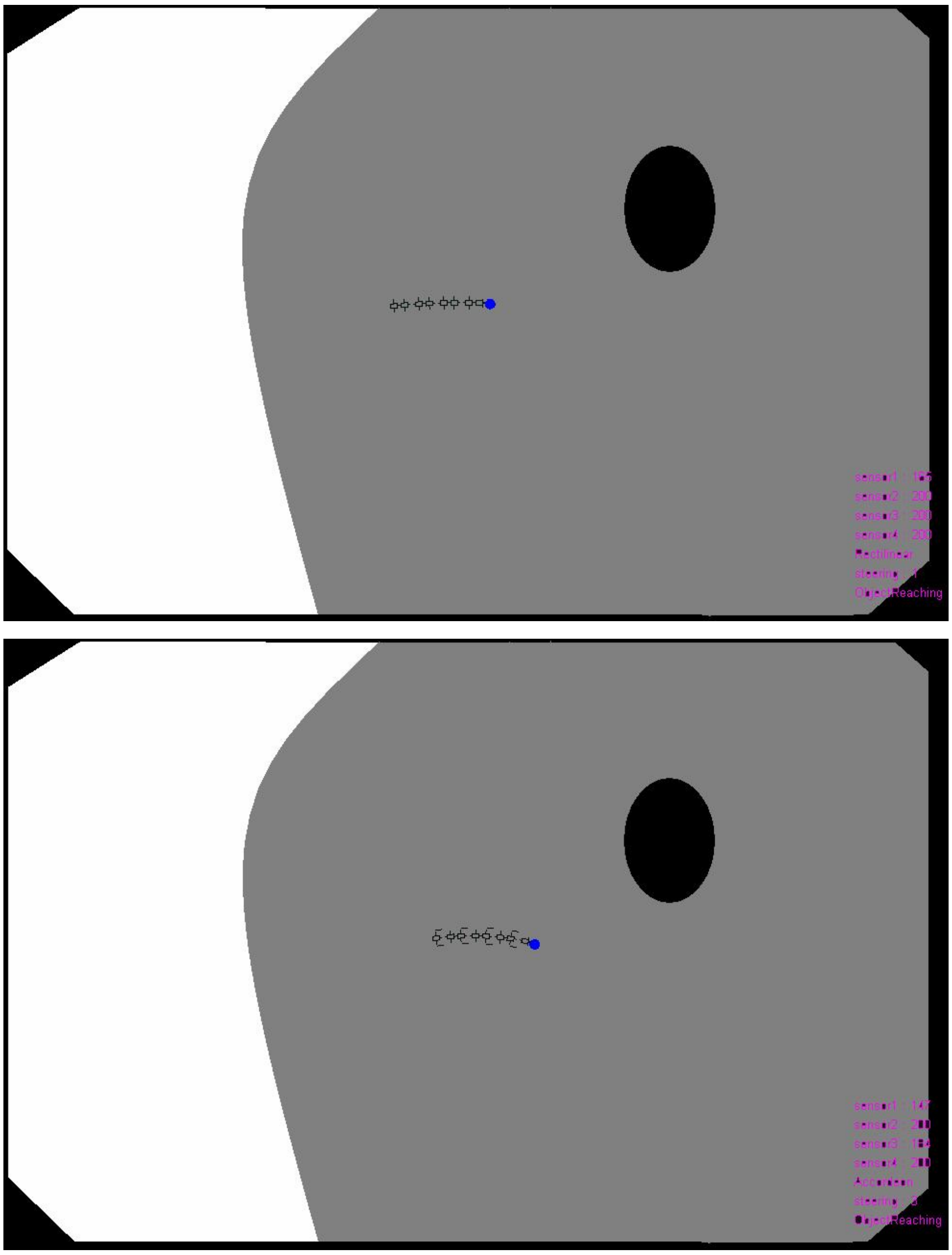


Figure 3.76. The model continues to push the object with its head whiling switching between accordion and rectilinear gaits.

In the low friction area of the environment, the model switches back and forth on rectilinear and accordion. This issue was discussed in section 3.4.3. But the model continues to push the object without loosing it from the front of its head as seen in figure 3.76.

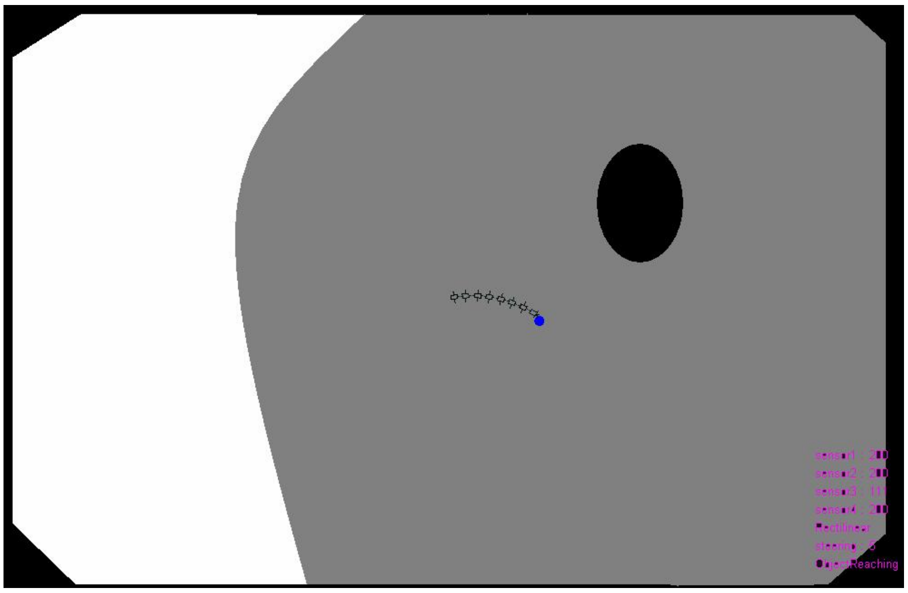


Figure 3.77. Since no undulation is present the model will keep pushing the obstacle with its head until the criteria in the 3.3.3 becomes invalid by a nearby obstacle or by loss of the object from the front side. In this next scenario this situation will be simulated.

Since accordion and rectilinear gait have no undulations, the model continues to push the object as seen in figure 3.77. The criterion in section 3.3.3 will be never met without encountering any obstacle. To illustrate the activation the criterion in section 3.3.3, the environment in figure 3.78 is used.

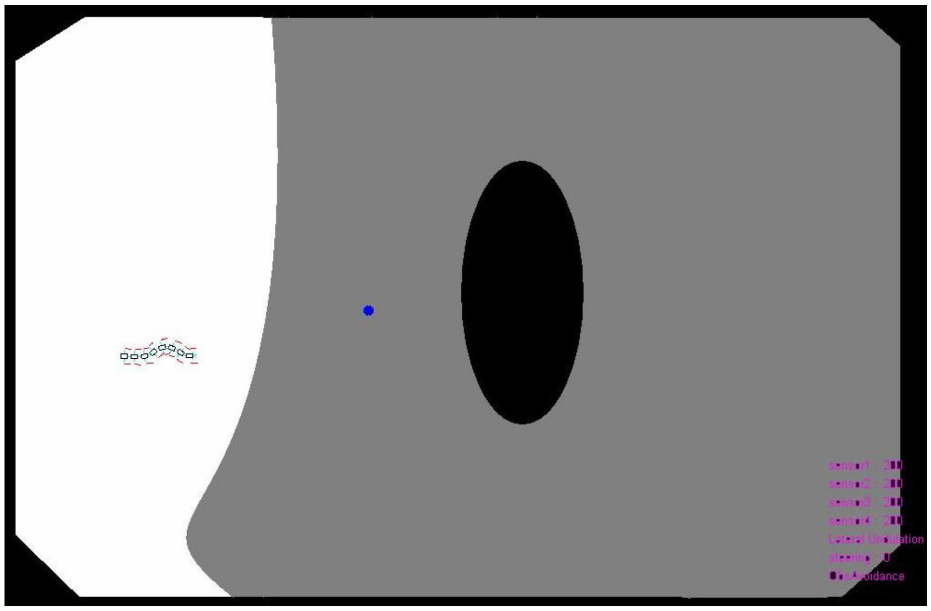
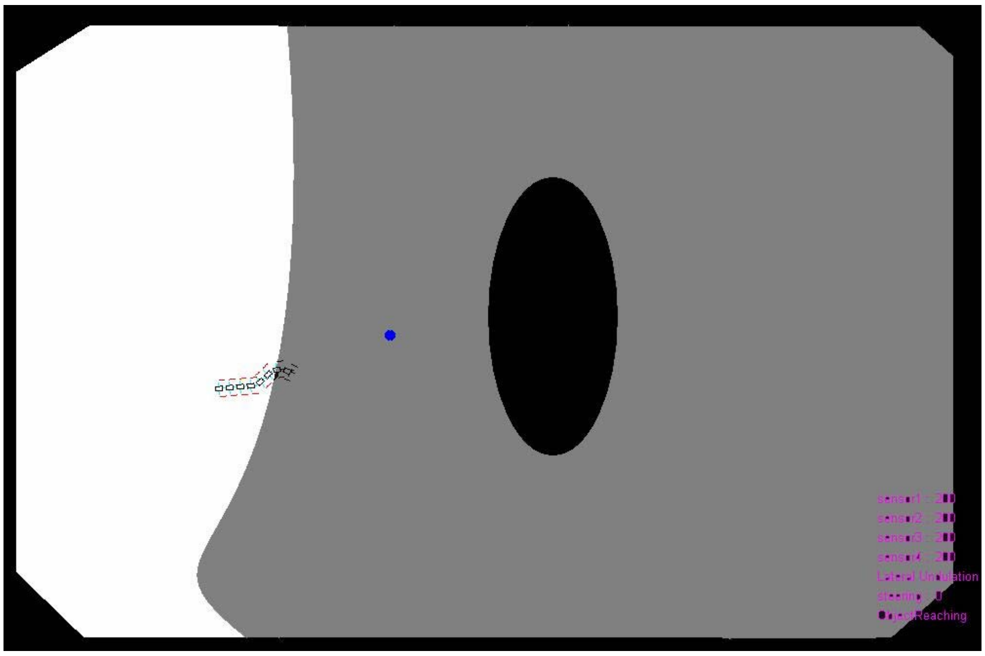
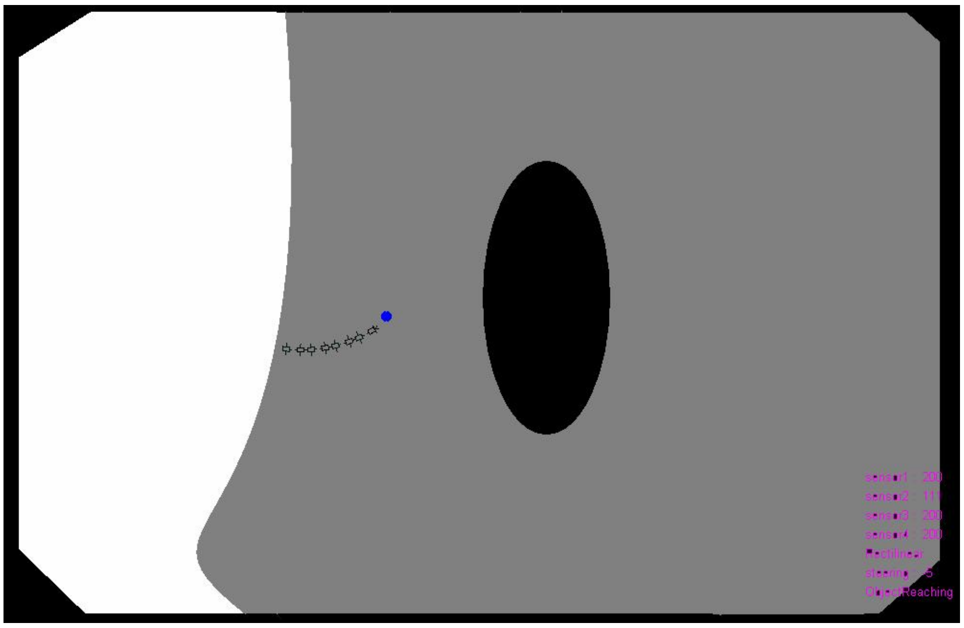


Figure 3.78. The model begins with lateral undulation.



```
sensor1 : 200
sensor2 : 200
sensor3 : 200
sensor4 : 200
Lateral Undulation
steering : 0
ObjectReaching
```

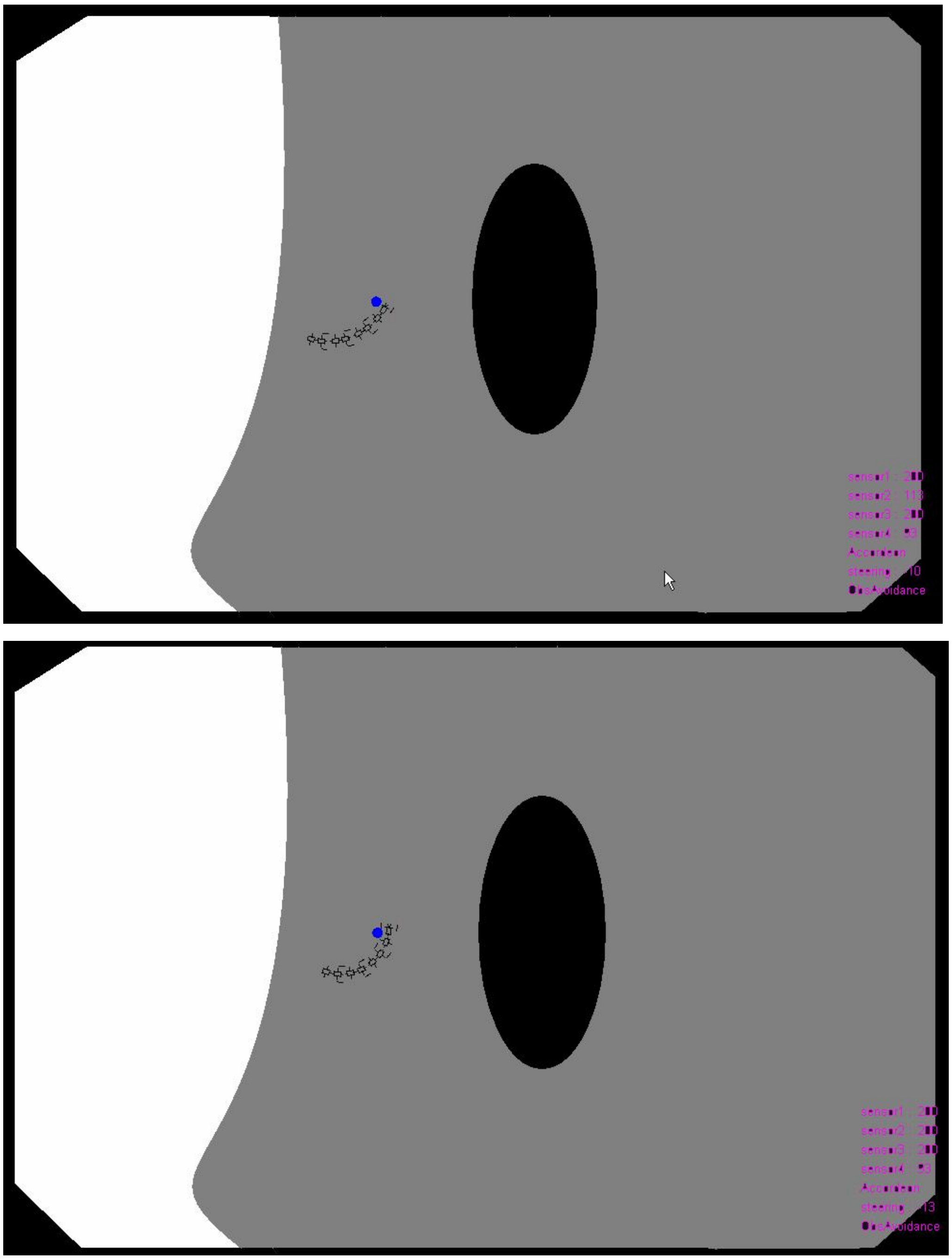
Figure 3.79. The model enters low friction area.



```
sensor1 : 200
sensor2 : 111
sensor3 : 200
sensor4 : 200
Rectilinear
steering : -5
ObjectReaching
```

Figure 3.80. In the low friction portion of the environment, the model reaches the object by rectilinear gait.

The model approaches the object as usual through figure 3.78 – 3.80.



```
sensor1 : 200
sensor2 : 113
sensor3 : 200
sensor4 : 58
Accordeon
steering : 10
ObsAvoidance
```

```
sensor1 : 200
sensor2 : 200
sensor3 : 200
sensor4 : 58
Accordeon
steering : 13
ObsAvoidance
```

Figure 3. 81. The model senses the nearby obstacle, and ceases object reaching behavior. The model tries to avoid the obstacle with accordion gait.

The model senses the obstacle in figure 3.81 and switches to obstacle avoidance behavior. The model ceases pushing the object and makes a left steering to avoid the obstacle as shown in figure 3.81.

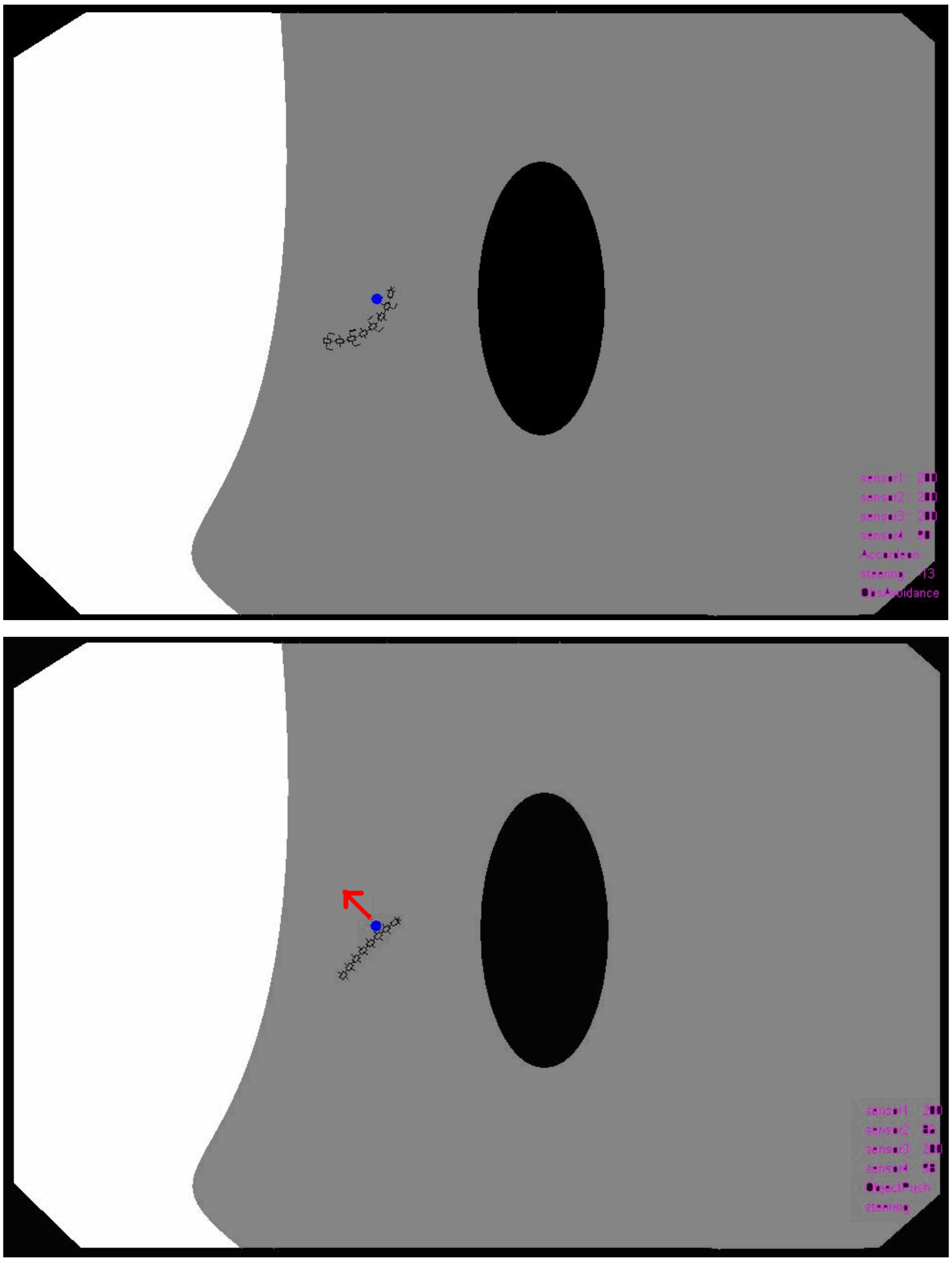


Figure 3.82. During the avoidance of the obstacle, the object get too near to the side of the model body triggering the criteria induced in 4.3.3. The model suspends obstacle avoidance, and pushes the object by curving its body.

During obstacle avoidance, the object gets too near to the body as seen in the figure 3.82 and the criterion in 3.3.3 is satisfied.

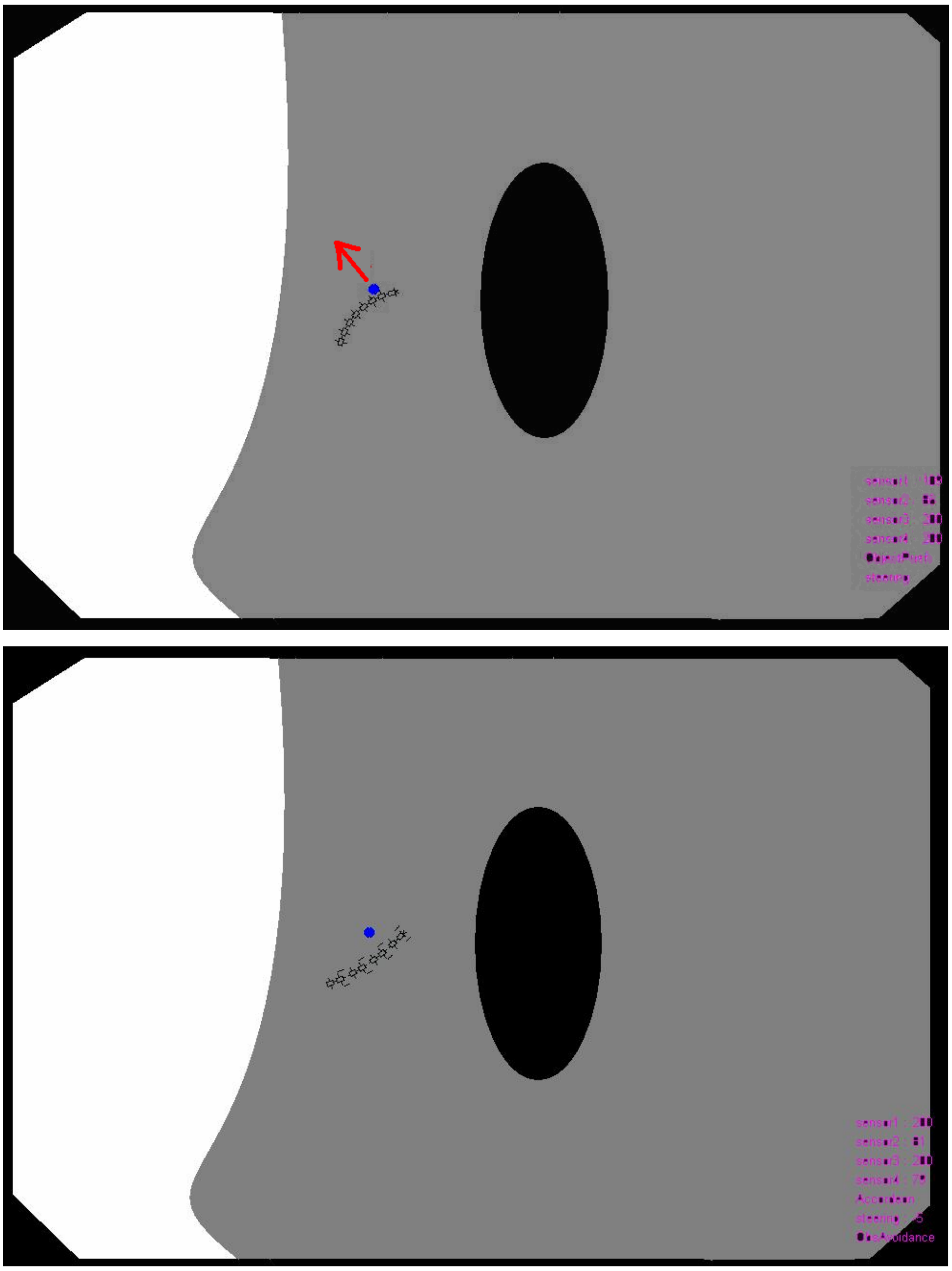


Figure 3.83. After the object is pushed to a safe distance, the model resumes accordion gait with obstacle avoidance.

The model pushes the object away by curving the body as seen in lower part of figure 3.82 and upper part of figure 3.83. After the object is pushed away, model continues its obstacle avoidance process.

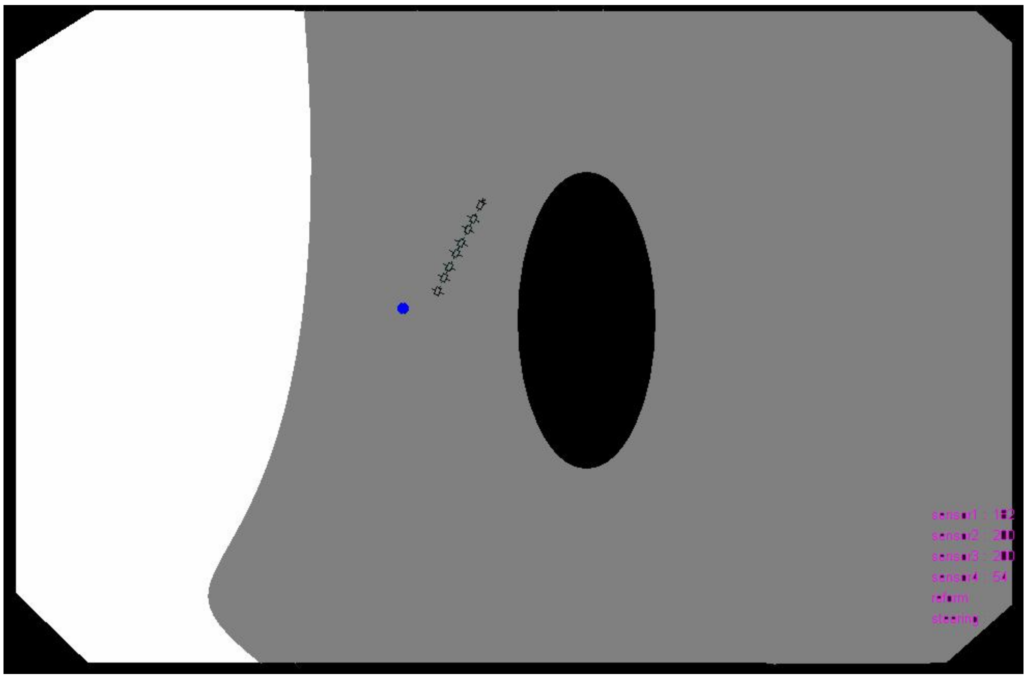


Figure 3.84. Without the object, ordinary navigation due to the environment takes place.

Since the body shape of the rectilenar and accordion gaits are not suitable for taking the object along the body, pushing it away is the only option which may help to minimize the interferene of the body with the object. Also this scheme may help to push the object nearer to the stationary obstacles, opening up more free space in narrow passages on low friction areas.

CHAPTER 4

MECHANICAL IMPLEMENTATION

4.1. Robot Structure

Mechanical design of the proposed snake robot consists of four tank chassis interconnected to each other by a three degree of freedom joints (Roll, pitch, and yaw) (figure 4.1). Each of the tank palettes are actuated by separate dc motors. The joints are passive except for yaw axis. Yaw axis of the joints is actuated by separate servos. This design, which is similar to the “Kohga”, purposefully selected for the ease of implementation and mechanically effective propulsion.

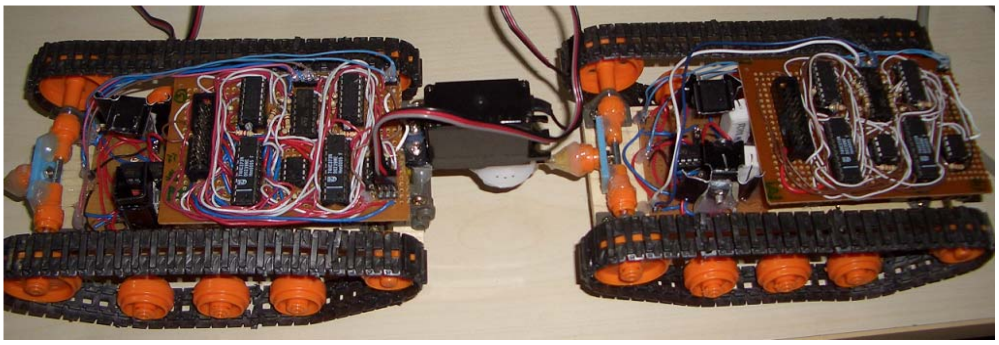


Figure 4.1. Proposed Snake-Like Robot (Only two sections visible).

4.1.1. The Body

The tank chassis are the main body of the proposed snake robot (figure 4.2). Each tank chassis is identical for each robot element consisting two dc motors with separate gearboxes driving 2 tank palettes.

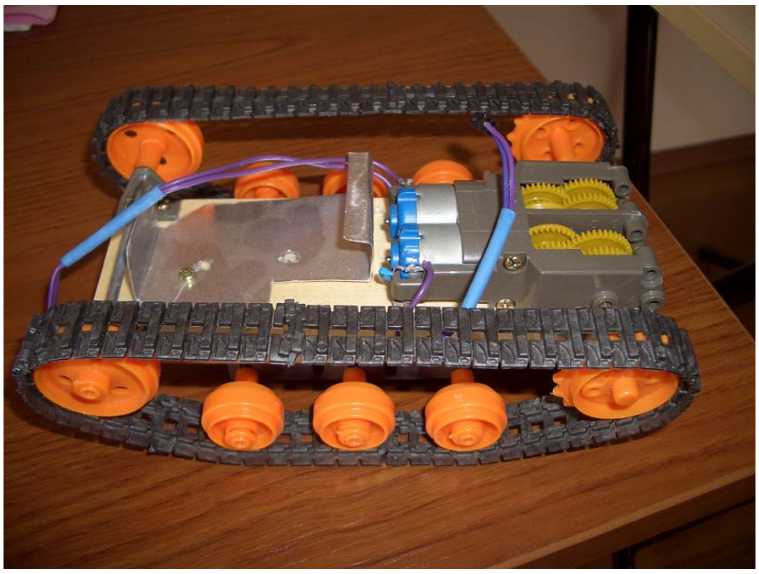


Figure 4.2. Tank Chassis with dc motors and gearboxes.

Gearboxes have 203/1 gear ratio. The dc motors are standard, operating under 3 volts; but unfortunately the precise model of the dc motor is not known.

The palettes of the tank have good grip through a large frictional surface providing good traction necessary for navigation on surfaces that may as well be slippery.

4.1.2. Joints

Robot elements (wagons) are connected by rotational joints having three degree of freedom each in the direction of roll, pitch and yaw. The roll and pitch axes are passive and free which permits the robot element to adapt itself to 3D terrains. The yaw axis is driven by a servo allowing its angular motion control (figure 4.3).

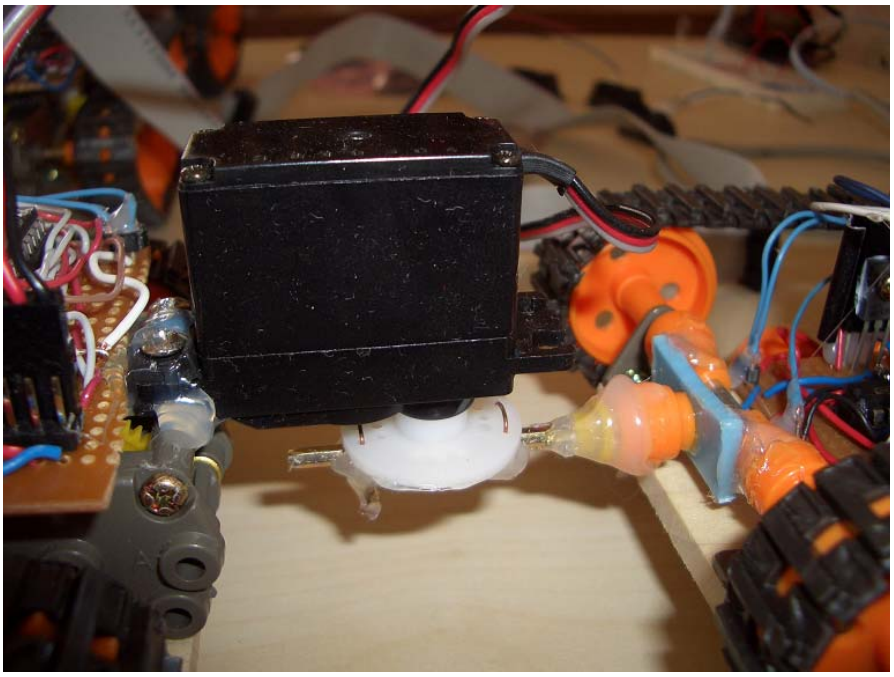


Figure 4.3. A joint between the tank chassis. Roll & Pitch axes are free; servo drives the yaw axis.

4.2. Microcontroller Architecture

The proposed snake-like robot consists of eight low level dc motor controllers, three servos and ten IR sensors which should be handled electronically to control the whole robot. This handling task should be conducted by a microcontroller. In the proposed implementation; this task is accomplished by an ATMEL 89C52 (figure 4.4) microcontroller. ATMEL 89C52 has features as two timers, 40 I/O ports, serial port, that make it very applicable to the task. For 89C52 an 8052 assembler code up to 8k can be embedded.

Although 89C52 is very flexible and has a moderate processing power, still its processing power is not sufficient to conduct the high level control specified in

2.3.9 FACL Controller section so the high level control is conducted in a PC instead of 89C52.

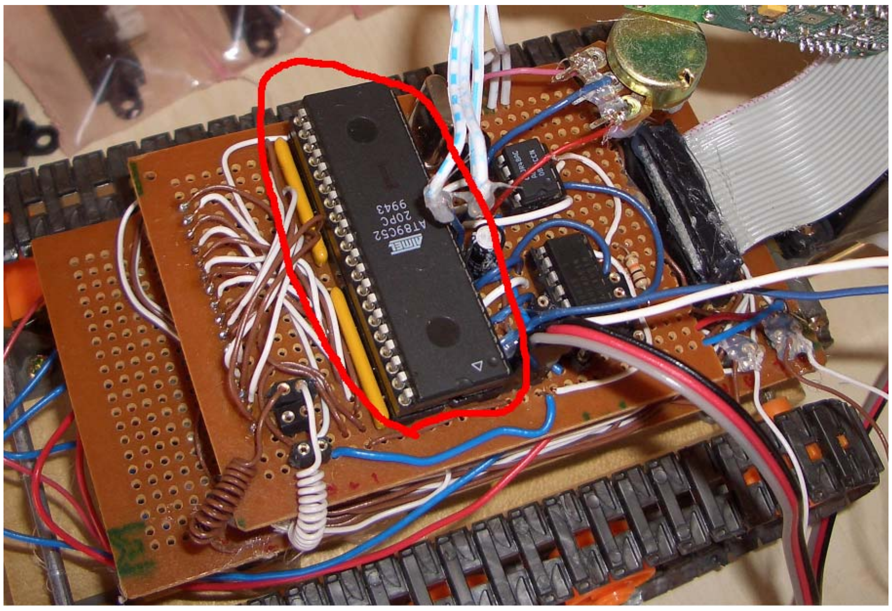


Figure 4.4. The 89C52 Microcontroller. The 89C51 marked with red in the figure is located at the head of the robot; and it drives all servos, all low level dc motor controllers

4.1.1. Sensors for the Controller

The proposed snake-like robot is equipped with ten SHARP infra-red detectors (figure 4.5). SHARP GP2D120 can sense distances between 80 and 10 cm; and produces a non-linear analog output (figure 4.6).

Figure 4.5. SHARP GP2D120 IR range sensor.

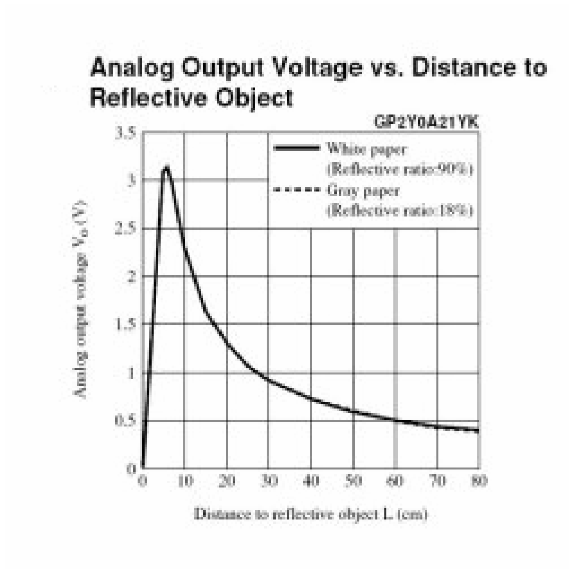


Figure 4.6. Distance, IR Sensor Output Voltage Relationship. [From datasheet of GP2D120.]

The orientations of the sensors are shown on figure 4.7.

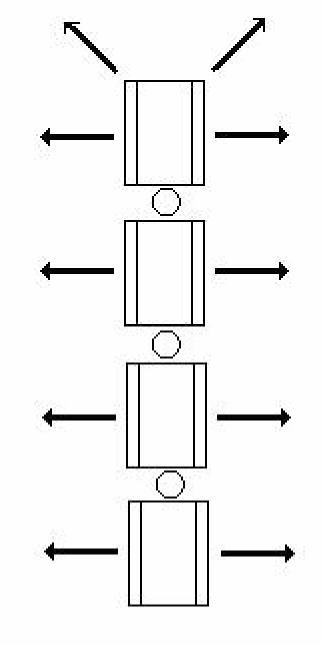


Figure 4.7. Orientation of sensors. Upper segment with four sensors is the head.

Each four segments of the robot is also equipped with identical ADXL320 accelerometers which measures accelerations in two dimensions (planar) up to +/-2 g. (figure 4.8).

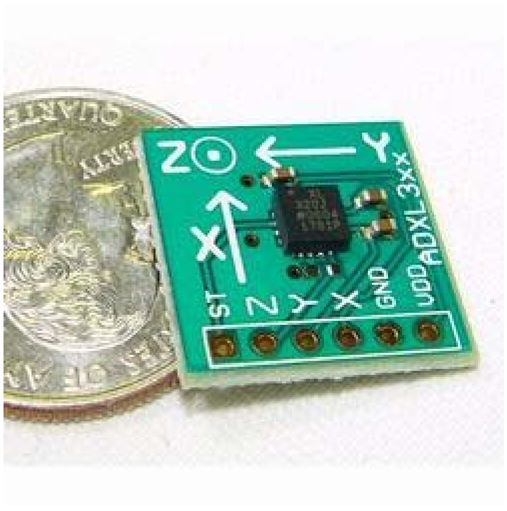


Figure 4.8. Accelerometers with two axes. Accelerometers are positioned to set the +x direction parallel with the corresponding robot segment forward direction. [Picture taken from www.sparkfun.com]

The output characteristic of the accelerometers depends on the user configuration. A low pass filter can be coupled to the output of the accelerometers to limit the bandwidth of the accelerometers rejecting most of the noise caused by the vibrations of the tank segments. The outputs of the used accelerometers are limited with 50 Hz by externally attached low pass filters.

4.2.2. Microcontroller to IR Sensor Interface

The microcontroller samples and quantizes the analog outputs of IR sensors and accelerometers through an analog to digital converter (ADC) and an analog switch which are shown on figure 4.9. For ADC an ADC0831 and for analog switch a 74HC4051 integrated circuits are used.

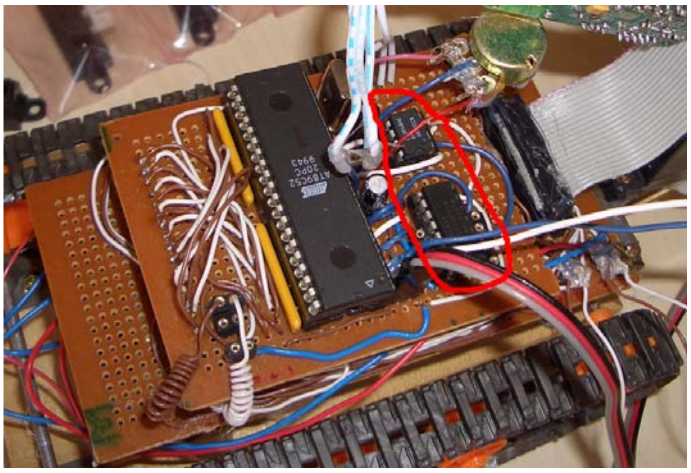


Figure 4.9. The ADC and the analog switch marked with red.

The interface of ADC, analog switch, and the microcontroller is as follows:

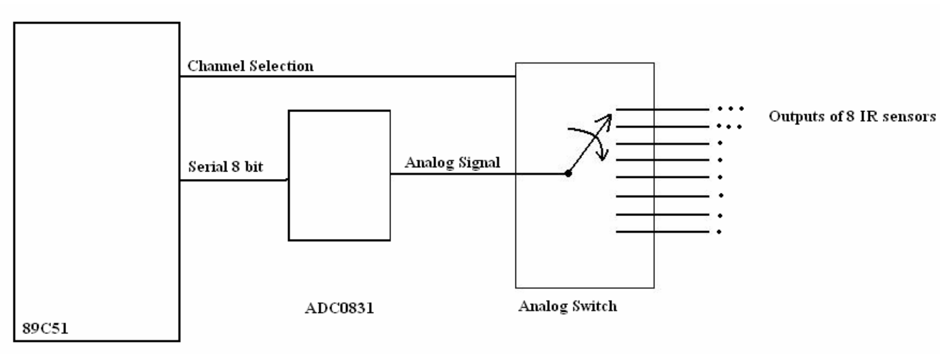


Figure 4.10. Interface of 89C51 and IR sensors and Accelerometers.

4.3.3. Joint Servos

The servos (figure 4.11) used to actuate the yaw axis of the joints produce 45 kg per meters torque and 240 degrees per second speed.

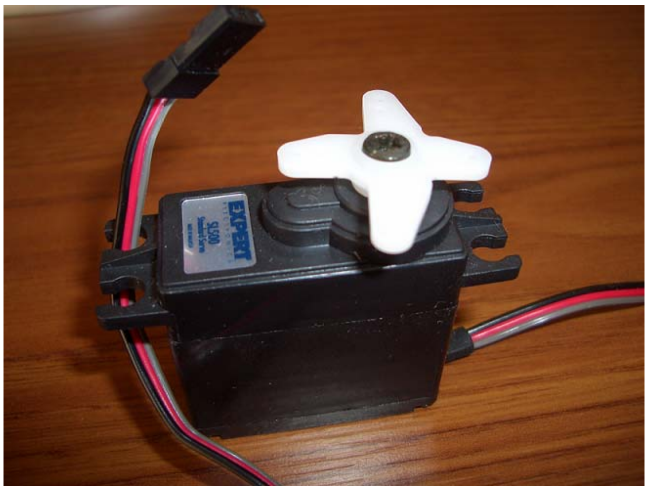


Figure 4.11. Servo used in joints.

The position of a servo can be adjusted by a pulse width modulated (PWM) signal as shown on figure 4.12.

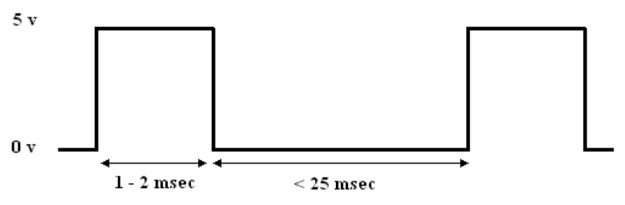


Figure 4.12. Servo input signal.

The peak of the signal which is at 5 volts determines the position of a servo. A 5 volt pulse with 1 milliseconds width takes servo to zero degrees; a pulse width of 2 milliseconds takes the servo to the maximum degree which is 180 degrees for a standard servo. A pulse width between 1 and 2 milliseconds can adjust the position of the servo to any degree within in its operation angles. The logic low part of the signal which is at 0 volts must not be greater than 25 milliseconds; because after 25 milliseconds the servo enters sleep mode. If a servo is fed with a valid control signal which has off part smaller than 25 milliseconds; the servo changes its position to the desired angle and holds its desired position. If servo enters sleep mode; it releases its position and becomes free.

4.2.4. Microcontroller to Servo Interface

The interface between the 89C52 and the servos is quite straightforward; since servos have control inputs which are pulse modulated signals which are digital. Servos can directly be connected to the microcontroller (figure 4.13).

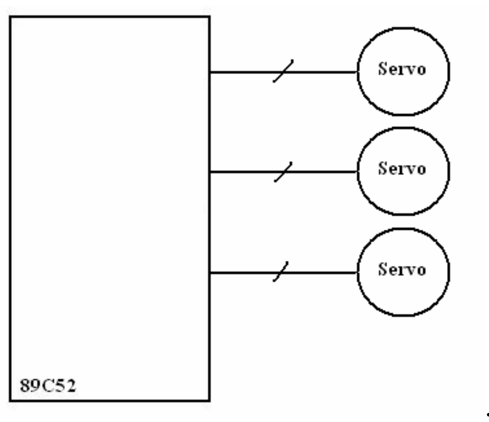


Figure 4.13. 89C52 Servo interface. (Servo power inputs are not shown.)

The control signal is produced as specified in 4.3.2 Joint Servos section.

4.2.5. Microcontroller to PC Interface

The high level control is conducted in a PC, the output of the high level controller should be sent to the robot; and the sensor outputs in the robot should be sent to the PC for input to the high level control (figure 4.15). The best interface is to use a RF (radio frequency) modem between the robot and the computer.

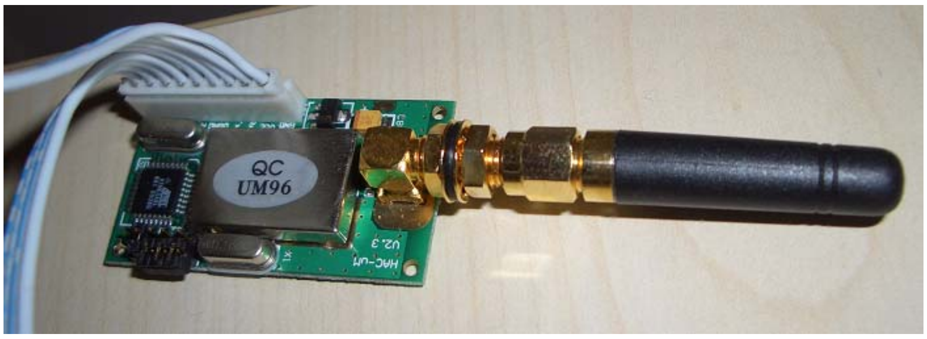


Figure 4.14. Half duplex RF modem.

Identical RF modems shown on figure 4.14 is used on both robot and the computer. The modem supports serial rs232 communication with 9600 baud rate.

The RF modem is half duplex which means both modems can not transmit and receive simultaneously. Only one of the PC or robot can speak at a time so a very simple protocol is used in the interface. The protocol is as follows:

1. PC sends eight palettes speeds in order,
2. PC send three servo angles in order,
3. After receiving 1 or 2 the robot samples all fourteen of the sensors and sends all sensor output to the PC in order.

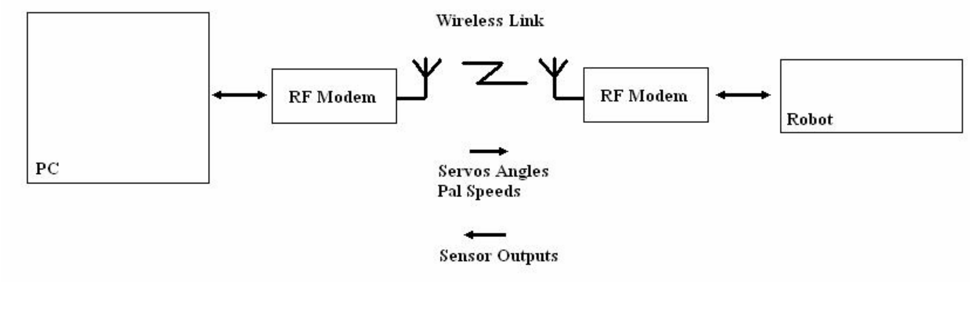


Figure 4.15. PC & robot interface.

4.2.6. Microcontroller to DC Motor Controller Interface

All of the eight dc motor (low level) controllers require an analog input so a DAC (digital to analog) converter is necessary to interface the 89C52 and the low level controllers. DAC0808 is used for digital to analog conversion which has 8 bit parallel input and corresponding 256 level resolution. For proper low level control, the analog input signal must be continuously fed to the low level controllers so all low level controllers have separate DACs. All eight of the DACs are interfaced to the 89C52 via a data bus (figure 4.16; 4.17). On the databus the 89C52 can address any of the eight DACs and set its output voltage level.

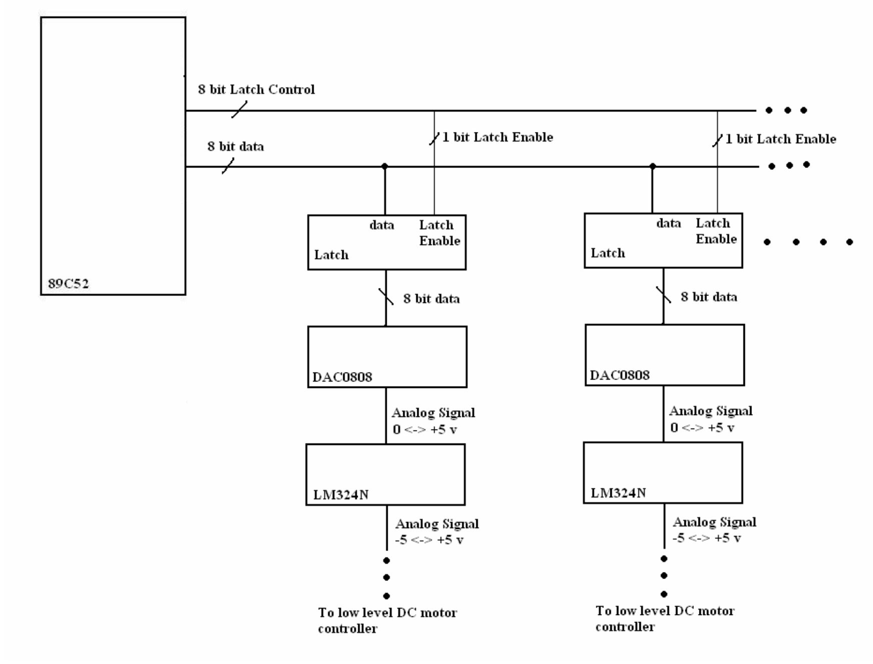


Figure 4.16. Databus and low level control interface topology. The 89C52 can address any of the eight DACs and set and hold their output voltage through a latch.

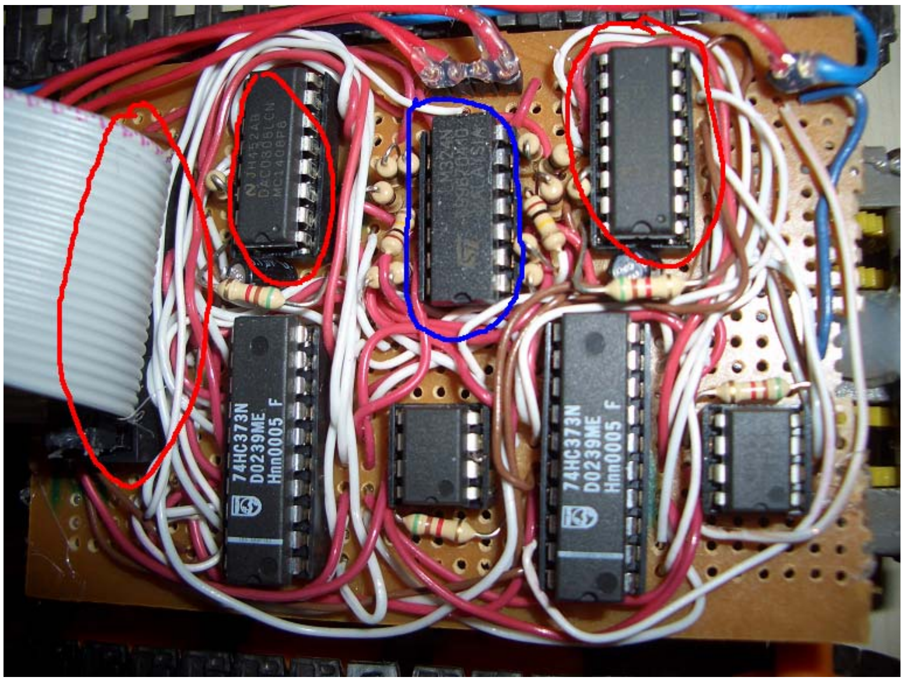


Figure 4.17. Dual DAC0808 on a tank segment fed through a databus. The outputs of DACs are configured to be between 0 – 5 volts.) For two directional DC motor control, the output of each DACs are readjusted between -5 <–> +5 volts by a quad operational amplifier LM324N marked with blue in this figure. The adjustment operation is done by first amplifying the DAC output by a gain of 2 then adding a -5v offset.

4.3. Low Level Control

The implemented snake-like robot has a total eight dc motors (two per each robot element) which are electronically speed controlled and three servos which are position controlled.

4.3.1. Palette Speed Control

The speed of each robot element, called segments thereafter, is generated at the palettes by the corresponding DC motor and coupled gearboxes. For proper realization of snake gaits, the speed of the segment palettes must be speed controlled via dc motors, but unfortunately the precise dc motor model is not known. Building a low level controller with complicated feedback such as a feedback of actual speed counting the RPM of the palette is very cumbersome so a different and basic approach is followed. Consider the electrical model of a dc motor:

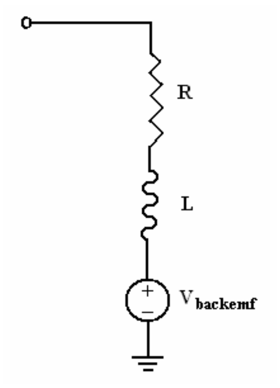


Figure 4.18. DC Motor Model

The back electromotor force potential $V_{backemf}$ is proportional with the turning speed (RPM) of the motor. So the speed can be controlled by a feedback of$V_{backemf}$. It should be noted that a controller can be built to control the speed by reducing the error by the desired $V'_{backemf}$ and the actual$V_{backemf}$; but the relationship between actual$V_{backemf}$ and the speed of the motor is not precisely known.

The gearboxes coupled to the dc motors have a gear ratio of 201:1 which has a very high low pass effect. This very high gearbox ratio decreases the maximum speed of the robot but enables the use of more basic controllers. In this case a very simple proportional controller is sufficient to control the speed keeping the low control of the robot more basic and reliable.

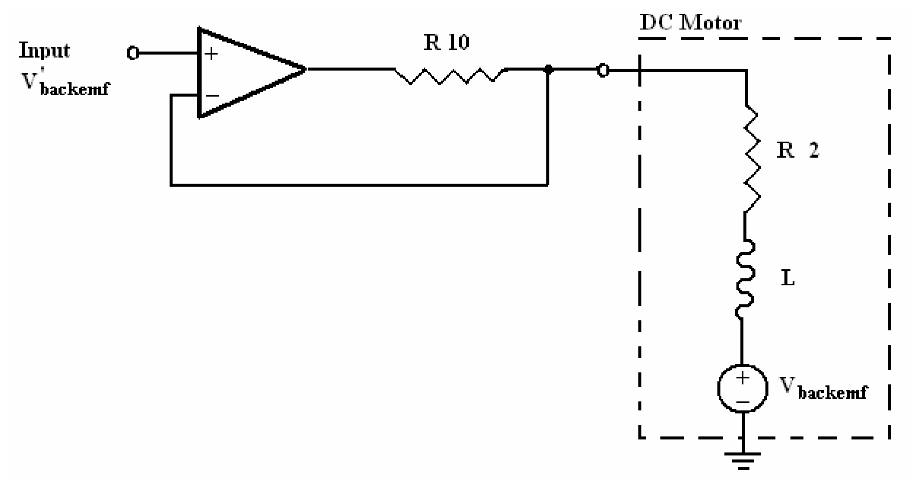


Figure 4.19. DC Motor Low Level Controller.

The proportional controller is realized by an operational amplifier. The serial resistance with 10 ohms is much greater than the internal armature resistance of the dc motor which is about 2 ohms so the voltage drop on the internal armature resistance can be neglected. For simplicity the inductance of the dc motor can also be omitted, thus with this scheme the back emf is obtained between the terminals of dc motor, which connect to the inverting input (-) of the operational amplifier (figure 4.19). The desired back emf ($V_{backemf}^{'}$) is supplied to the non-inverting input of the amplifier, and the amplifier assures the actual $V_{backemf}$ is equal to $V_{backemf}^{'}$ in its saturation limits.

The dc motors require high input currents which can not be supplied with a standard operational amplifier, so a push – pull type of buffer is used to assist the operational amplifier. The push – pull type of buffer consists of two pnp and npn type of bjt (bipolar junction transistor) transistors (figure 4.20).

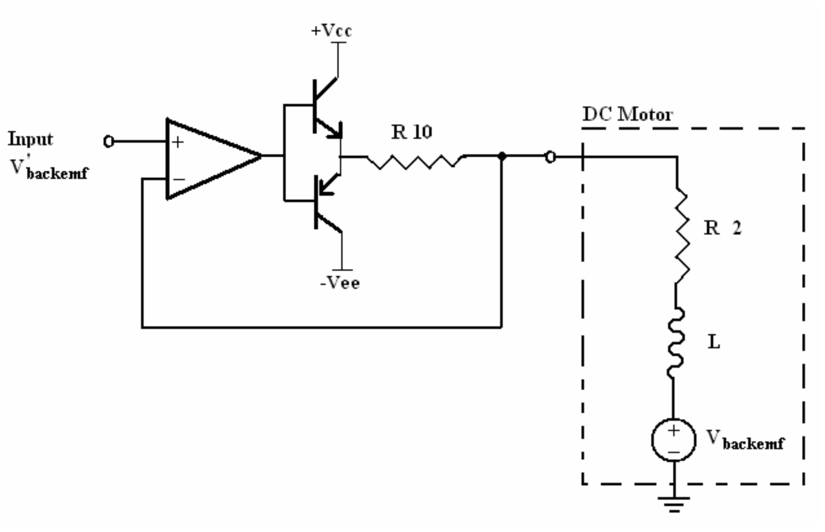


Figure 4.20. Schematics of DC motor low level controller.

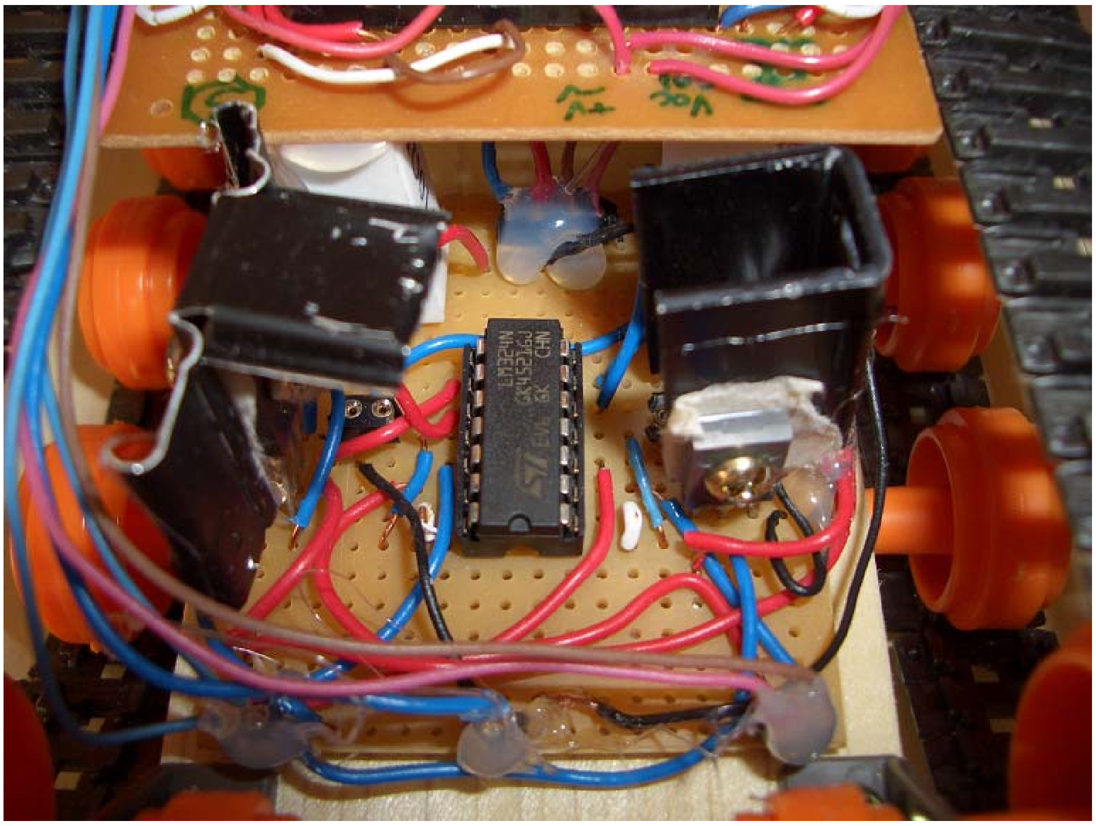


Figure 4.21. DC motor low level controller. In the figure a dual low level DC motor controller is shown which controls the dc motors of a tank segment. The integrated circuit (LM 324) in the figure contains dual operational amplifiers. The two big heat sinks are used to cool two pairs of BD 125 and BD126 bjt transistors.

4.3.1.1 Input – Output Relation

One of main drawbacks of the low level control induced for palette speed control is that it focuses on the measurement of a desired back emf voltage. But the relation between back emf and the actual speed of the palettes are still unknown.

The relation between the back emf values and the resultant palette speeds are derived by taking samples running from segment palettes and fitting a separate fifth order polynomial to the corresponding samples.

Table 4.1. Back EMF versus Speed

cm/sec	Back EMF (V)											
	0	0.68	0.76	0.84	0.96	1.04	1.12	1.2	1.28	1.36	1.44	1.52
Palette 1	0	1.2	1.3	1.5	1.7	2.1	2.3	2.5	2.8	3.3	3.5	3.7
Palette 2	0	1.4	1.7	2	2.1	2.3	2.8	3	3.2	3.4	3.5	3.8
Palette 3	0	1.11	1.14	1.2	1.5	1.9	2.2	2.4	2.7	2.8	3.1	3.3
Palette 4	0	1.1	1.3	1.6	1.8	2	2.3	2.4	2.7	3.1	3.5	3.7
Palette 5	0	1.7	1.9	2.1	2.2	2.7	2.9	3.1	3.3	3.4	3.8	4.1
Palette 6	0	0.6	0.7	1	1.4	1.5	1.7	1.8	2.2	2.5	2.6	2.9
Palette 7	0	0.9	1	1.01	1.1	1.6	1.7	1.8	2.2	2.32	2.38	2.85
Palette 8	0	1.2	1.3	1.4	1.7	1.9	2	2.3	2.5	2.56	2.85	3.22

4.5. Control of the Mechanical Robot

Since the simulation mode possesses all the mechanical robot's all dynamic and kinematic properties in addition to the low level control, if we tune controllers in the simulation environment those trained controllers should be able to control the mechanical robot. However the hardware has severe limitations such as lacking the ability of rising specific segments, or not being equipped of prismatic joints that enables elongation. Because of these limitations, the controller network of the robot model has to be suitably truncated.

The robot is equipped with accelerometers on each segment, but these accelerometers practically can not be used to detect slippage as it is in the simulation. In the simulation environment, simply integration of outputs of the accelerometers gives linear speed; integration of linear speed gives the position so the slippage can easily be determined in a simulated case. But in practical realization, due to noise and vibrations; the outputs of the accelerometers are noisy and integration of these noisy outputs will result in errors growing in time. Also the electronic realization of the robot which utilizes a "sample and send" strategy is not suitable for processing the outputs of the accelerometers. The processing of the accelerometer outputs, rejecting noise, is a signal processing problem and should be handled onboard the robot. Due to the limitations and unsuitability, the slippage detection and its coupled extension to selecting the best gait will not be utilized on the mechanical robot.

To summarize, the mechanical robot;

- Can not perform rectilinear and sidewinding gaits since no segments can be lifted up,
- Can not sense the friction of the environment,
- Can not detect, or track an object (debri) or goal point.
- Can perform lateral undulation with obstacle avoidance,

- Can perform an adapted accordion gait with obstacle avoidance,
- Can sense the tilt orientation of each tank segment through the two dimensional accelerometers.

In addition to the mechanical limitations, the robot also has imperfect low level control, joints with low stiffness, IR sensors with noisy and short ranged output. With all these induced impairments, the mechanical robot will only be controlled to conduct lateral undulation and accordion gaits with obstacle avoidance behavior.

4.5.1. Lateral Undulation for Mechanical Robot

Mechanical robot is technically able to perform identical lateral undulation as the robot model. The impairments of the low level control of tank palettes would be compensated by actively driven yaw axis of the joints.

One of the main problem is that the sensors on the mechanical robot has a range of 80cm with noisy output, where in the simulations IR sensors could be able to detect obstacles from a maximum distance of 200 cm without any noise. The interface of the mechanical robot and the obstacle avoidance controllers of simulation model can be made by multiplying the outputs of the IR sensors according to fulfill input range of the controller. Since the outputs of the IR sensors are continuous no states will be overlooked. But the controller may need to be set more "reactive" by increasing its steering actions. The sensor issue will be discussed in results section.

4.5.2. Accordion for Mechanical Robot

The accordion gait can not be directly performed by the mechanical robot since no prismatic joints are present. But a similar motion can be obtained by moving the pallets of the tank segments at the same time. The steering can be done as the original accordion gait by giving the body a constant curvature. This gait is advantageous because the tank segments push each other at the steering direction directly generating a high forward force.

The steering of accordion gaits requires high servo torque, because for steering the joint servos should overcome the friction force on the tank palettes, and give a curvature shape to the body. In the simulations, the servos were assumed to be ideal whereas in the rail life application it is not the case. The problem of insufficient joint servos will be discussed in the results section.

4.6. Mechanical Results

4.6.1 Accordion Gait

Accordion gait is implemented on the hardware robot using the obstacle avoidance controller developed and trained on the simulation model. The robot is set to move at 3cm/sec speed, and the steering is determined by the aforementioned obstacle avoidance controller that is run at every five seconds since the robot is moving slower than its simulation model. Unfortunately the joint servos of the robot did not produce enough torque to change the curvature of the body. So the servos are assisted by the palette motion which is introduces in figure 4.22.

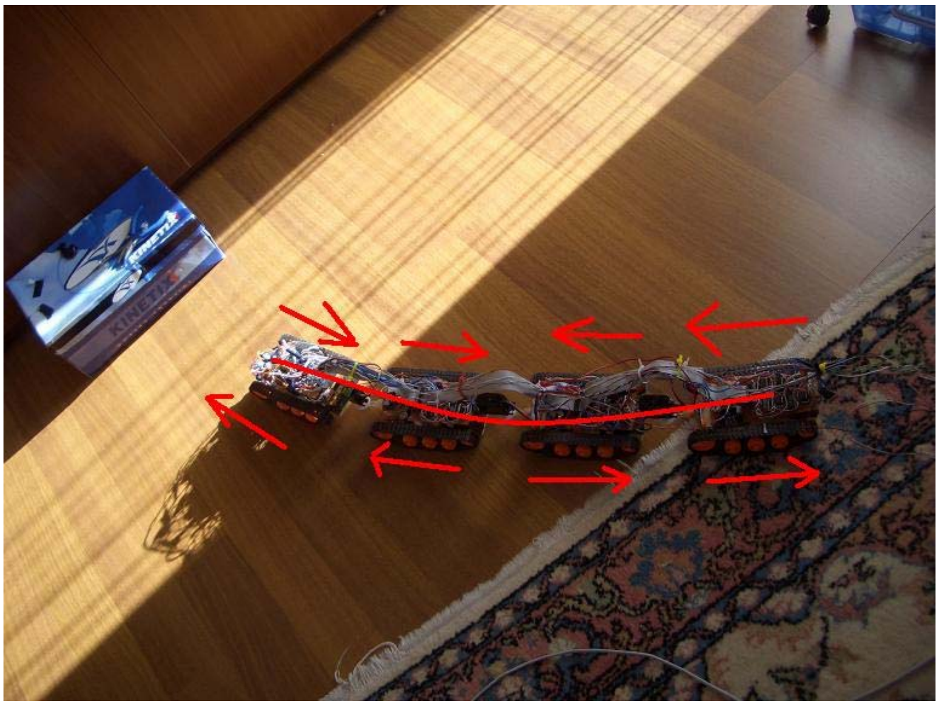


Figure 4.22. The steering of accordion gait. The joint servos are assisted by the palette motion indicated with red arrows.

The palettes move in specific directions to help the servos rotate the body. Assistance is provided to the servos for one second. After that one second, the palettes continue their ordinary corresponding motion.

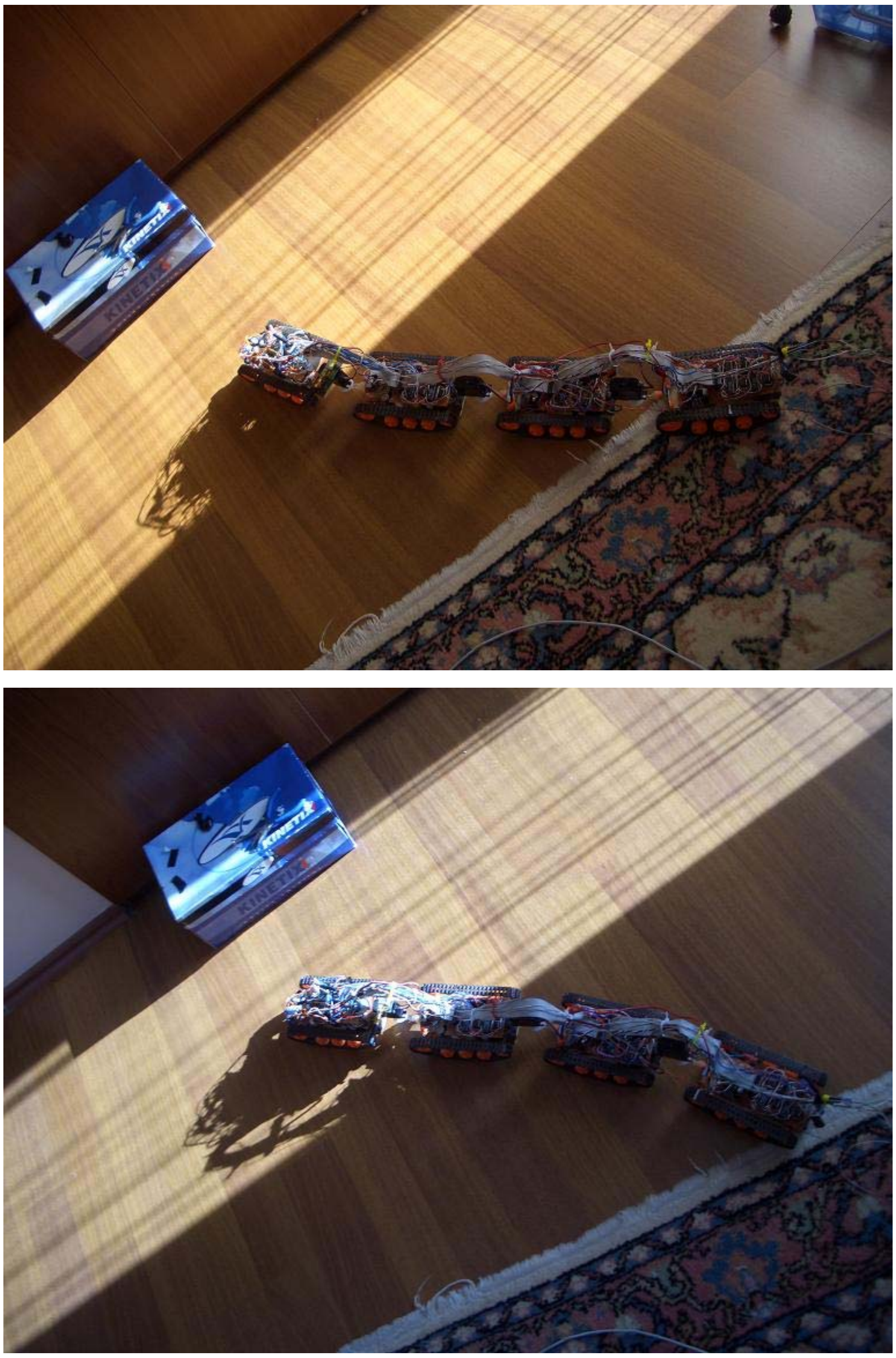


Figure 4.23. The robot performs accordion gait while avoiding the obstacles.

The robot performs accordion gait as demonstrated in figure 4.23. The robot senses the blue box, and steers left to avoid it.

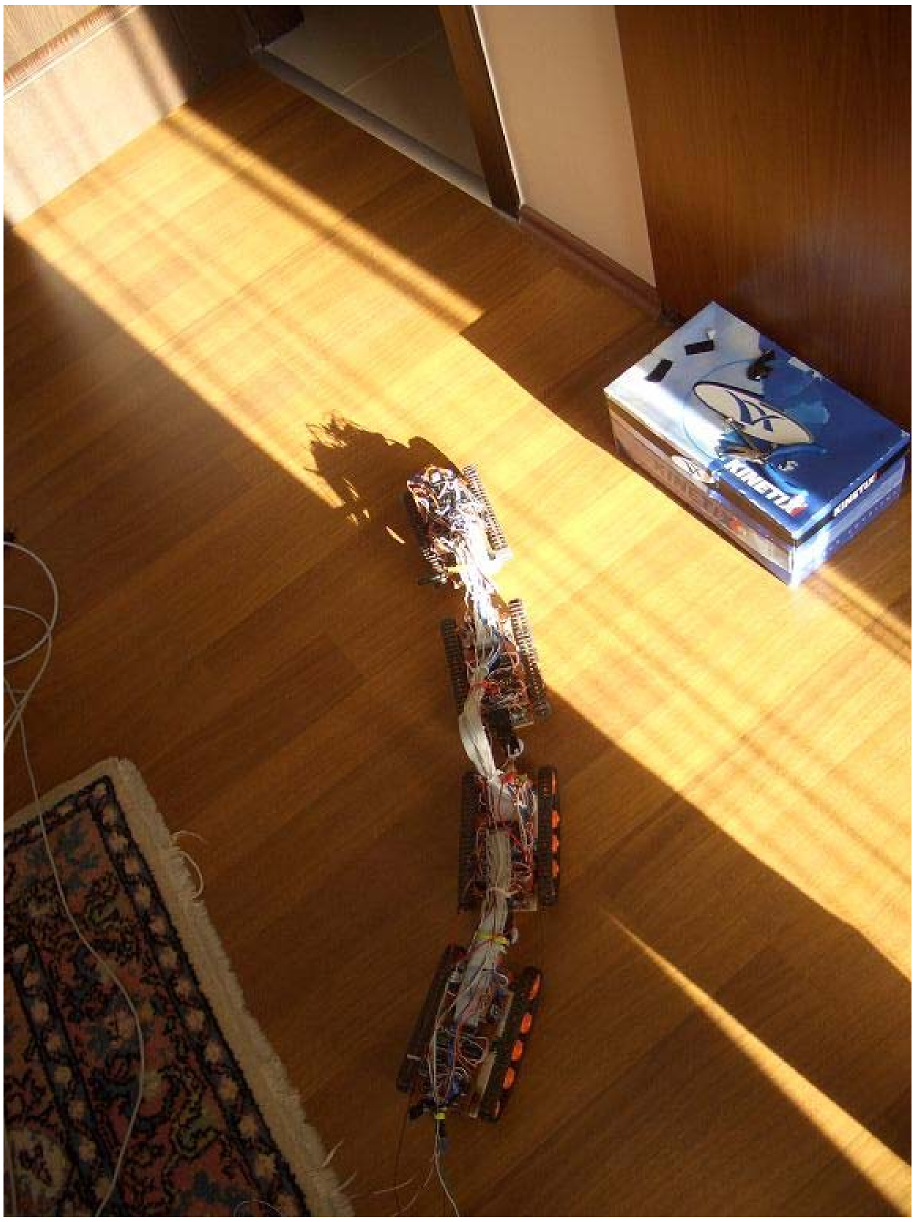


Figure 4.24. Accordion gait. The robot avoids the blue box.

The robot fully avoids the obstacle of blue box as seen in figure 4.24.

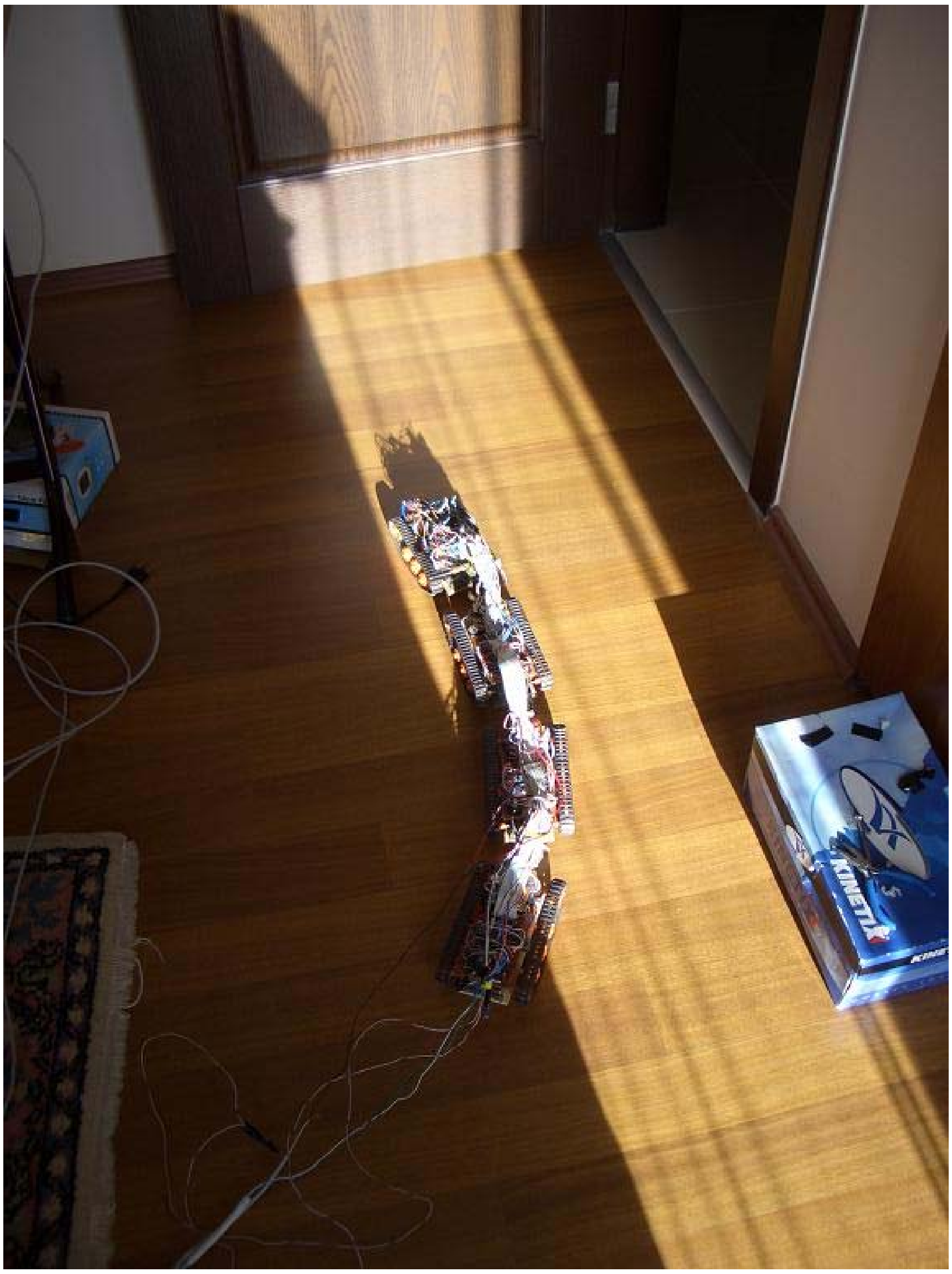


Figure 4.25. Accordion gait.

The robot continues its motion without any steering change since it did not sense a nearby obstacle as seen in figure 4.25.

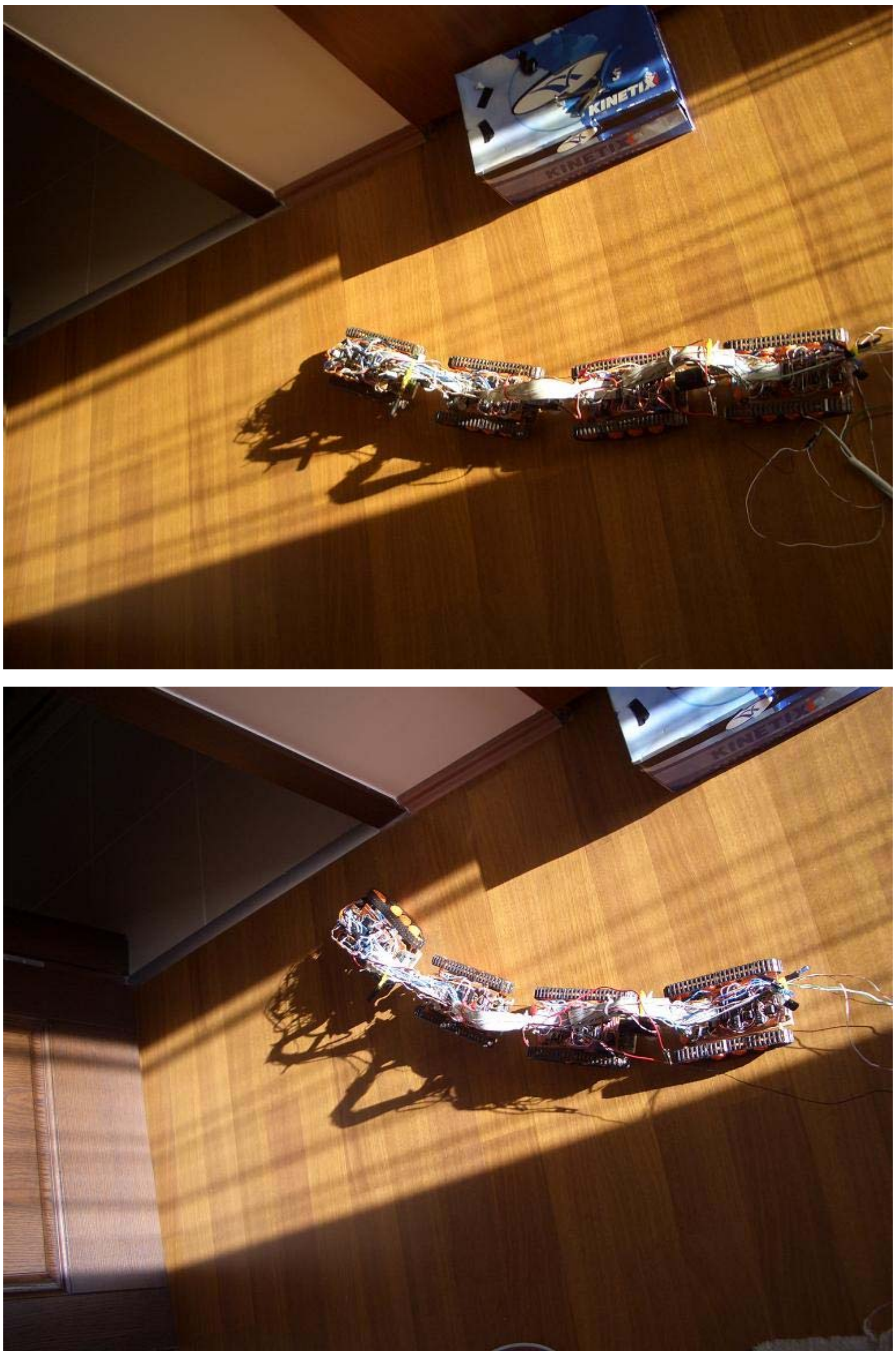


Figure 4.26. Accordion gait. The robot detects the door, and steers to avoid it.

In figure 4.26 the robot senses the door in the front and steers right. But this steering brings the robot dangerously close to the wall as seen in the lower part of the figure 4.26.

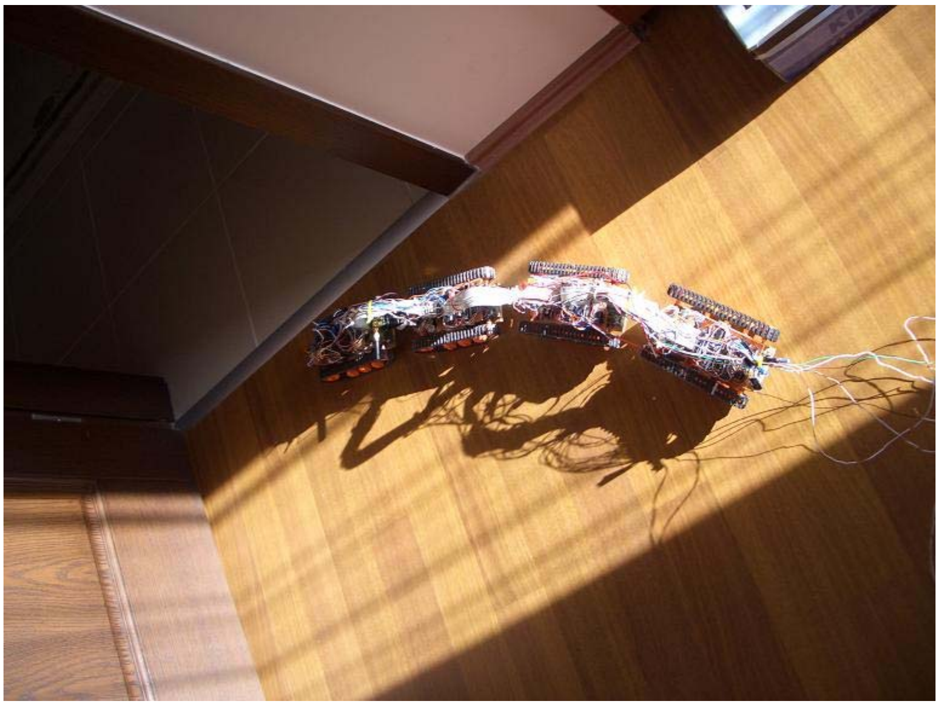
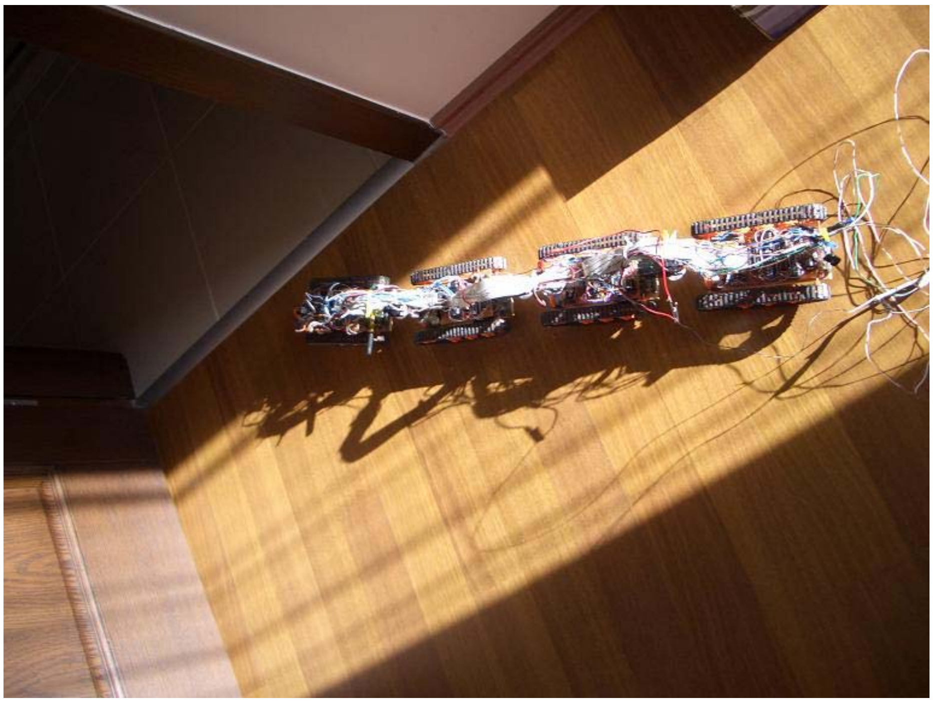



Figure 4.27. Accordion gait. After encountering the other side of the door entrance, the robot steers again to opposite direction.

The robot immediately steers to left in order not to collide with the wall as shown in upper part of figure 4.27.

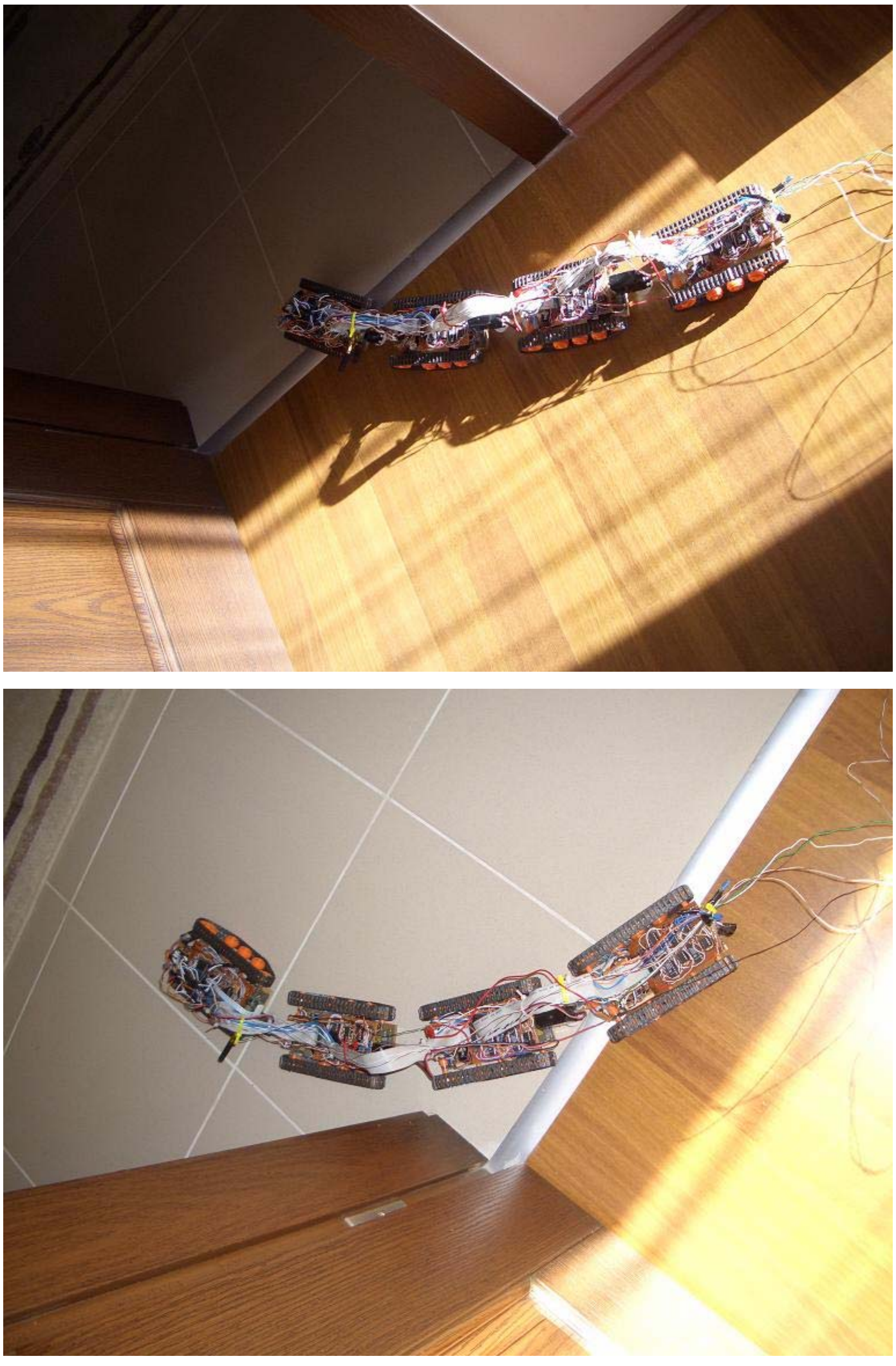


Figure 4.28. Accordion gait. The robot passes through the door way.

The robot makes a final right steering to go through the door passage as shown in figure 4.28.

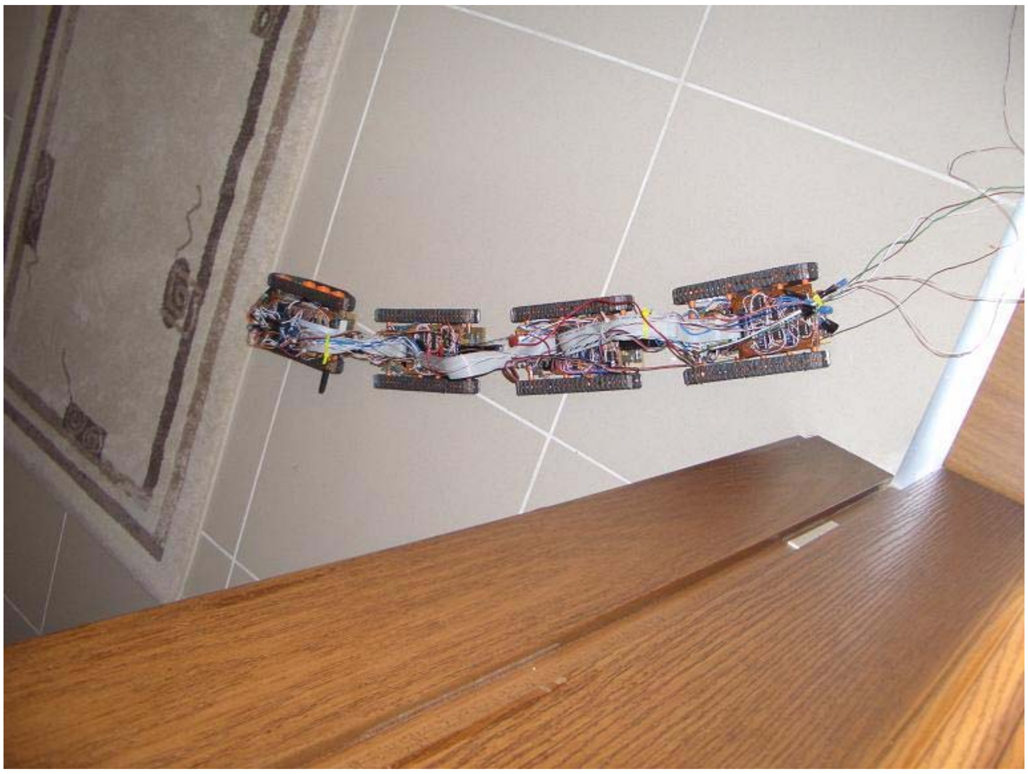


Figure 4.29. Accordion gait. The robot passes through the door way.

As seen in the figures 4.23 – 4.29, the robot is able to avoid surrounding obstacles by the accordion gait using the controller developed and first tested on the simulation model. But it must be noted that the mechanical robot can not handle obstacle avoidance problem for every situation it encounters. Especially if the robot encounters an obstacle ahead, it can not avoid it because the front sensors can not detect the obstacle until it is too late for the robot to avoid it. This issue was also present in the simulations but solved with increased IR sensor range and detection cone.

The robot's structure prevents sharp steering, which was achievable by a single robot segment itself, so perfect obstacle avoidance should not be expected.

4.6.2 Lateral Undulation Gait

The lateral undulation is performed with the same scheme which was introduced in section 3.2.1.

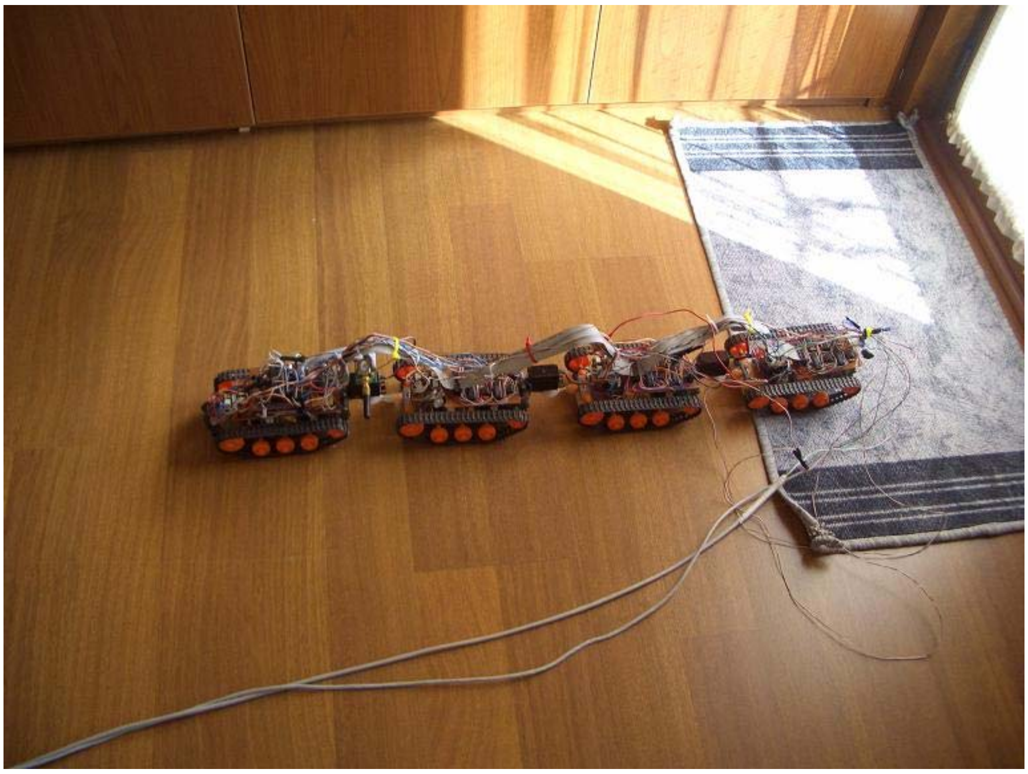


Figure 4.30. Lateral Undulation Gait without obstacle avoidance behavior.

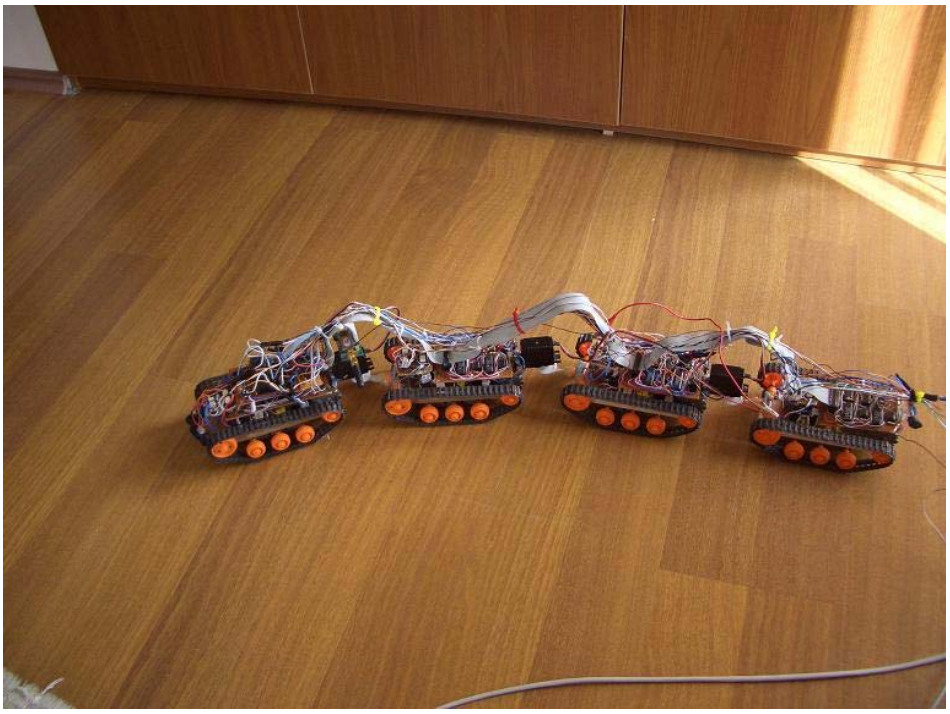
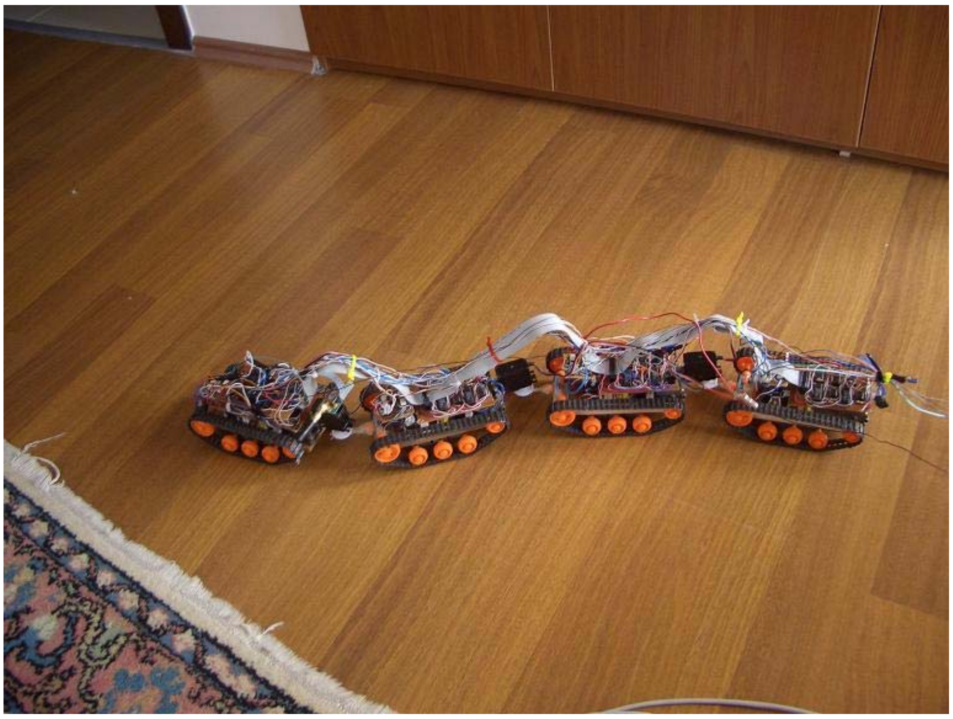



Figure 4.31. Lateral Undulation Gait without Obstacle Avoidance behavior.

Head segment steers to left and right alternatively as its successor segments follows the head's path as shown on figure 4.31.

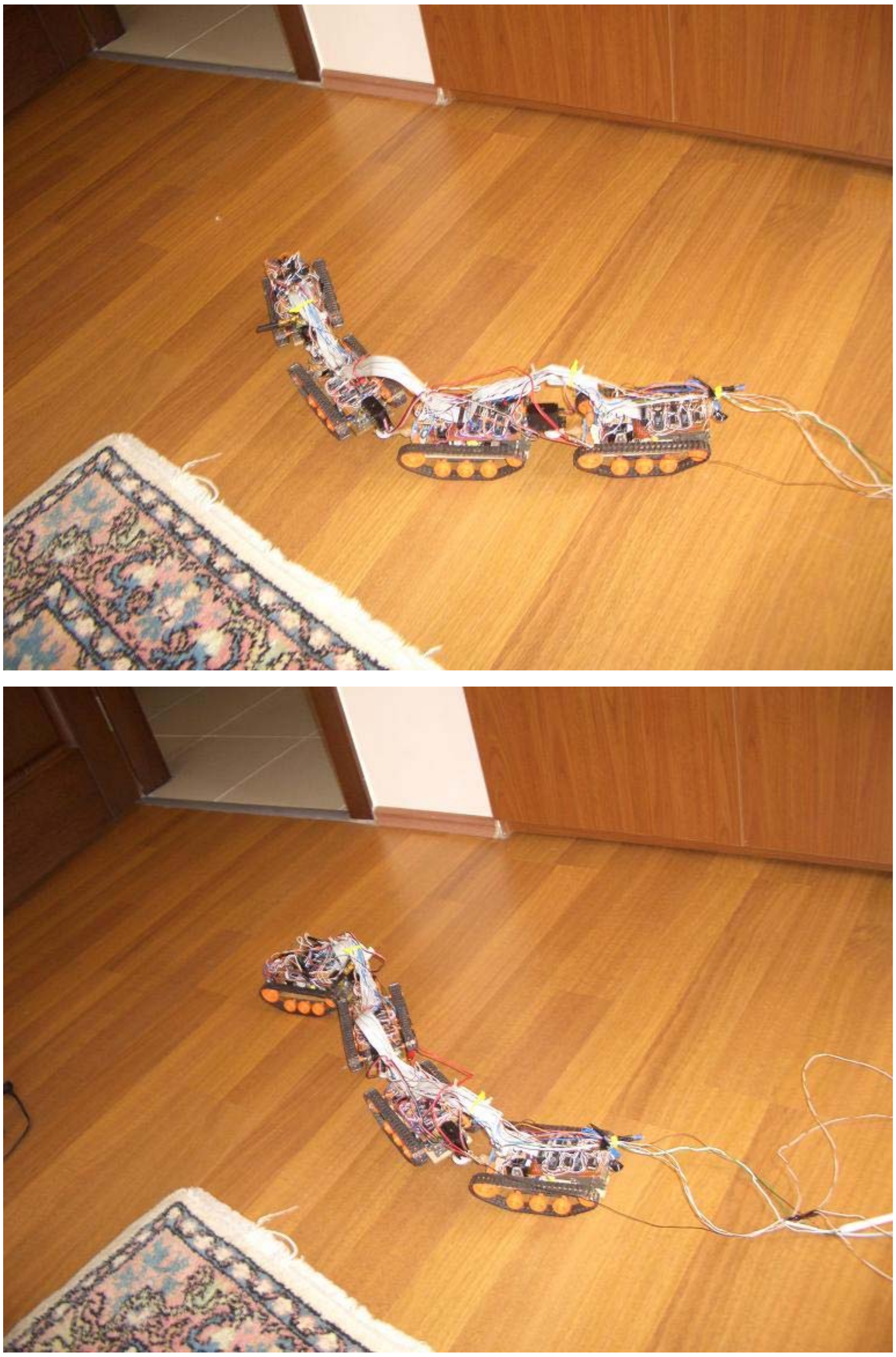


Figure 4.32. Lateral Undulation Gait without Obstacle Avoidance behavior.

The robot continues performing lateral undulation as shown in figure 4.32.

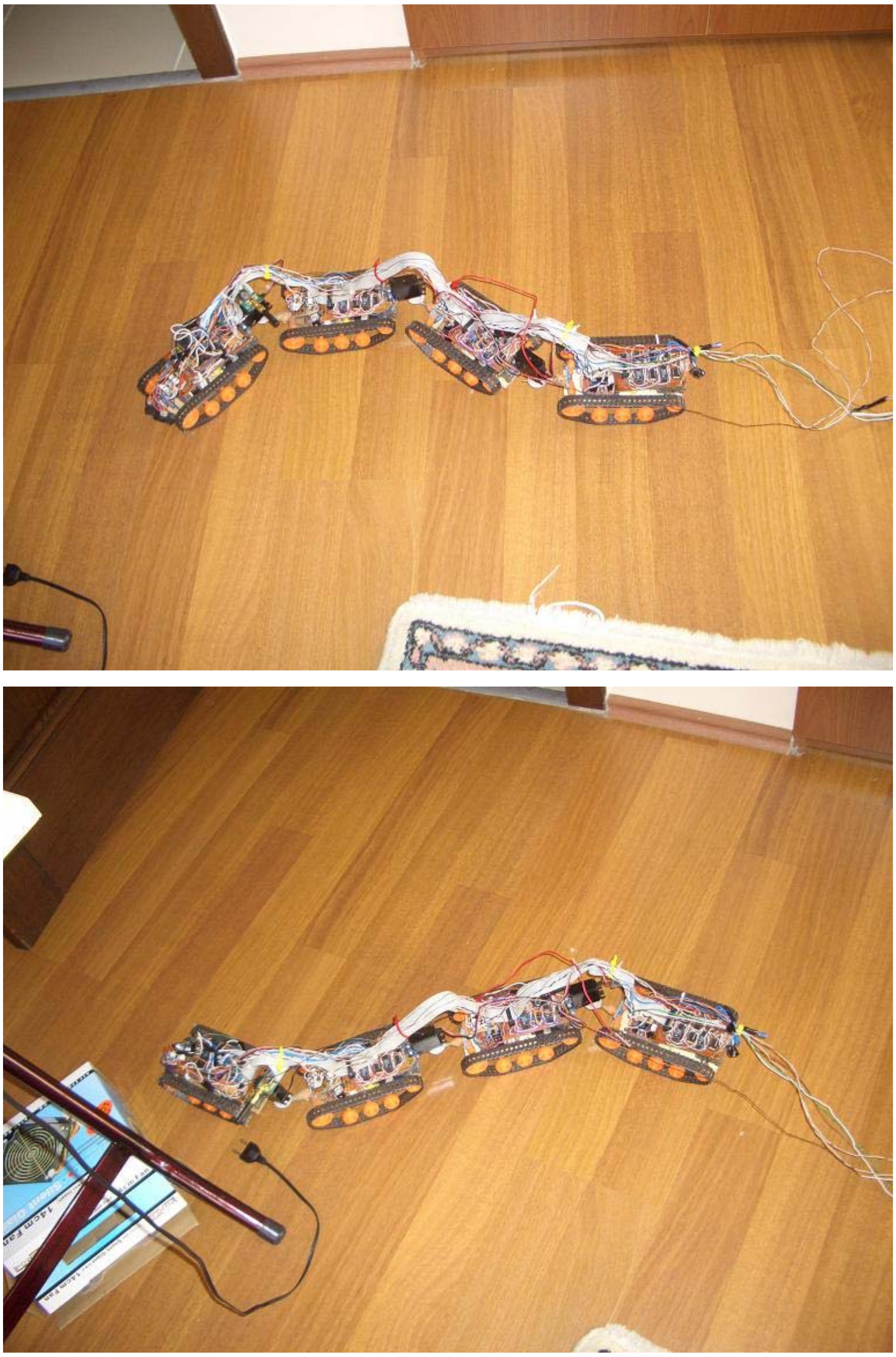


Figure 4.33. Lateral Undulation Gait without Obstacle Avoidance behavior.

The lateral undulation scheme worked on the mechanical robot as illustrated in figures 4.30, 4.31, 4.32, 4.33. Each segment followed the path of its predecessor as the head segment propagated by alternating steering. The success of the simulation model's lateral undulation scheme on hardware proves the fidelity of the model.

The lateral undulation obstacle avoidance controller of the model runs at 1Hz rate.

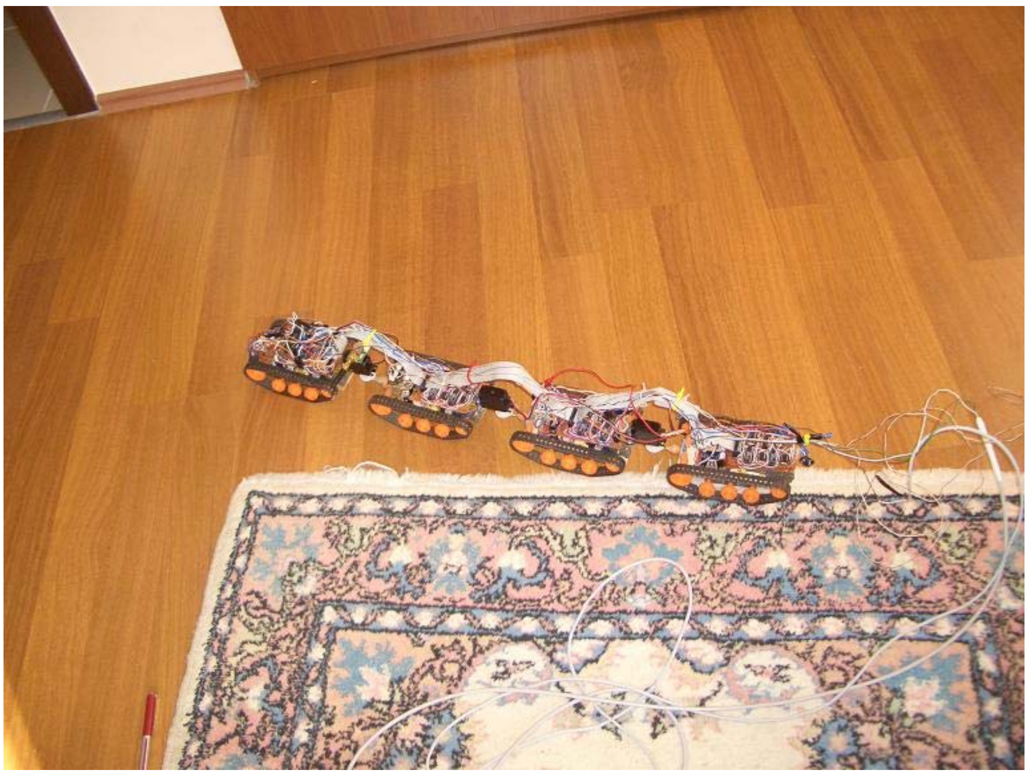


Figure 4.34. Lateral Undulation Gait with Obstacle Avoidance behavior.

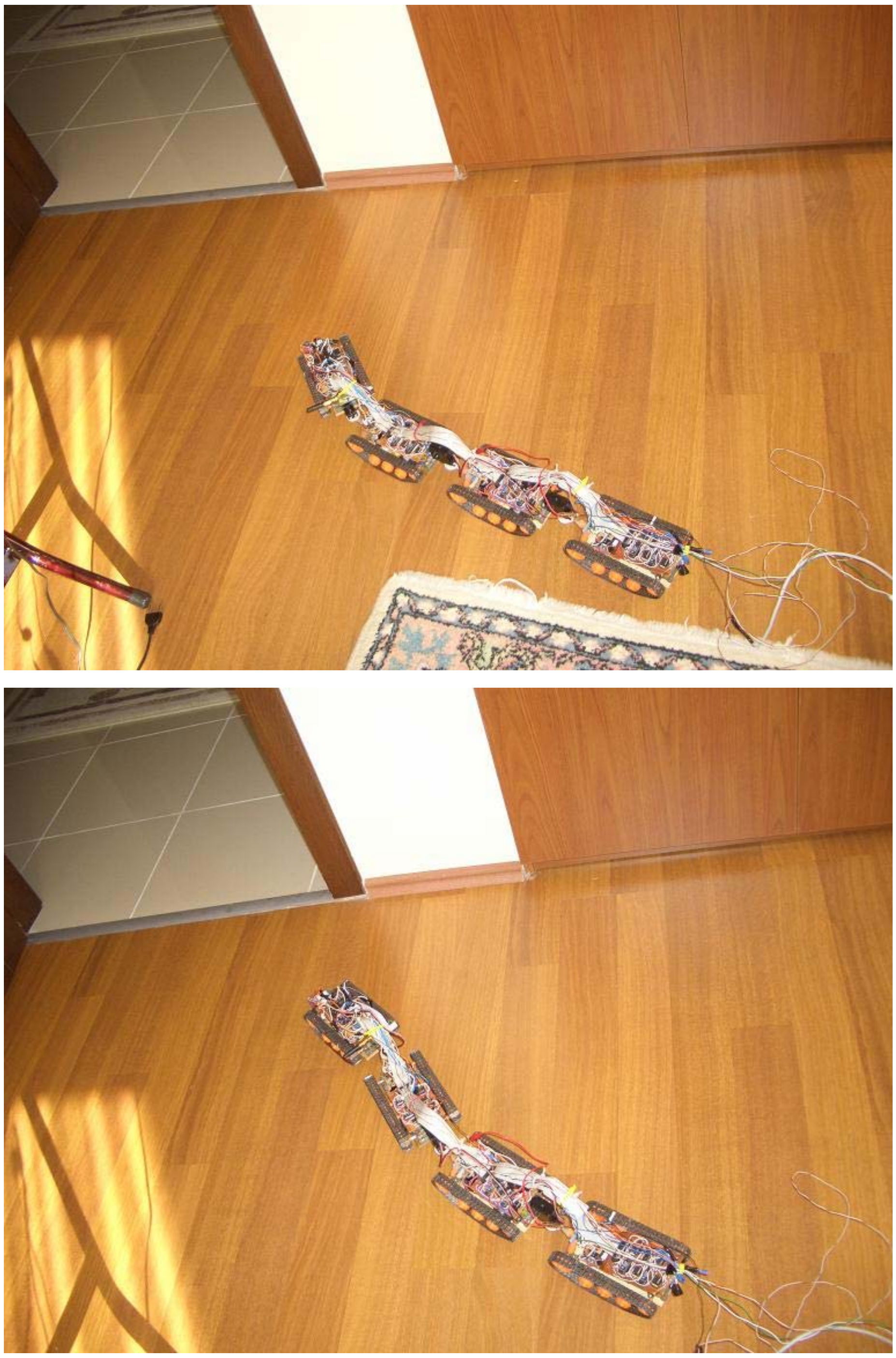


Figure 4.35. Lateral Undulation Gait with Obstacle Avoidance behavior.

As the robot undergoes undulations it senses the wall in its front and steers left in figure 4.35.

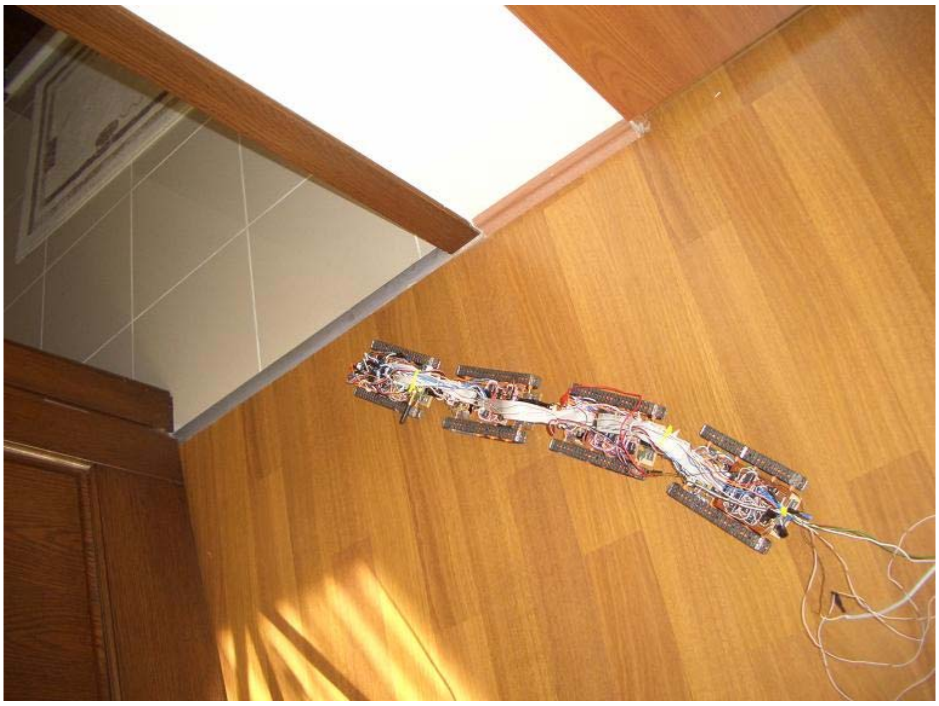
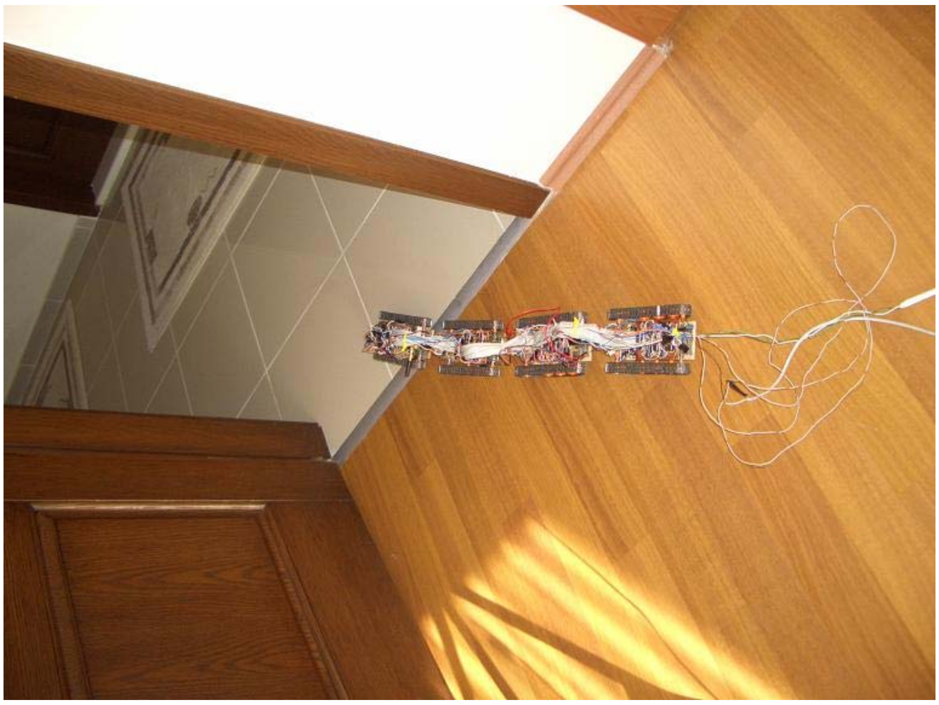



Figure 4.36. Lateral Undulation Gait with Obstacle Avoidance behavior.

The robot goes through the doorway as seen in figure 4.36. Steering of obstacle avoidance controller disturbs the body curvature yielding to a strait body segment orientation.

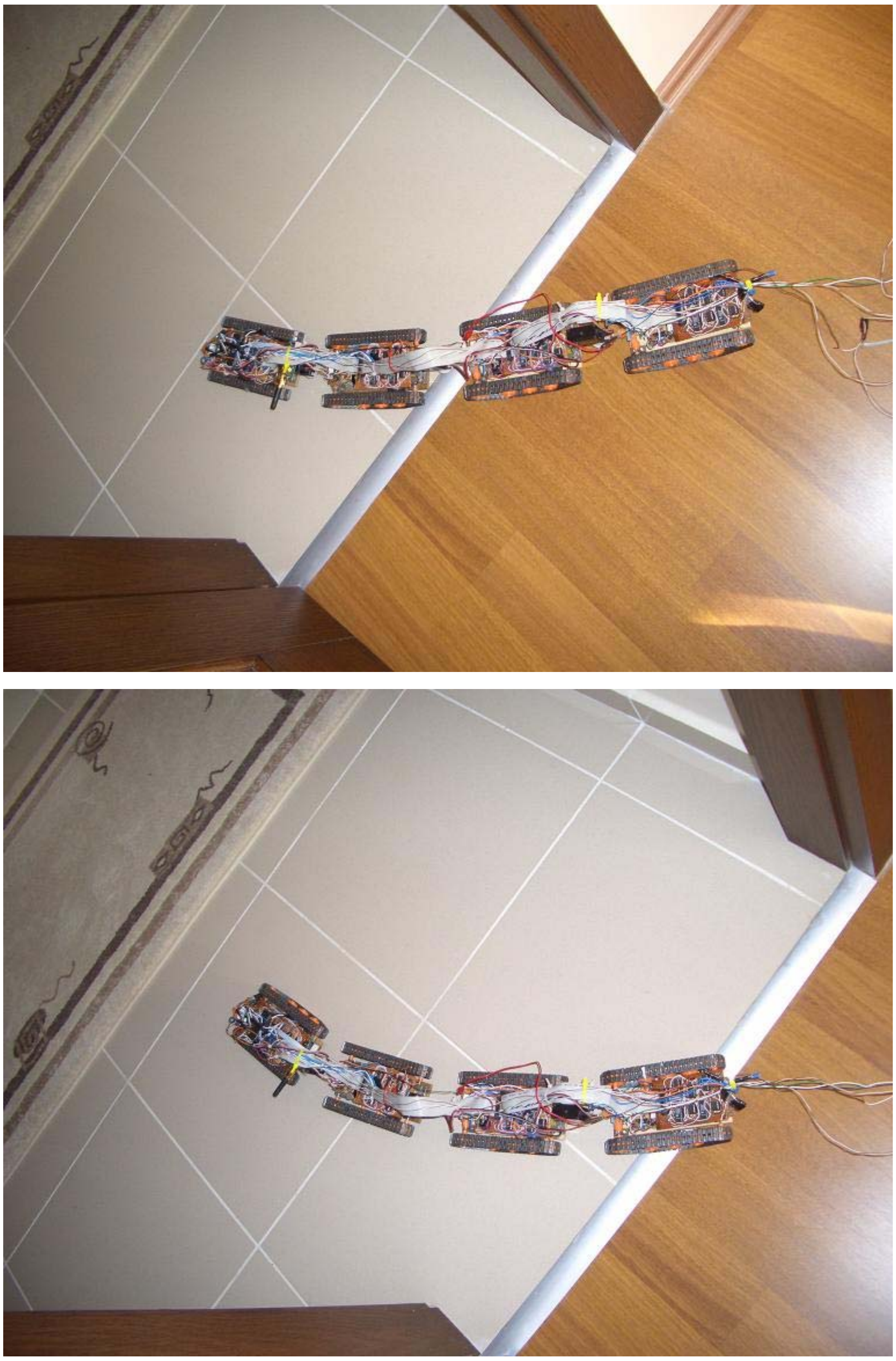


Figure 4.37. Lateral Undulation Gait with Obstacle Avoidance behavior.

The robot leaves the door passage in figure 4.37.

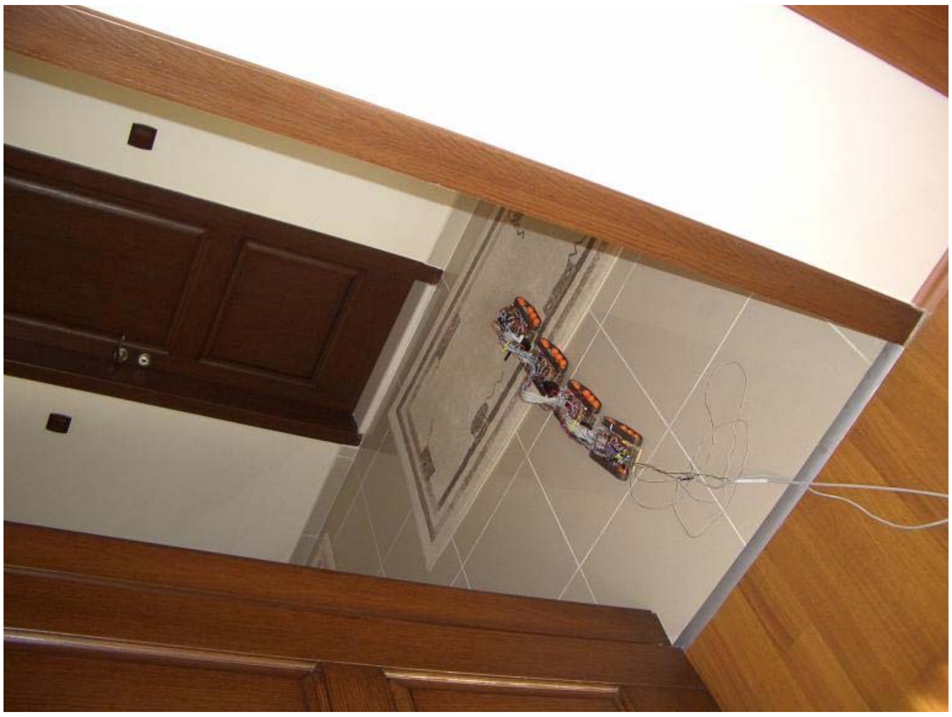
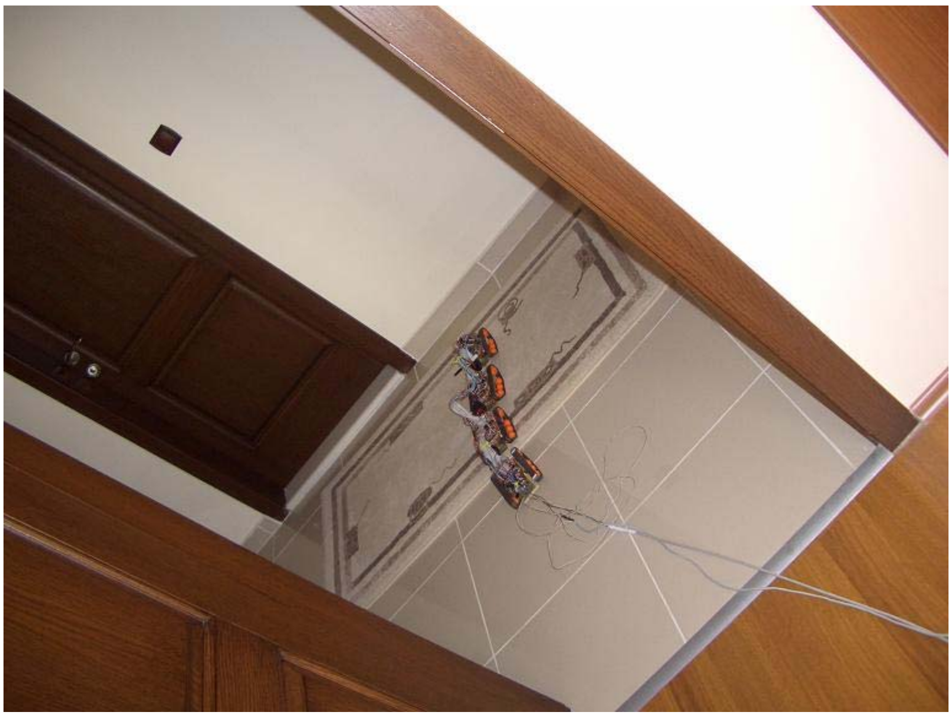



Figure 4.38. Lateral Undulation Gait with Obstacle Avoidance behavior.

The robot continues ordinary lateral undulation as it senses no nearby obstacles in figure 4.38.

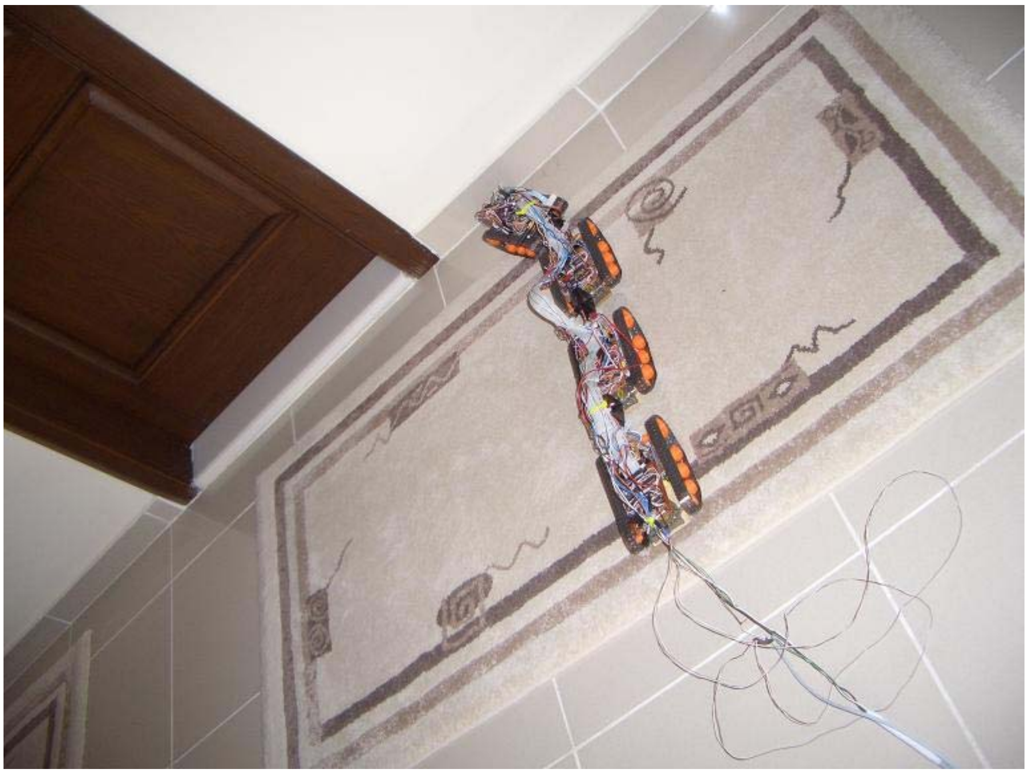


Figure 4.39. Lateral Undulation Gait with Obstacle Avoidance behavior. The robot crashes into the wall.

As seen in figures 4.34 – 4.39, the robot avoids obstacles and goes through the door way. But when it encounters a frontal wall, it collides even though it tries to avoid by steering to left as shown on figure 4.39.

Undulations are sometimes suppressed by the obstacle avoidance controller. The obstacle avoidance controller gives outputs which contradicts with the lateral undulations, and makes the robot move in a strait direction. In the simulations, the frontal range sensors of the robot head were modeled so as to always sense in the frontal direction as shown in figure 4.40. The mechanical robot lacks this adjustment, so it treats undulation as a steering. As the mechanical robot makes undulations, its frontal sensors momentarily sense the nearby obstacles as frontal obstacles which were infact not on the global direction of the robot. Thus the obstacle avoidance controller steers the robot on the opposite direction of the undulation so it suppresses the undulations even though the robot was not proceeding to an obstacle.

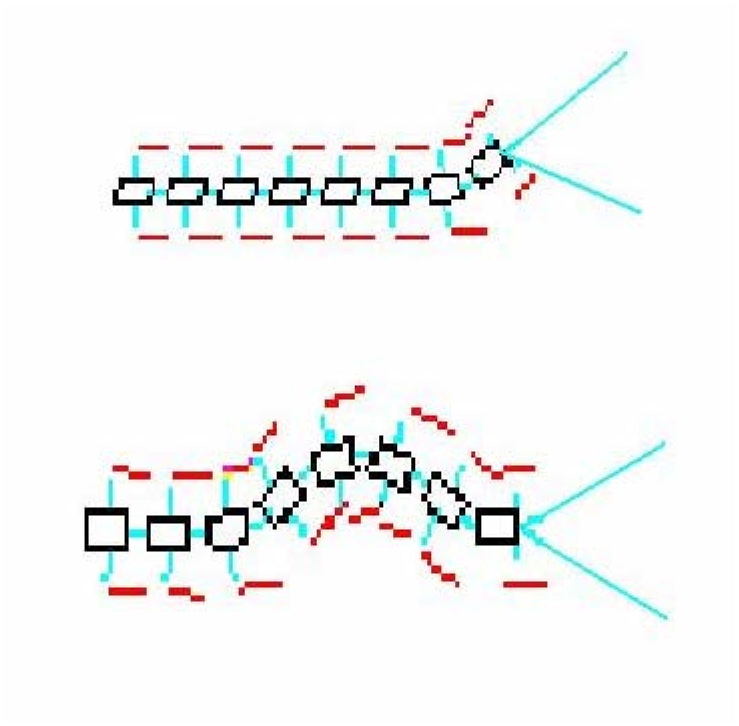


Figure 4.40. Lateral Undulation of the model. Frontal sensor directions are indicated by long blue lines.

CHAPTER 5

CONCLUSIONS

This thesis aims at the construction and control of a mechanical snake robot for SAR operations. A snake robot as described in section 4.1 which has similarities with “Kohga” (section 2.1.7) is realized and a simulation model of the realized robot is constructed with SIMULINK. The control mechanism for the hardware robot solves the problem of navigation in an unknown, variable friction environment by applying appropriate snake gaits. Also some extra capabilities are given to the robot such as debri (object) reaching and manipulation.

The main reasons for performing snake gaits are to capture some of the biological snake’s features and to have a platform which is suitable for SAR operations. The architecture of the implemented mechanical snake robot is built of interconnected tank segments; and a mechanical design possessing coupled mobility architecture achieve to have similarities with biological snakes for motion and navigation.

Considering mechanical limitations of the hardware robot, its simulation model is developed as a demonstrator of its full capabilities. Avoiding some limitations and impairments of the physical implementation, the simulation robot model possessed extra features such as being able to lift specific body parts and sensing slippage. But despite the addition of extra features, the simulation is kept to have similar dynamical and kinematical properties than its mechanical counterpart.

Several snake gaits which are derived and adapted from the real snakes' motions are applied to the hardware robot model and resultant motions are observed and discussed. Gaits showed different characteristics due to the clear difference of the hardware robot model from that a biological snake. These different characteristics are not unexpected, and these differences are acceptable since a snake robot strictly similar to the biological counterpart is beyond the aim of this thesis.

For the control task of the snake robot, obstacle avoidance and object reaching behaviors are integrated to the adapted gaits. The uncertainty in obstacle avoidance behavior is solved by self trainable FACL controllers for each gait. The object reaching behavior problem is solved by manually setting the look-up tables of the fuzzy controllers. Since object reaching is a simple task, usage of simple controller reduced the complexity of the simulation.

For the high level control of the snake robot, each adapted gaits with two behaviors are selected by a high level controller. The high level controller aims to "select the best applicable gait". The high level controller is also a fuzzy logic controller utilizing FACL learning scheme which is equipped with a reinforcement indicating the handling of slippage. For this indication first an examination of the adapted gaits is carried out by observing the friction characteristics of the adapted gaits on surfaces with variable friction. After the observation a threshold is determined to handle slippage.

Finally after training all controllers, some rule based actions are added to the controller network to manipulate an object using robot body under specific situations. The rules of object manipulation did not generate uncertainties, and are defined by user intuition.

The training of controllers is performed by a lower to upper procedure. First the lower level controllers are trained standalone. The higher level controller is trained using the already trained low level controllers.

The results of the simulations are obtained in arbitrary environments, which may include obstacles of any shape, size and number and variable friction. The controller network accomplished to solve the navigation problem derives an optimal policy. The derived policy didn't possess a single dominated choice for all situations. This is because applications of different choices are advantageous in some situations proving the usefulness of richness of information: With proper exploration of the FACL controllers, no choice is dominated. The examples of such situations are when friction is high and all obstacles are far away, selection of both sidewinding and lateral undulation is equally preferable. Another example of these situations occurs when friction is low and a nearby obstacle is present. In such case, the high level controller may select among rectilinear or accordion gaits. If one desires to avoid switching between gaits and have one gait selection per situation, the exploration and exploitation balance of the high lever controller should be broken toward favoring exploitation.

The object manipulation capability of the implemented model is very useful in SAR (search & rescue) operations. The location of disasters like collapsed building includes debris where navigation by treating debris as obstacle and trying to avoiding them may be impossible. The proposed control scheme approaches detect debris and manipulates them to a nearby stationary obstacle, clearing the environment from distributed debris. If the manipulation somehow becomes impossible or a pop up debris appear on the side of robot, the robot is also able to push away the debris by its body to a safer distance where the debris does not interfere with the subsequent robot navigation.

Some features of the simulation are not implemented on the mechanical robot such as debri detection and approaching or slippage detection because these features are practically complex on hardware. Differentiating debris from a stationary obstacle, tracking the debris in an unknown environment and sensing slippage in the practical applications is a very big problem to be solved. Although simulations had these practical capabilities, it opened a lot of issues for future works.

The mechanical robot is premature when compared to the abilities of its simulation. Despite these impairments, the mechanical robot is controlled to perform lateral undulation and accordion gaits while avoiding obstacles. The mechanical robot was directly controlled by a truncated controller of the simulation. The obstacle avoidance controllers are not tuned and optimized on the mechanical robot because the mechanical robot sensors had noisy measurements. In real life applications, IR sensors working together jammed the measurements of each other resulting in inconstant deviation on the distance measurements. The IR sensors on the mechanical robot had not enough range. With the current sensors and their orientations the robot can not handle the obstacle avoidance problem for every possible situation.

Some main advantages of the snake gaits such as redundancy, traction, and flexibility are tried to be captured making the proposed snake robot applicable for SAR operations. In summary the mechanical hardware robot has:

- Good traction due to coupled mobility,
- Redundancy,
- High forward propulsion force,

- Penetration abilities, which are beyond the capabilities of single vehicle robots, making itself suitable for SAR operations. A single vehicle robot producing enough propulsion and traction has to be big in size degrading the penetration abilities. But the implemented hardware robot's coupled mobility body properties allows generation of good traction and high propulsion force while still keeping the frontal cross-section of the hardware robot same with a segment of it body, ensuring high penetration ability.

FUTURE WORK

Future work of this thesis can be focused on the implementation of the missing features of the mechanical robot. In addition to the abilities of the robot simulation, new features regarding SAR operations can also be developed and adapted. Suggested features are as follows:

- Slippage determination in real applications,
- Goal point determination, and reaching by various sensors (e.g. a camera or a directional microphone),
- Palette speed controller with better accuracy, (shaft encoder)
- Deployment of more rigid robot structure encapsulating electronic components and own power source.
- Actively driven joint pitch axis enabling stair climbing,
- Grasping and enwrapment abilities.

The determination of slippage requires more than just an accelerometer. For effective slippage determination, the outputs of the accelerometers should be sampled and processes onboard the robot itself requiring a DSP card. The control task of the robot can be carried on the onboard DSP card, making the robot a standalone platform. Also for the speed control of the palettes, a shaft encoder can be coupled to the dc motors improving the accuracy of palette speed control.

The proposed snake-like robot tries to navigate in the environment without colliding and slippage by performing snake gaits. But in real life applications the snake-like robot should chase an aim position or direction instead of navigation around unconsciously. So as a future work an addition of goal point or direction to the control of the snake robot is plausible. In simulation adding a goal point or direction is quite easy but in the mechanical design it is cumbersome. But a goal can be implemented on the robot, recognizing a pattern such a human voice; this will yield a more useful robot for SAR. Also different goal points/directions determination such as a path for evacuation from the disaster site coupled with grasping or enwrapment techniques would enable the robot to rescue a victim from the disaster site.

Navigation in unknown environments with obstacles can be conducted more efficiently by interacting with obstacles instead of always trying to avoid them. The motion of the body can be assisted by using support gained from the obstacles as stated in [19].

In this thesis the energy consumption of the snake gaits selection was omitted. Considering the energy consumption during selection of the gaits may result in more effective navigation solutions.

Finally simulations can be made more realistic by 3D modeling.

REFERENCES

[1] Snakes and Strings: New Robotic Components for Rescue Operations; Shigeo Hirose and Edwardo F. Fukushima Tokyo Institute o f Technology, 2-12-1 Ookayama Meguro-ku, JAPAN

[2] Serpentine Locomotion with Snakes; Masashi Saito, Masakazu Fukaya, and Tetsuya Iwasaki; IEEE Control Systems Magazine

[3] K. L. Paap, M. Dehlwisch, B. Klaassen, GMD-Snake: a Semi-Autonomous Snake-like Robot, In: Distributed Autonomous Robotic Systems 2, Springer-Verlag, Tokyo, 1996

[4] Conradt, J., and Varshavskaya, P. (2003). Distributed Central Pattern Generator Control for a Serpentine Robot. Proceedings of the Joint International Conference on Artificial Neural Networks and Neural Information Processing (ICANN/ICONIP), Istanbul http://www.ini.unizh.ch/~conradt/projects/WormBot/

[5] Kousuke Inoue, Shugen Ma and Chenghua Jin "Neural Oscillator Network – Based Controller for Meandering Locomotion of Snake – Like Robots" International Conference on Robotics & Automation, New Orleans, LA April 2004

[6]Bernhard Klaassen, Karl L. Paap: GMD-SNAKE2: A Snake-Like Robot Driven by Wheels and a Method for Motion Control. ICRA 1999: 3014-3019

[7]Development of Genbu Hitoshi KIMURA, Shigeo HIROSE, Tokyo Institute of Tech. 2002 IEEE

[8]Development of The Snake-like Rescue Robot "Kohga" Tetsushi Kamegawa, Tatsuhiro Yamasaki, Hiroki Igarashi and Fumitoshi Matsuno Tokyo Institude of Technology,2004 IEEE

[9] 3-D Grasping During Serpentine Motion with a Snake-like Robot - Barış ATAKAN Robotics and Applications ~RA 2005~ http://www.actapress.com/Content_Of_Proceeding.aspx?ProceedingID=337

[10] 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

[11] Ari E.O., Erkmen I., Erkmen A. M., "FACL Based 3D Grasping Controller for a Snake Robot During Locomotion" IROS 2006

[12] 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.

[13] Jyh-Shing R. Jang "Self-Learning Fuzzy Controllers Based on Temporal Back Propagation", IEEE Transactions on Neural Networks, Vol 3, No. 5, September 1992.

[14] Hamid R. Berenji, Pratap Khedkar "Learning and Tuning Fuzzy Logic Controllers Through Reinforcemetns" IEEE Transactions on Neural Netwroks, Vol 3, No 5, September 1992

[15] Richard Sutton Sat May 31 13:56:52 EDT 1997 http://www.univ.kiev.ua/~yawd/books/AI/3/node1.html

[16] Lionel Jouffe "Fuzzy Inference System Learning by Reinforcement" 1998 IEEE

[17] Lionel Jouffe "Fuzzy Q-Learning" FUZZ 1997 IEEE

[18] http://reptilis.net/serpentes/moving.html

http://en.wikipedia.org/wiki/Snake

[19] Yansong Shan and Yoram Koren "Design and Motion Planning of a Mechanical Snake", IEEE Transactions on Systems, MAN, and Cybernetics, Vol. 23, No 4, July/August 1993

[20]Static Friction Models http://www.20sim.com/webhelp4/library/iconic_diagrams/Mechanical/Friction/Static_Friction_Models.htm

[21] Hee Rak Beom, Hyung Suck Cho "A Sensor Based Navigation for a Mobile Robot Using Fuzzy Logic and Reinforcement Learning" IEEE Trans. On Systems, MAN, and Cybernetics, Vol 25, No 3, March 1995

APPENDIX

A. MICROCONTROLLER DETAILED DESIGN

Atmel 89C52 is used to handle the sensor output sampling and, conduction of data distribution on the robot. 89C52 communicates with a PC through a 9600 bout serial RF (radio frequency) link.

A.1. Port Connections of 89C52

89C52 has four 8 bit ports, which can be used for I/O purposes. Onboard components are connected through these ports. Port interfaces are summarized on figure A.1. Port 2 is used for addressing a latch, and port 0 is used to send the data to the addressed latch. Port 0 is common to all latches, but only the addressed latches captures the data sent by port 0. Latches of the each palette are connected to one pin of port 2 as the indicated sequence on figure A.1. The outputs of port 0 and port 2 form the data bus. The servos are connected and controlled through first three bits of the port 1. Remaining of p1 is used for channel selection for the analog switch where selection among 32 channels is possible. First two bits of port 3 is reserved for serial communications and interfaced to the RF modem. The preceding three bits are used to control the operation the ADC (analog digital converter).

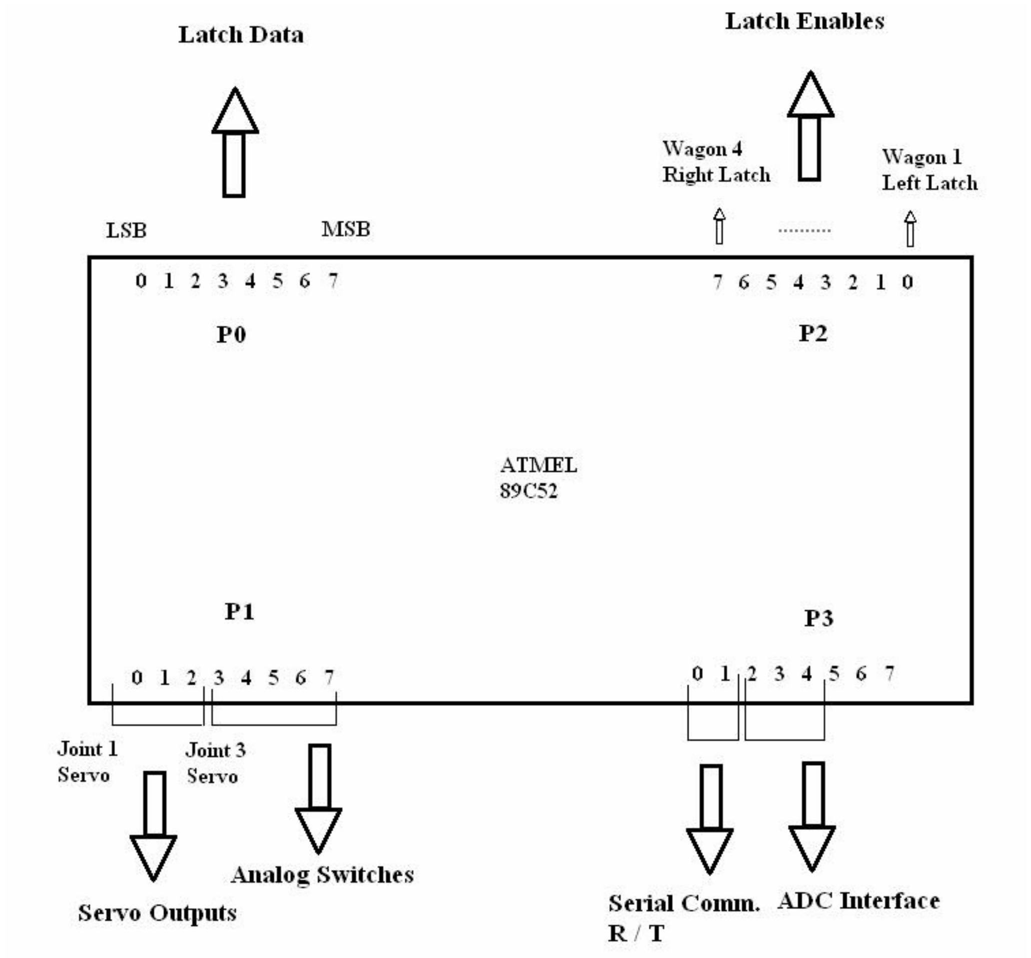


Figure A.1. Port interfaces of 89C52

B. NOISE REDUCTION

B.1. DC Motor with Brushes

Dc motors with brushes usually consist of three groups of coils inside, oriented with 120 degrees. Brushes carry the dc current to one of the three coils which is perpendicular o the magnetic field direction. Coils are coupled to the armature of the dc motor and they rotate as the armature rotates. While operation of a dc motor, bushes make contact with the corresponding coil. Coil characteristics does not permit the current passing through them to be zero immediately, so sparking occurs on the brushes as the brushes loose contact from a particular coil. These continuous sparking broadcasts radio frequency noise and also ripples the supply lines of the motor.

When interfacing a dc motor to sensitive electronic equipment like a microcontroller, special care must be taken. The robot built in this thesis has a microcontroller and rf communications making the controller structure very vulnerable to dc motors.

RF emissions from a dc motor can be reduced by shunting high frequencies on the dc motor itself before they can be emitted. In this thesis shunting is conducted by use of three 100nF capacitors as shown in the figure B.1. One capacitor is connected between the terminals of the dc motor. Other two capacitors are soldered between the terminals and the housing of the dc motor.

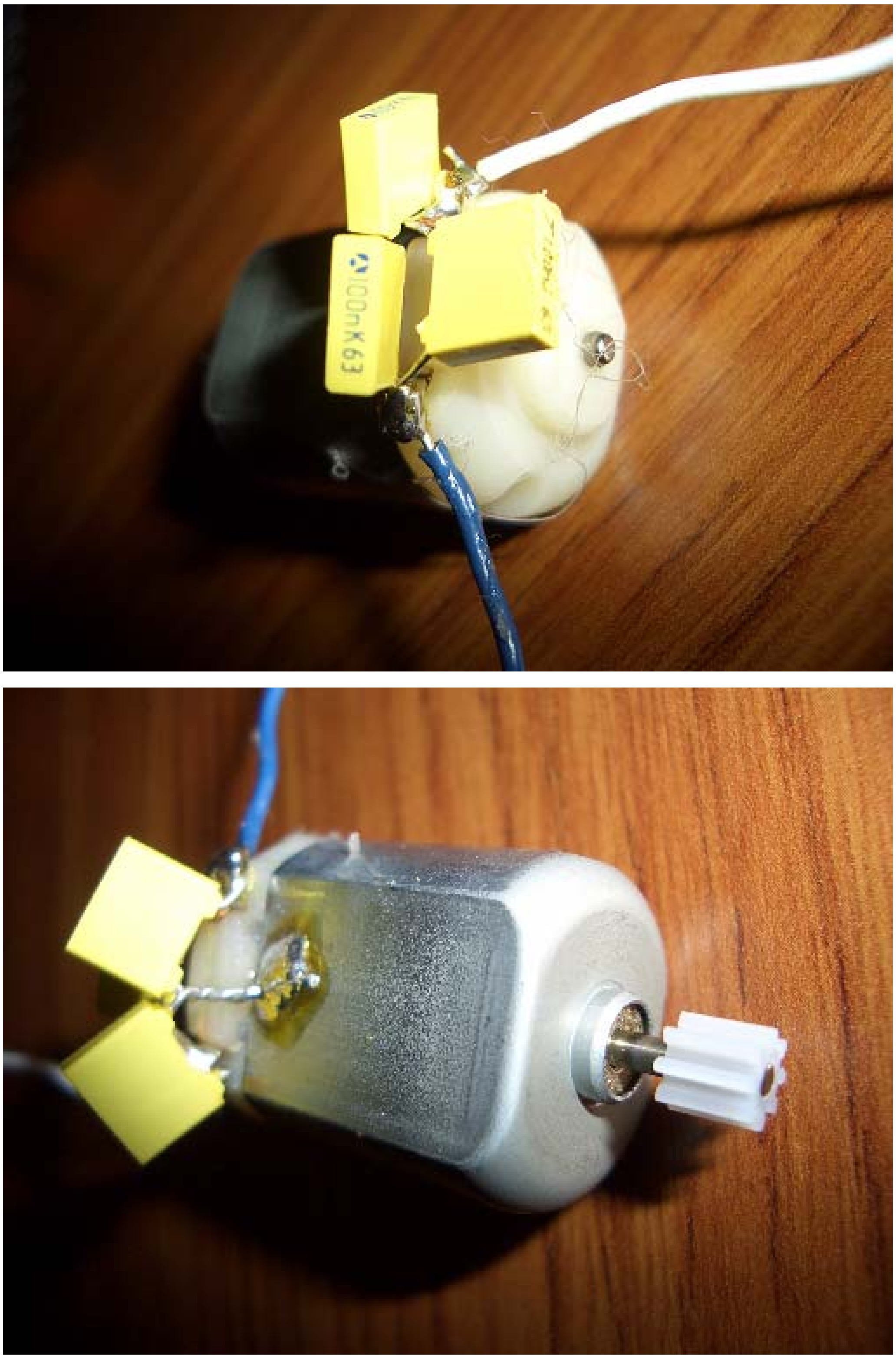


Figure B.1. Shunting of a dc motor. Three 100nF capacitors used. One capacitor is attached between the terminals of the dc motor. Other two capacitors are soldered between the housing and each terminal.

Shunting of rf emissions is not sufficient since a dc motor also ripples the power supply lines. The best thing is to use separate power sources for the microcontroller architecture and the dc motors. But sharing the same power ground still allows the ripples to effect microcontrollers and connected electronical devices. The power source of the dc motors must be separated without having any common power or signal ground.

Dc motors of the robot built for this thesis have isolated power source from the rest of the robot. The isolation is accomplished by use of optocouplers. (figure B.2.)

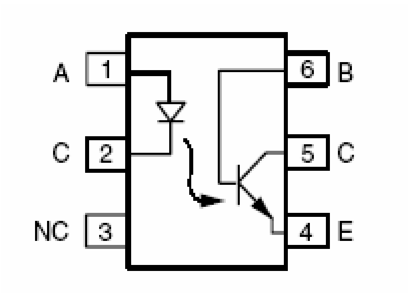


Figure B.2. An optocoupler with phototransistor.

Optocoupler is a device which encapsulates a phototransistor (or sometimes a photo resistor) and a light emitter diode. The light emitter diode converts the electrical signals to light, while the phototransistor converts the light back to electrical signal. The transformation of the signal allows interface between electrical circuits without any electrical connections.

The circuitry shown on figure B.3 is used for each dc motor in the robot. The used optocoupler is “4N25”. The resistor values are selected to be 350, 250, 250, 250 ohms (R1, R2, R3, and R4) respectively to have an output between plus and minus five volts. Refer to the datasheet of “4N25” for more details.

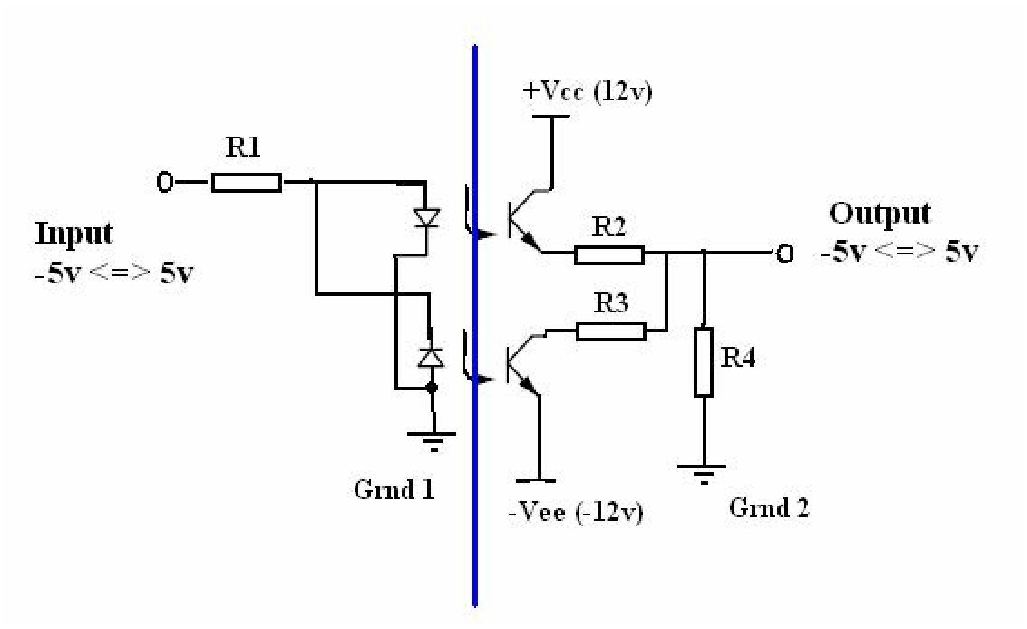


Figure B.3. Optocoupler circuitry for one DC motor. Isolation boarder is indicated by blue line.

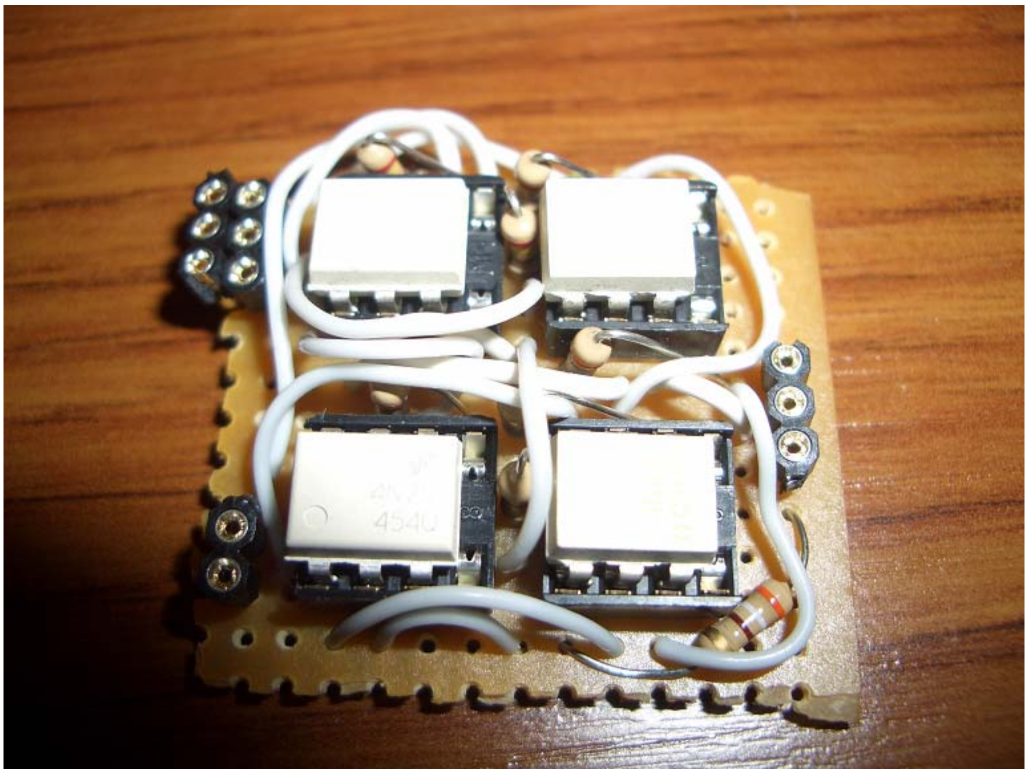


Figure B.4. Four optocouplers interfacing two dc motors on a tank segment.

C. PARTS LIST

RF Modem Long Range (500m) 433MHz - Includes Antenna and Interface Cable

9600 baud half duplex serial connection.

Price : 40 $ each (Must be ordered in pairs)

http://www.sparkfun.com/commerce/product_info.php?products_id=155

Infrared Proximity Sensor - Sharp GP2Y0A21YK

Price : 11 $ each

http://www.sparkfun.com/commerce/product_info.php?products_id=242

Tank Treads

Price: $6.95

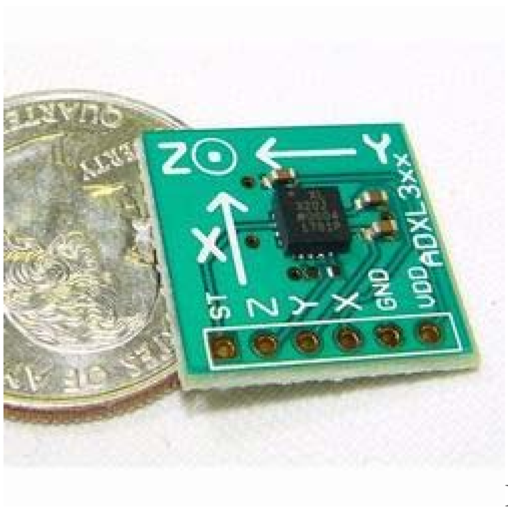
http://www.sparkfun.com/commerce/product_info.php?products_id=321

Dual Motor GearBox (Two dc motors included)

Price: $9.95

http://www.sparkfun.com/commerce/product_info.php?products_id=319

Two Axes Planar Accelerometer - ADXL322 +/-2g



Price: $24.95

http://www.sparkfun.com/commerce/product_info.php?products_id=849