Robust Walking of a Quadruped Robot

WONG CHERN YUEN ANTHONY

(B.Eng(Hons.), UNSW)

A THESIS SUBMITTED FOR

THE DEGREE MASTER OF ENGINEERING

DEPARTMENT OF MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2003

Acknowledgment

Words cannot describe my heart felt thanks to everybody for being so supportive and understanding during this project. I could not have done it without them.

I would like to thank firstly, my supervisor, A/Prof Marcelo H. Ang Jr., for his guidance and support with the project. It has been truly been enjoyable and a great learning experience working under him.

I also would like to thank the staff in the Control and Mechatronics Laboratory - Mrs Ooi, Miss Tshin, Mdm Liaw, Ms Hamida, Mr Zhang, and Mr Yee for their willingness to help me technically and logistically. I will truly cherish the friendship we have built up in these two years.

Thanks also to my fellow students in the laboratory, for the good times we had in the laboratory, and the many simulating discussions we had about our work and interest. It was truly a wonderful time working with these “kids” and their “adult toys.”

I would like to thank my youth pastor and members of my youth group, for their prayers and concern, for their friendship and always being there to spice up my life in those tired and gloomy days.

Thanks to my gang for putting up with my busy schedule and always being so concerned about my work; for listening to what I have to say about my project although they don’t understand much of it. They are truly an understanding bunch.

I am forever grateful to my loving family – my parents, my aunts and my sister and her boyfriend, for their support in my decision to take on this project, and understanding my love for this field. Their love and concern made it possible for me to concentrate on this project and not have to worry about other things in life.

I am indebted to my wife, Wai Fong for her love and patience as she stood by me as I completed this project. She was there for me always, a source of light in my darkest hour. I am grateful to her parents for being supportive throughout these two years.

Lastly, but most importantly, I would like to thank God for listening to me in my prayers, for guiding me with His Word, for getting me out of my troubled times, and placing me with the people I have mentioned. I definitely could not have done it without him. Praise to God.

Table of Contents

List of Figures

List of Tables

SUMMARY

A modular controller was developed to evaluate performance of robust locomotion control algorithms. An NUS developed quadruped was used as the testbed for this project. A low level controller was developed as two modules, the hardware interface and the fuzzy controller, which is attachable to any high level controller. Two high level controllers were developed - the behaviour-based and Central Pattern Generator (CPG) controllers. The behaviour-based controller is a modular controller based on interaction between various modules. *Basic* modules were developed to provide *basic* motion. Other modules were implemented to improve the controller's robustness. The CPG controller is based on the animal spinal system's coordination of locomotion. Pribe, Grossberg and Cohen's Oscillator Model for CPG were implemented. The algorithms used in the behaviour-based controller were integrated to this controller to provide gait stability. Both static and dynamic gait patterns were simulated with online gait change.

Chapter 1

INTRODUCTION

1.1 Legged Locomotion

Legged robots have been around for quite a while. They are considered mobile robots and are used in the area where wheeled robots find difficulty in moving. Many researchers marvel at how animals and insects move and try to model their actions and style of movements. What looks easy at first glance is not always as easy as it seems. The way that a cockroach moves differs from the way that a preying mantis moves. The speed at which they move is also different. The idea of studying these creatures is really to achieve what they have achieved in terms of efficient walking.

1.2 Diversity in Nature

God seems to have a sense of humour when he created all creatures big and small. All creatures have different forms and mechanisms that aid it in its environment. All animals and insects have different type of sensors that work differently to aid them in their task. The creature's position in the food chain determines the way that it moves. Therefore, the walking mechanism for animals is different; all with different leg structures with different lengths, placed differently, serving various purposes. Nature has definitely provided us with a multitude of solutions to problems. It is up to us to decide which solution is the most viable for robot design. Spiders, cockroaches and various insects have been models used for legged robot design. Extracting information from the observation of these insects has given us much insight on gait and structure. This knowledge is tested on machines to correlate the findings. More can be gained as the research in this field matures. As more is understood about legged locomotion, autonomous robots can be developed that will take over human tasks. This is the goal of legged robot research.

1.3 Insights in Legged Robotics

The study of animals have given us insight on how walking is achieved. At this point in time, our understanding of biological walking is still shallow. There are not many legged robots that can achieve autonomous locomotion for long periods of time. Recovery is a major problem for legged robots. If the robot becomes unstable, there must be a way to recover, like a cockroach's ability to flip itself over once it overturns. A lack of this capability reduces autonomy of robots as instability is unavoidable in the real world.

1.4 The NUS Quadruped

A quadruped was built in 1998 by Cheong Choon Ghee [Cheong 1998]. Different controllers were built for this robot to develop a working walking gait. Over the years, sensors were added to the robot to improve the stability and flexibility of the controller [Atienza 2001]. All work done in this project is based on this robot. A detailed description of the quadruped is presented in Chapter 3.

1.5 Goals

Building a robust robotic system is developing a system that is able to operate withstanding conditions or changes in the environment. This can be built into various aspects of the robot system, both in terms of hardware and software. This thesis focuses on incorporating robust algorithms into a legged robot controller. Various biologically inspired controllers are available but two intriguing controllers looked at in this project are the Behaviour-based controller and the Central Pattern Generator (CPG). The first controller is a high level controller that mimics behaviour interactions whereas the second controller is a low level controller that looks at the locomotion generating mechanism in the spinal system of animals.

Software design was also taken into consideration in this thesis. It is important to build a software structure that can be further developed. Important targets for the software design are:

1) modularity,
2) structure that help further development,
3) real-time for control purposes, and
4) Windows interface for user friendliness.

Taking into consideration the various requirements for the controller, the Windows operating system was chosen as the platform for the development. Since real-time is a requirement for this controller, Venturcom's RTX [RTX User], a real-time extension of Windows NT/2000, was used. Development of the controller was done in

Microsoft's Visual C++ as RTX provides ample support for this development platform.

1.6 Outline of Thesis

Chapter Two gives an overview of legged locomotion research. Research in this area has provided vast information about legged locomotion, controller design and legged robot design. An overview of the NUS quadruped, the project goals and the low level controller are presented in Chapter Three. Chapter Four explains the structure of the controller program developed. Chapter Five and Six explain the implementation of two controllers developed for this robot, the behaviour-based controller and the CPG respectively. A comparison of these two controllers is done in Chapter Seven and finally, Chapter Eight provides conclusions.

Chapter 2

LEGGED ROBOTS

2.1 Introduction

Legged robotic is a form of mobile robotics that has its application in specific areas. It is important to understand the various hardware design and control methods before we dwell into developing the robust controller. This chapter gives an overview of the various ways of hardware and software design considerations which provides the groundwork for the development of this thesis.

2.2 Motivation

Today, most mobile robots are either wheeled or tracked vehicles. This is because they have the advantage of speed and stability in a designed and controlled environment. Since most factory floors have flat and smooth surfaces, it is easier to manoeuvre such types of robots. Therefore, wheeled or tracked robots have the advantage over other forms of mobile robots in this aspect.

It has been estimated that more than half of the Earth is inaccessible to wheeled devices. The area of legged robotics research focuses on this problem. The drive towards legged robotics research is that of taking on tasks that wheeled robots cannot execute due to its inability to overcome irregular terrain. Research has shown that legged robots are not only effective when moving but can also isolate their bodies from terrain irregularities, avoid undesirable footholds, regulate stability and achieve energy efficiency.

In the industry and applications, the potential use of legged robots has extended to areas where there is a need for walking on the non-flat surfaces, climbing of walls and ceilings, and to perform ground-to-wall transition movements. The applications of these uses are:

1) wall climbing in ship hulls, nuclear plants [Briones 1994] and oil plants;
2) interplanetary exploration [Krotkov 1992];
3) minefield clearance [Nomani 2002];
4) bomb disposal;
5) surveillance of hazardous or dangerous environment such as volcano craters [Bares 1999] and chemical plants, and
6) climbing/crawling inside pipes.

2.3 Advantage over Wheeled Robots

The following is a list of the advantages of legged robots over wheeled robots.

- Walking removes the energy wasted due to the slipping of the wheels. It also removes slip error during navigation.
- Since there is less contact with the ground, there is less harm done to the environment.
- The posture of the robot is not dependent on the contour of the ground it stands on. By adjusting the length of the legs, the body is able to maintain its posture

whatever the contour of the terrain it is on. This allows the robot to be able to climb steps and slopes easily.

- As legged robots can handle frontal, backward and sideward motion, it can move without having to turn the body of the robot.
- The problem of overcoming obstacles is simplified as the robot is able to step over and step onto obstacles.

2.4 Classification of Legged Robot Research

Legged robot research can be classified into two areas. Firstly, biologically inspired robots where legged robots are built to study the gait ([Cruse 1991], [Beer 1998]), nervous system and the different areas of insects and animals. Secondly, legged robots are used to deal with locomotion where wheel robots cannot handle. These two areas are not researched as separate entities, as working with biological systems allow understanding of how creatures and insects handle the different terrain. This acquired knowledge gives engineers vital data that they can employ to optimize their robot designs.

2.4.1 Engineered Robots

The first form of legged robotic research aims towards problem solving. The engineering techniques of robot building and control are explored. These robots are usually simple in design to study the different aspects of locomotion from an engineer’s point of view. For example, a wall climbing robot [Nagakubo 1994] climbs a vertical surface and hence a low center of gravity for stability and suction cups on its feet for climbing.

2.4.2 Biologically Inspired Robots

Every biological system has evolved over time and works for the task it performs in its environment. Every creature on this earth is equipped with the skills and anatomy to survive in its environment. The first approach takes the understanding of joint movement, muscle activity and the neurological system to aid the development of a biologically efficient robotic system. The main issue in this approach is the degree of realism necessary for the robot to perform its task. The second approach takes the reverse approach where robots are used to model animal. Simulations cannot fully characterize the dynamic relationships as with an actual robot. Thus robots give a clearer picture of the physics involved in legged locomotion. However, it is difficult to match the physical properties of the robot with the animal reducing the reliability of the data obtained from observing the robot.

In terms of control architecture, nature has provided three design procedures that can be used to develop legged locomotion controllers, namely that of evolution, development and learning. The first two determine the role of the controller and the last fine-tunes the controller for adaptation to the environment. The study in the area of evolution has led to the development of various evolutionary control architectures (using neural networks), or different forms of genetic algorithms/programming, or a combination of the two [Kodjabachian 1995]. Neuroscientists and biologists use these robots as an avenue to verify their theories benefiting both the neuroscientist and the engineer.

Therefore, the use of robots to study animal locomotion is gaining popularity. They include the study of cockroaches [Nelson 1997], stick insects [Cruse 1991] and other forms of insects. Not only does this help in developing better robotic systems, but also a further understanding of how animal locomotion is produced.

2.5 Legged Robot Control

Designing and building a legged robot controller is a daunting task. There are many methods for legged robot control but all can be categorized into either of two methods: planner-based control and reactive control.

2.5.1 Planner-based approach

The planner-based approach plans the action of the robot according to a centralised world model [Mat'aric 1992]. Sensory information verifies the known world model, which in turn generates a sequence of actions that the robot performs. This method requires the model to be accurate, i.e. the model of the environment has to be developed for the robot to be able to locate itself. This means that the robot's movement is analytically modelled, accurately planned and strictly controlled.

2.5.2 Reactive approach

The other extreme of control is the reactive approach. This form of control is implemented as a collection of condition-action pairs [Mat'aric 1992]. The conditions are sensor dependent - once a sensor is triggered, the corresponding action pair will be activated. It requires knowledge of the control environment to design the condition-action pairs.

This model has minimal internal states and works on looking up the correct action based on the appropriate set of sensor readings. This form of control has high efficiency due to minimal computation. The controller is thus able to react at the pace of the environment. However, there are no predictive components and therefore it lacks in the area of run-time flexibility since is has little representational power.

Biological systems provide evidence for these sense and act reflexes that are decentralised from the supervisory controller (the brain). This method of control allows the movement of the robot to emerge from a set of specified rules, or from interaction between different control processes.

2.5.3 A Hybrid Approach – Reactive and Planner-based

It has been found that planning alone without adaptation is not sufficient for robots to walk robustly on natural terrain. Unpredictable events occur commonly in the natural environment. These include events such as obstacles, unexpected holes, and slipping of the foot. These situations have to be handled for robust walking.

Therefore, the integration of the above mentioned two approaches has recently gained wide acceptance in controller development of some legged robots [Mat'aric 1997]. This is an approach that combines deliberative reasoning and behaviour type reaction. This means that predictable events are handled by planned actions whereas the reactive behaviour handles unpredictable events. This truly represents the construction of the control present in animals, thus providing an approach that would capture the essence of animal legged locomotion.

In designing a controller, the higher level planner module can provide guidance of the robot, and optimization of performance, whereas the lower level reactive approach handles the unforeseen circumstances that need to be addressed immediately.

2.6 The Science of Walking

Creatures, man and animal alike have been walking the earth since the creation of land. Different animals possess different number of legs and different types of limbs. Different creatures have different forms of locomotion to suit its natural habitat's terrain. Locomotion, although an age-old art, still has a sense of mystery, as human has just started to comprehend how it works.

Before looking into the different aspects of locomotion, this section shows us how different machines and animals are. This is crucial to understanding the limitations of machines and ways to improve the design of the robot.

- Engineering materials are stronger than biological ones, but are rarely strong in both compression and tension.
- Biological processes can produce complex shapes and composite material whereas in engineering all parts are kept as simple as possible due to availability and ease of manufacturing.
- Muscles and load sensing to remove stress from the bone but these are not present in robots.
- Robots can stop consuming energy upon remaining in a static position and do not experience fatigue.
- Robots are heavier but are able to pull or push better than muscles. Linear and angular motion and actuation is more achievable with motors.
- Position and velocity control of rapid accelerated coupled masses for robots are not as developed.

2.6.1 Walking Considerations

In this section we consider the important factors in walking. Walking involves coordination of limbs, joints and the body. The goals of walking can be divided into two main categories – moving to the desired foot position, and maintaining foot stability. These two goals can be further split into the following sub-categories.

1) Foot placement – Foot placement has to do with the location and maintaining stability. The location of the foot depends upon the goal of the robot. Stability requires that the robot ensure that the foot is securely placed on the ground to prevent slippage before moving the other legs.
2) Walking method – The locomotion of a mechanism by support and propulsion that comes from the coordinated motions of its joints and limbs.
3) Weight support – The robot must be able to support its own weight while walking, without which the robot will not be able to carry out locomotion.
4) Avoiding obstacles – There are two ways for obstacle avoidance in walking. First the option of walking over the obstacle, and secondly seeking an alternative path. Obstacle Avoidance for dynamic walking is difficult as the recovery of the robot depends of how fast and how timely the system can react.
5) Height adjustment – Legged robots can adjust their heights and therefore a mechanism is required to ensure that there is always posture stability of the system.

These points are used as guidelines to the design of a robust controller.

2.6.2 Static and Dynamic Legged Locomotion

There are two types of locomotion – static walking and dynamic walking. Static walking maintains static equilibrium throughout robot locomotion. This is achieved by ensuring the robot is supported by at least three points of contact at all times. Therefore, robots that perform static walking must have four legs or more.

Dynamic walking is legged locomotion where instability is allowed – not requiring at least three legs are on the ground. The robot can continue moving as long as the robot does not fall over. There can be periods where all the legs are not on the ground. Robots with four legs or less usually perform dynamic walking. Robot with four legs can perform dynamic walking like the gait during a horse's gallop. Biped can perform gaits like running. Stability in dynamic walking is the ability to obtain a stable cycle instead of a stable position.

2.6.3 Free Gaits versus Fixed Gaits

Free gait provides the freedom for the legs to move freely at any point in time to provide the support and propulsion for the legs. A fixed gait on the other hand, is a periodic gait of which the action taken by each leg is governed by a sequence of events that cannot be altered during a move. Therefore fixed gait controllers do not require online planning of movement. Fixed periodic gaits are stable and fast on smooth terrain, but not as efficient on uneven terrain. Free gaits are more suitable for uneven terrain. A method for developing free gait is needed. Free gaits are generated based on the situation, taking into account the manoeuvrability and stability of the gait. A free gait is an unplanned gait where there is no rule set for the order of leg movement. Free gaits are aperiodic, irregular and not often seen in the real world. Its main concern is stability and is usually applied when moving in dense obstacle fields. Gaits that fall between these two extremes can make use of the advantages of the two methods – the computational efficiency of the fixed gait and the freedom to act of the free gait.

Planning a gait include optimisation of movement, speed control and finding the most stable gait among the gaits available for the number of legs the robot has. A good gait generator has the ability to switch between gaits depending on the terrain it is on, as a measure of the stability compromised due to a change in speed. Since a periodic gait is more stable and efficient gait, the robot should always degenerate to a periodic gait once it traversing on flat terrain. The gait must be able to adapt to terrain conditions and switch to an aperiodic free gait when the need requires.

2.7 Robust Structural Design

The aim of this section is to show the various steps that can be taken to ensure robustness of the robot in terms of the design of the robot. Simplicity is the first step towards building a robust robot as simple design means less chance for error. Only the required actuation and sensing components are added.

The second factor to consider is autonomy. Design of the robot, both hardware and software has to be considered. The battery lifespan must be sufficient for the robot to complete its task. The robot must be lightweight to reduce the amount of energy required to move the robot.

Most animals have evolved to achieve optimal locomotion in its own terrain. The only problem is that nature does not always produce the best design for robotic locomotion. Most legged robots mimic animals in structure. A good design is important when it comes to any robot. A good design would mean,

1) simple control,
2) reduced dynamic effects,
3) increased stability,
4) good strength to weight ratio,
5) self contained power,
6) self contained computational power,
7) ease of movement, and
8) low cost.

Although most designs of robots mimic biological systems, mechanical design improvements are often added to increase the efficiency of the robot. This is task dependent, as robots built for different purposes require different features. Cheong [Cheong 1998] did a survey of the different legged robot designs. The survey of the available robots has shown that there are many methods used for the different components of a legged robot.

2.7.1 Leg Structure

Five types of leg structures are used widely:

1) Pantograph based,
2) Translation style,
3) One-link leg with rotational joints,
4) Three-link insect type leg, and
5) Two-link leg.

The type of movement will dictate the leg design used. A legged robot that is built for wall climbing differs from a robot built to run. Different factors have to be taken into consideration. Certain questions have to be answered, such as “What is the use of the knee in legged locomotion?”, “Should the knee joint bend inwards like birds or outwards like animals?”, “Are we dealing with static or dynamic locomotion?”, etc.. Answers to these questions affect the decisions made in the selection of leg design.

Planer robots are easy to control as there are fewer complications involved. These robots are used to study the movement in a certain plane, i.e., two-dimensional. There must be at least three degrees of freedom per leg for the robot to be suitable for various terrains to maintain its posture during locomotion as the ground is a three dimensional entity.

2.7.2 Leg Configurations

The number of legs for statically stable walking robots must not be less than four. Many quadrupeds have been built, as they require the minimum number of leg required for static walking. They can also perform dynamic gaits like galloping and trotting. This not only simplifies leg design but also reduces the weight of the robot.

The speed of the robot reduces as more legs are added to the robot. Additional legs are added for redundancy in case there are damaged ones. But more joints have to be controlled if more legs are added. Sensing of the different feet increases in complexity as well.

2.7.3 Types of Joint Movement

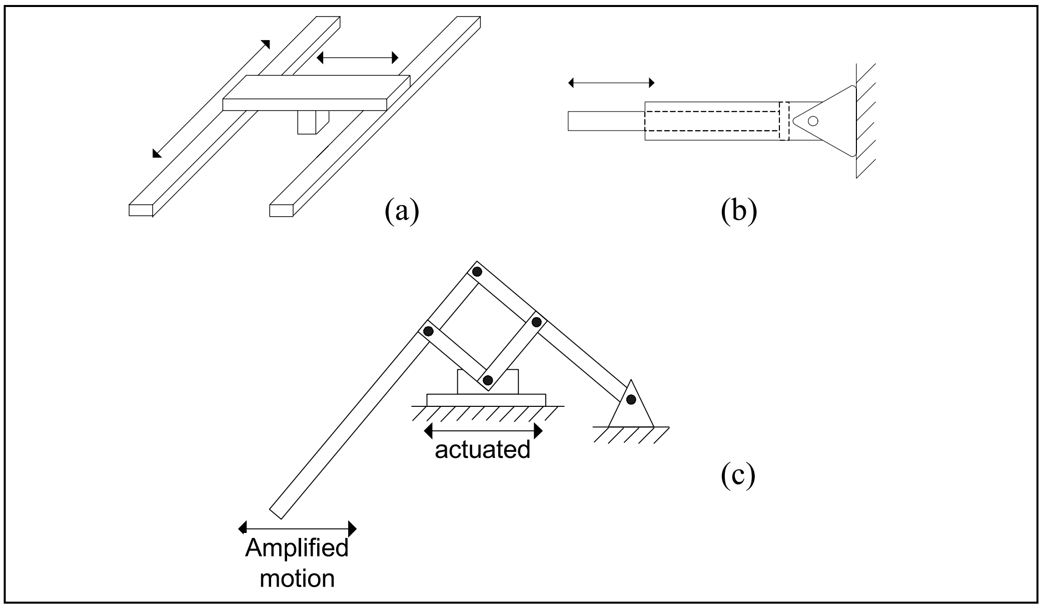


Figure 2.1 – (a) Cartesian implementation is an example of a translational joint. (b) Pneumatic actuators are examples of telescoping joint. (c) Single axis pantograph joint example.

There are many forms of actuation that are used for joint motion. Linear actuation is easy to implement as control of translational joints (Figure 2.1a) can be solved using easy kinematics. Another type of actuators that are linear in motion are the rectilinear actuators, such as telescoping joints (Figure 2.1b) and sliding joints. A method of linear actuator with amplification is implementing actuation with the pantograph joint (Figure 2.1c). The control for a pantograph leg is simple as a distance covered by a motor in a certain direction causes a proportional movement of the foot in that direction.

Rotary actuation is the other more commonly used method. This method likes animal joint movement and, therefore, is used to emulate animal joint motion.

2.7.4 Actuation Method

Electricity (as in electric motors), hydraulics and pneumatic are the common types of actuation medium used for robotics.

Electric motors can be used to actuate the joint in a linear or rotational motion. This form of power is clean but the problem is that the motors must produce high torque. This problem can be solved by using gear reduction which slows down the motion but increases the torque produced. But there are trade-offs when gear reduction is employed. It increases the apparent output inertia, which not only increases the impedance of the output but also greatly reduces the gear's tolerance to shock.

Gear friction in the motors is highly non-linear and time varying. This reduces the fidelity of the forces transmitted to the output of the actuator. The Massachusetts Institute of Technology (MIT) developed an actuator to alleviate the problem [Pratt 1995]. The solution is to place a spring between the output of gear reduction and the load. The force output of the motor can be controlled with this added elasticity and an added feedback loop. This elasticity and feedback reduces the rigidity of the motion thus gives a "softness" in actuation like human touch.

Hydraulics allows for robust velocity control with a very high specific force/torque and force/torque density, thereby not requiring gears. Hydraulic systems use fluid for transmission of power; hence they have high stiffness (due to the incompressibility of the fluid) and do not require brakes for holding position. The problem with hydraulics is that it is messy due to fluid leakage and a pump has to be connected to the robot. The pump is noisy and power inefficient.

Pneumatics is cheap and easy to install. One of the problems of pneumatics is that air is compressible and cannot be treated as rigid. A pump is required for the compressed air, which might be a problem in terms of mobility and noise.

Actuators can be placed either at the joints where the actuation is taking place or placed away from the joint where the joints are actuated by a system of pulleys and wires. Most legged robots have actuators at their joint. An example of a robot with the actuator away from the joint is the Spring Flamingo built by Jerry Pratt of MIT [Pratt 1998]. The actuators are located at the upper body of the robot with power transmitted to the joint via cable wires. This design reduces the weight at the legs and therefore reduces the inertia at the legs.

2.7.5 Leg Disposition

Mammals walk differently from reptiles. Mammals have legs under their body whereas reptiles have legs on their sides. The advantages and disadvantages are discussed in [Cheong 1998]. The NUS quadruped has a wide base like a reptile and, therefore, more stable. The CG is low as well, which makes it easy for the body to be lowered to ground.

2.7.6 Foot Design

Foot design is important as the stability of the robot on uneven terrain depends on the foot of the robot. The foot of the robot must be able to have a grip on the ground. The traction on the foot is important to locomotion as slip is reduced.

The foot design must provide for adequate traction either by friction or digging into the ground. The foot must provide a degree of shock absorption like the sole of human feet. The shape of the foot is determined by the task of the robot. If the robot needs to climb walls or hang from bars, sucker pads and claws can be added to the robot.

Selecting the shape of the base area of the robot determines how load distribution is handled. The contact surface determines the stability of the robot. The feet of the NUS quadruped were designed with these points in mind.

2.7.7 Joint Design

There are two types of joints that are used in legged robots, either an active joint or a passive joint. The passive joint has a holding brake whereas the active joint uses an actuator. Animals have passive joints with active control using muscles and tendons.

2.7.8 Energy Losses

Though legged locomotion is efficient as compared to other forms of locomotion, there are still energy losses due to the system inefficiencies. As mobile robots require a self-contained power source, power consumption is an important issue. These are some areas where energy losses are evident.

1) Power wasted in supporting the weight of the robot. This is important for muscle like actuators that dissipate energy when at rest.
2) Too many actuators.
3) Oscillating limbs (e.g., waste of energy when the center of gravity moves up and down during locomotion).

2.7.9 Shock Absorption

A degree of compliance is required for a smooth ride. This elevates the shock that is subjected to the actuators as the foot lands on the foot.

2.8 Incorporating Intelligence

Usually a legged robot is designed to perform tasks such as data collection or moving equipment. The robot must first be able to perform robust walking before it is able to perform the above tasks. The inability of the robot to perform stable legged locomotion renders it useless for the tasks.

Intelligence gives a robot the ability to act and react. Intelligence in legged robots is the ability to avoid obstacles, to understand the situation and plan ahead to recover. These various algorithms make up the brain of the robotic system.

Intelligence is feedback to the system based on sensory information and the robot state information. It can either affect the system by modifying the current output or affecting the system with a new set of output. Over the years many algorithms have been developed to incorporate intelligence into robot controllers ([Brooks 1989] , [Krotkov 1992], [Hillel 1992]).

Integration of the different algorithms is possible but the effort of implementing such systems might not be practical, e.g. high computing power requirement [Ijspeert 1999], extensive training required, etc.. As numerous algorithms have been developed in the field of legged robotics, the aim of this work is to integrate a set of suitable algorithms to develop the robot controller.

2.9 Summary

An understanding of the different aspects in this field is required before the controller is developed. This chapter first introduces legged robotics and its uses. Other issues that are dealt with include the art of walking, structural design, and gait control. This knowledge will be used in the next few chapters in the development of the controller.

Chapter 3

THE NUS QUADRUPED AND CURRENT GOALS

3.1 Introduction

The first step taken in developing the controller is to understand the existing system and set the goals to be achieved. Hence in this chapter, the NUS quadruped is introduced. The motion kinematics and hardware is discussed in the first half of this chapter. Whereas, the goals and guidelines for developing the robust controller are developed in the second half of this chapter.

3.2 Past Work

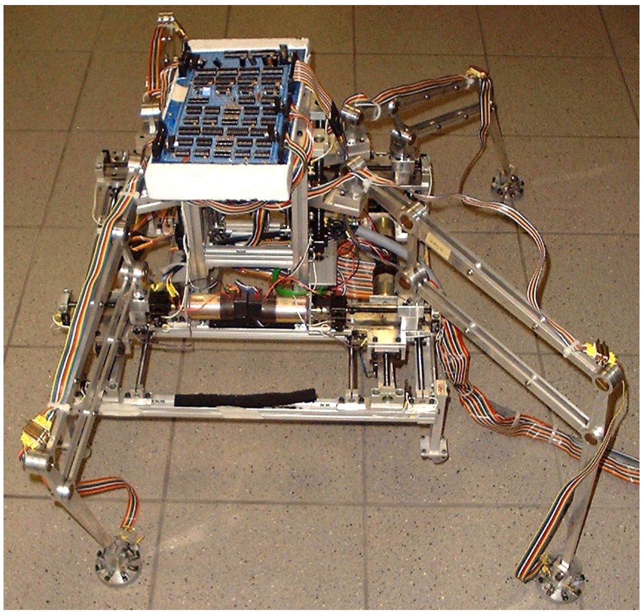


Figure 3.1 - The NUS Quadruped

The NUS Quadruped was designed and built by Cheong Choon Ghee [Cheong 1998]. It is a quadruped based on a pantograph leg design driven by linear actuation in each axes (Figure 3.1). The controller of the robot is a PC. The power supply and motor amplifiers of the robot are not onboard.

Since some equipment are not onboard, the robot has a four-metre long "umbilical" cable that links all sensors and encoders signals to the computer and power to the motors and sensors. The PC has Digital Input/Output cards, encoder counter cards, motor drives and Digital/Analog Cards that communicate with the robot. The block diagram of the robot system is shown in Figure 3.2. The top part of the figure shows the equipment that is not onboard.

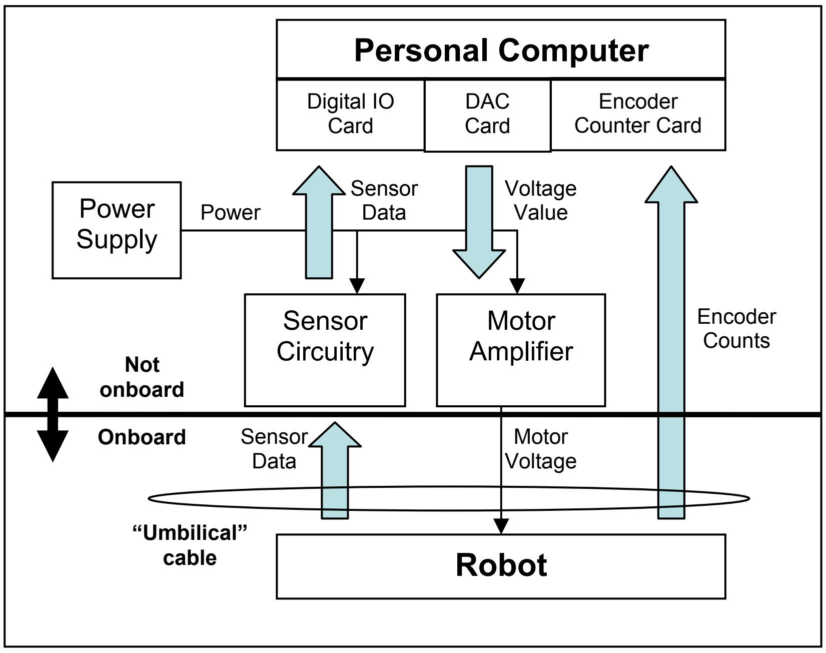


Figure 3.2 – Block Diagram of Robot Setup

Another graduate student, Rowel Atienza, continued with the quadruped, where he built a high level controller for the robot to walk on both even and uneven terrain

[Atienza 2001]. The controller was also able to keep the stable posture of the body. It was based on Brooks' Subsumption architecture [Brooks 1986], a behaviour-based approach. A group of sensors were added to the robot as behaviour-based controllers depend on sensory feedback to work efficiently. Proximity sensors and force sensors were added to all the feet of the robot to facilitate his work. A circuit was designed to transfer sensor data from the robot to the computer via the long "umbilical" cable. A low-level fuzzy controller was developed to move the different joints of the robot. The robot was able to perform simple walking, climbing steps and clearing obstacles.

Cheong's work [Cheong 1998] includes the design considerations of building the NUS quadruped. Whereas Atienza's thesis [Atienza 1998] contains information about the sensors added to the robot. The circuit diagram and robot design CAD drawings can be found in both Atienza's [Atienza 1998] and Cheong's [Cheong 1998] theses.

3.2.1 Hardware Design

This is a brief description of the hardware used and the design considerations of the robot to gain a better insight of both the structural and actuation design of the NUS quadruped.

3.2.1.1 Leg and Foot Design

Simple Joint Control

No complicated kinematics and inverse kinematics calculations are required as the robot uses the pantograph leg for actuation (Figure 3.3). Every foot is actuated in all three axes with servomotor through a ball screw. The pantograph design leg movement to servo motor movement ratio is four to one.

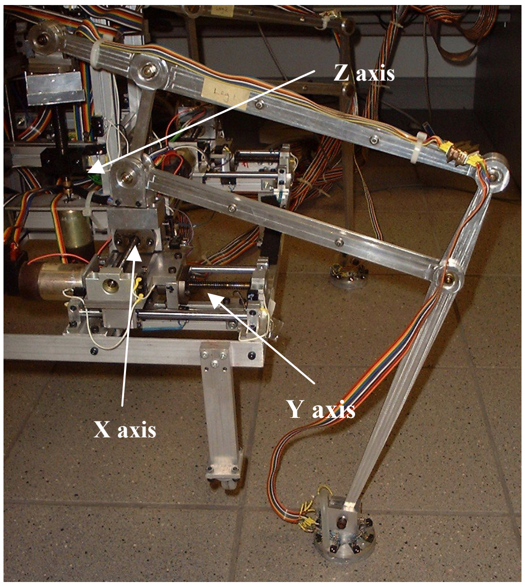


Figure 3.3 – The pantograph leg design

There are a total of twelve motors onboard, three motors for each leg controlling the three different axes. Since the three axes are controlled individually, motion in the X, Y and Z axes are decoupled, which reduces the complexity of control.

Foot Design

The foot has a round base and swivels around one axis with one degree of freedom. The swivel allows the robot to adapt to the terrain it is stepping on. The foot swivels in the x-direction (left-right direction) of the robot (Figure 3.4). The orientation of the foot with respect to the ground does not change when the leg moves.

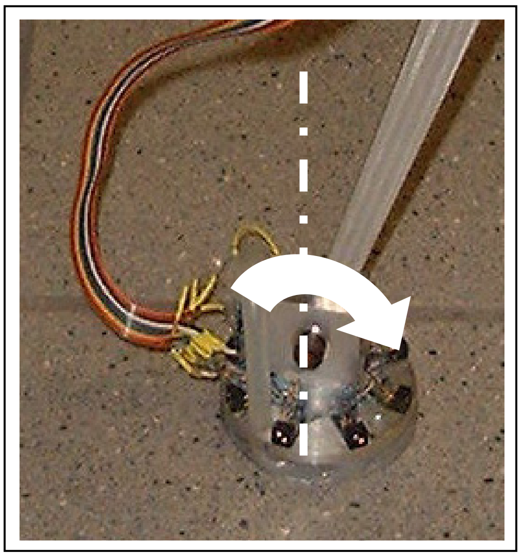


Figure 3.4 –X-direction swivel of the foot

3.2.1.2 Sensors

Humans use their sense of touch and sight to navigate and a balancing mechanism to maintain posture and stability. Robots, like humans require a set of senses to achieve stable walking. Different senses work differently to obtain information about the environment and the current state of the robot. A clearer picture of the environment helps in the analysis of the situation and therefore ensures more deterministic reactions. Sensors are crucial to the adaptability and stability of the robot.

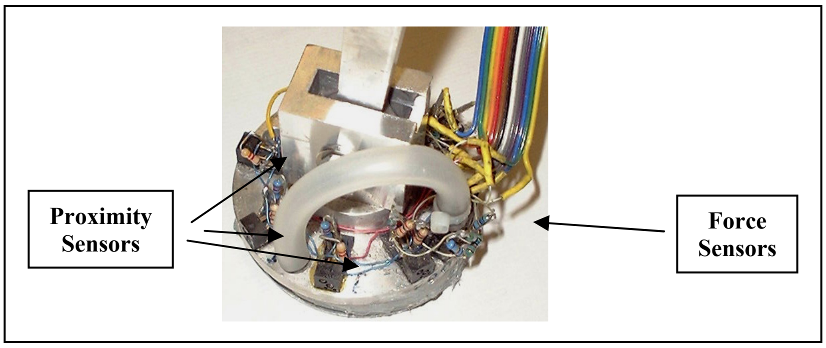


Figure 3.5 – Sensors used on the foot of the robot

There are two types of sensors in use as shown in Figure 3.5. There are eight proximity sensors around the foot and a force sensor. All four feet have the similar configurations.

Proximity Sensors

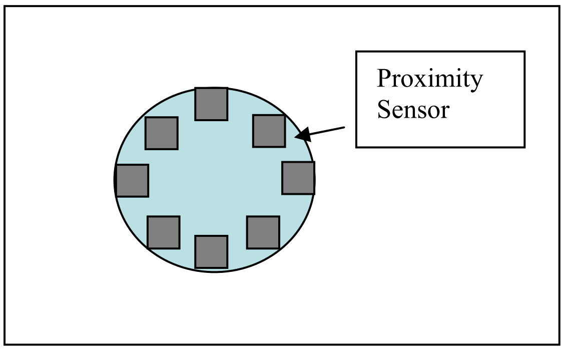


Figure 3.6 – Position of proximity sensor on foot

Each foot of the robot has a set of 8 infrared sensors spread over equal angles around it (Figure 3.6). These proximity sensors are used to detect obstacles around the robot. The direction of the encountered obstacle can be determined by noting the triggered sensors. It informs the robot about objects in the vicinity. It detects objects up to 3 centimetres away.

Force Sensors

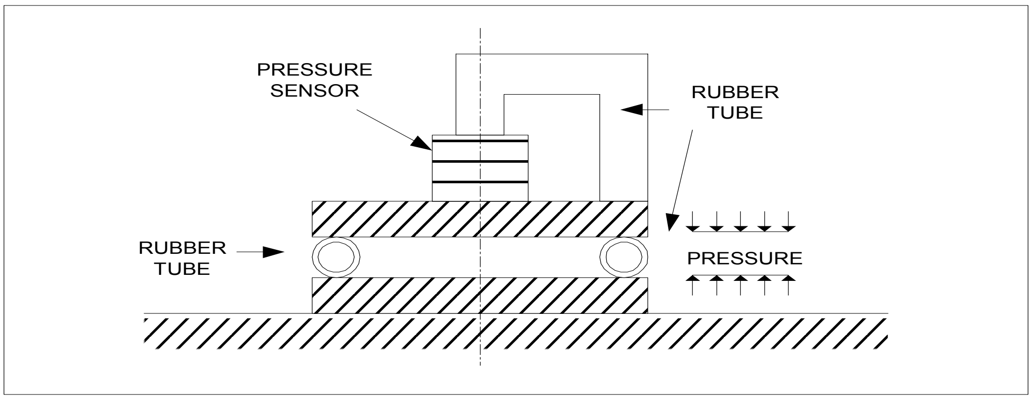


Figure 3.7 – A cross-section of the force sensor

Each foot is equipped with a pressure sensor that is placed at the bottom of the foot [Atienza 1998]. A cross-section of this sensor is shown in Figure 3.7. A sealed rubber tube if filled with water and sealed at both ends with a pressure sensor on one end inside the tube. The pressure acting on the water-filled tube triggers the pressure sensor. The robot will stop moving its foot down once the force generated due to the contact reach a threshold. It serves to maintain the posture of the robot, ensuring that the body of the robot is always parallel to the ground, unaffected by the contours of the terrain. A detailed description of the sensor is found in [Atienza 1998].

3.3 Aspects of a Robust Walking Controller

A good set of guidelines is important in the development of controllers. A set of guidelines is listed in this section for the development of controllers for legged robots.

The main objective of this project is to develop a robust walking robot controller. Robustness in robotics control is the ability to handle or react to unexpected events or disturbances. In the field of robotics, there are two aspects to robustness. The first implies that the robot can cope effectively with unexpected changes in the environment or surroundings and secondly, cope with damages. It is difficult to cover both aspects within the timeframe set for this project. Therefore this project deals with only the first aspect.

Another important purpose of a robot is to take over tasks that humans perform. The robot must be able to perform these tasks autonomously. It not only encompasses ensuring the task is performed properly but it should have the ability to modify the behaviour of the robot if the task is incorrectly performed. Mobile robots require both robustness and autonomy to work as their task requires movement, which involves many uncertainties requiring immediate action.

Robustness is the main focus of this thesis but other goals are included to improve the software and structure of the controller. The other goals include modularity, reactivity, adaptation, and expandability.

3.3.1 Walking Planning

Walking can be achieved by deliberately planning the gait (offline planning), without the ability to change the gait pattern while the gait is being generated. This reduces the flexibility of the controller of the robot but is simple to implement. This kind of walking does not take into consideration the various problems encountered during walking. It ensures stability of the gait but sacrifices flexibility. Different offline gaits have been implemented by studying the gait pattern of animals.

Deliberate planning also prevents the robot from changing its gait when it encounters an obstacle as the gait has been planned and cannot be changed. This is unlike animals where the gait changes according to the terrain and obstacles it comes across. An improvement to this would be to execute the planned gait and modify the existing gait if an unforeseeable event occurs.

A way to avoid planning is to respond to situations reflexively. Therefore, it requires gait control to be online, and real-time planning. Gait planning takes into consideration what the robot senses, and therefore, is able to react to different situations. Speed is compromised, as the controller needs to process sensory data, and create the gait while the robot is moving. However, the robot has the ability to react autonomously to any unforeseen circumstances and to act by foreseeing the consequence of certain actions.

The second part of walking planning involves stability control. When considering static walking, the robot must be stable at all times. Steps taken by the robot must be checked to ensure that every move that the robot makes does not cause instability. Planning in advance can help to avoid all stances that are statically unstable.

3.3.2 Reactivity

A reaction is a response due to a stimulus. Reactivity in a system plays an important role in autonomy and stability of the system. A quick and sure response to an event ensures that the system maintains its stability regardless of the disturbances or events that occur.

In the area of walking, this involves maintaining stability in the event of obstacles, manoeuvring around uneven terrain, and the slipping of the feet. For autonomy, the robot must take into consideration these uncertainties and also to recover promptly if instability is detected. Once the robot falls over, it is impossible for it to recover without human intervention.

There are various methods to implement a reactive system. One suggested method is observed in human reaction is “symbolic” association [Brooks 1994]. This is the act of associating a set of inputs from the environment to a set of actions. This can be done online since this method requires little computation. This method emulates the way that a human neural system works. Depending on our sensory inputs, the body responds with a set of actions without feedback from the brain. For example, when human hand comes into contact with a hot object, it instinctively reacts by moving quickly from the hot object. There is no signal from the brain that tells us to move our hands away.

One form of robot controllers that involves this mechanism of reaction is the behaviour-based control. This controller is implemented in this project and is described in detail in Chapter 4. This form of control integrates modular architecture control, and symbolic association with sensory inputs. The interaction between the different modules is able to achieve walking without precise gait planning.

3.3.3 Anticipation

The robot's ability to anticipate has to do with predictive planning in order to prevent mistakes from happening. If the robot knows that there are many obstacles in the current path, the robot should work out another path, if possible, to reach the goal. This reduces the likelihood the robot might face problems.

This is obvious in human behaviour. If a man sees a wall in front of him and a door on the right, human brain will tell him that the wall is an obstacle. The body simply cannot move forward to hit the wall. The mind re-plans the route by moving to the right to get through the door.

Planning based on knowledge helps if the environment in which the robot moves in remains unchanged. The locations of all the obstacles are known and therefore the robot can easily manoeuvre within the space. This information can be acquired as the robot moves around.

Ability to anticipate can be thought of as the robot's arsenal to combat instability in the system. By studying the movement of the robot and analysing the situations the robot might face, a set of solutions is mapped to a corresponding sensor data. As the robot's arsenal or mapped pairs increases, the robot is able to react to almost all the situations that it might face, even those that were not perceived. As the pool of reaction pairs grow, the robot will be able to better deal with unstable situations.

3.3.4 Adaptation

Adaptability plays an important role in mobile robotics. Humans and animals adapt to their surrounding environment for survival. Animals develop different muscles depending on the type of terrain it has to overcome or the type of prey it hunts. Evolution occurs as the environment changes. Likewise, robots must have a certain degree of adaptation if it is to move robustly or autonomously.

Adaptation requires sensing of the environment via various sensors such as ultrasonic, infra-red and limit switches. The more the robot knows the environment, the better it is able to adapt to the environment. Adaptation without knowledge of the environment is impossible as it is a process of knowing and changing to suit the environment.

This can be observed with human beings. When clearing a rugged terrain, the first thing we do is to slow down and try to clear the different obstacles while assessing the situation. While the task of clearing is being performed, it adapts to the terrain. We find out about the hardness of the ground, the kind of objects we should and should not be stepping on, and what to look out for when looking for the next foothold. As we get accustomed to the terrain, speed and efficiency increases. This is the human's ability to adapt.

In summary, with the above three aspects, the following guidelines for the controller were formed. The controller should have:

1) the ability to react fast to sudden changes that cause instability to the system,
2) the ability to change the course of action depending on what the robot can sense about far away objects or past experiences, and
3) a stability control mechanism.

All the above are only achievable if the robot is able to sense, feel and see the environment. The minimum requirement for a human to move around without aids is to feel and see the surrounding. Adequate sensors must be added to facilitate this need to know the environment.

3.3.5 Errors in Sensing

Imprecise sensing of the environment may lead to the wrong perception of the world model. If no corrective measures are taken, the robot might perform the wrong action. If the noise in the proximity sensor causes a false reading to occur, the software would react as if it detects an obstacle in its path. The inability for the actuators to perform will cause the controller to fail as well.

3.3.6 Errors in Control

Precision in the movement improves the controllability of the robot. If the motor controller is not tuned properly, displacement error causes the robot to deviate from the goal. The accumulated error can be significant if the robot moves a substantial distance.

3.3.7 Modularity

Modular systems are useful in design of both software and hardware components. The following are the advantages of having a modular system.

- It is easy to add modules to the existing controller without having knowledge of the other parts of the controller, since each module is treated as a black box. It allows a group of developers to build different parts of the controller, which are based on a standard module interface.
- It is easy to debug, as the different parts of the robot controller are independent.
- Modules can incrementally be added to the system, making improvements a step at a time.

Building a modular system involves firstly dividing the entire system into sub-systems that can operate on their own. These sub-systems can be developed as independent modules each providing a set of inputs and outputs. The modules can interact with other modules which as a group can form a module on its own. The communications between the different modules play an important role in the integration of the entire system.

Modules are implemented at various levels. Some modules perform simple task such as move an axis forward. Other modules use these simple modules to perform a function such as moving the robot forward. Hence every module is a building block that can be used for further development of the system.

A standard for each module is required for developers to understand how to develop a module and also how to use a module. Hence the following properties were defined:

- A set of communication rules. Both the method and the content of the message packets have to be specified.
- An internal structure with different actions/functions.
- A set of registers for keeping track of the current status.

3.3.8 Expandability

Expandability of the system, in terms of software, is easy if the complexity of adding modules is reduced. Intelligence and functionality of the robot can be gradually developed as different components are incrementally added to the system.

Hardware expansion and modularity are not dealt with in this thesis but worth a mention. Building a modular robot is not a new concept. It gives the designer the opportunity to change and upgrade the robot without having to make major modifications to the robot. An understanding of this system allows other developers to add other hardware components to the robot without the need to understand the robot's system as a whole. LEGO$^{®}$ Mindstorms^{TM} is an example of a modular robotic system; as is PolyBot [Yim 2000], a modular robot built by Mark Yim of Stanford University.

3.4 Software Design Considerations

The controller is implemented on a PC and, therefore, the software methods are important. This section deals with the different software requirements, design factors and the tools used.

3.4.1 Modular Software Design

Modularity can be developed in software using object-oriented development tools such as C++. Object-oriented has the following properties, namely class structure, inheritance, and abstraction. These properties are used to develop a software structure that incorporates expandability, modularity and reusability into the controller. The class structures of the controllers and how the different properties of classes will be developed as the implementations of the two controllers are explained in the subsequent chapters.

3.4.2 Real-Time

The timeliness and accuracy of the motor voltages sent to the motors and the sensor information read are crucial to the operation of the robot. A failure would cause the robot to go into an indeterminable state rendering the robot inoperable, hence, a deterministic real-time system is required. In this section, the idea of real-time is introduced and a study of the current real-time solutions is presented concluding with the explanation of how the real-time system works.

3.4.2.1 Real-Time Response

Real-time response is one in which the correctness of the computations of the response not only depends upon the logical correctness of the computation of the response but also upon the time at which the response is produced.

Response plays an important role in any system. An autonomous response must be correct and timely. Therefore, real-time must be incorporated in the robotic system to be able to perform effective evasive actions and to react to unpredictable situations. Moreover the amount of actuators and sensors used in this system require that no bottleneck compromise the performance of the system.

3.4.2.2 Real time Requirements

In addition to real-time response, there are a number of other software tools that real-time systems provide:

- A multi-threaded, preemptive scheduler with a large number of thread priorities.
- Predictable thread synchronization mechanisms.
- A system of priority inheritance.
- Fast clocks and timers.

3.4.2.3 Real-Time Operating System Selection

There are many factors to consider when selecting real-time operating systems. These include the tools available on the operating system, hardware compatibility of the operating system, and the price of the operating system.

There are many real-time operating systems available which cater to the design requirement. There are some real-time operating systems that are variants of the UNIX platform and some extensions to currently available operating systems such as Linux and Microsoft Windows. These are the more popular operating systems and extensions:

- QNX,
- VxWorks(R),
- Venturcom's RTX,
- Lynx's LynxOS.

Other then Venturcom's RTX, The rest are UNIX-based operating systems. Venturcom's RTX is a hard real-time extension to the Windows NT/2000 platform. Venturcom's RTX was selected to develop the robot controller program as it works on the popular Windows platform.

Why Windows NT/2000?

Windows NT and Windows 2000 are powerful operating systems used widely in industry today. It is used at all levels of the industrial hierarchy so as to provide a heterogeneous computing environment. Developing software on this platform allows easy access to the tools currently available in the current system. There need not be extra training for users to operate in a new operating system environment. Due to its popularity, many established tools are available on Windows NT/2000. This gives us the opportunity to use these tools to develop the robotic system. These tools include Microsoft's Visual C++, and various C libraries provided for the Windows platform. The Win32 API in Visual C++ is flexible and contains many libraries with tools that can be used.

What is lacking in Windows NT/2000?

The Windows NT platform has been developed for general use and lack the tools required for real-time control. The lacking features include:

- too few thread priorities,
- non-deterministic scheduling decisions, and
- priority inversion, particularly in interrupt processing.

The programmer cannot take full control of the system in terms of priority and scheduling of threads, therefore deeming it unsuitable for deterministic control.

Hardware Access Control - Windows NT/2000 Hardware Abstraction Layer

Hardware is not easily accessible in Windows NT based systems. This becomes a problem when control requires communication between devices connected to the PC. In Windows NT/2000, the hardware abstraction layer (HAL) is a layer of programming that allows interaction with a hardware device at a general or abstract level rather than at a detailed hardware level. The hardware abstraction layer can be called from either the operating system's kernel or from a device driver. In either case, the calling program can interact with the device in a more general way than it would otherwise.

VenturCom's Real-time Extension (RTX)

RTX enables Windows developed application components to have deterministic and high-speed response times. It takes control of Windows NT/2000 to give the operating system its real-time capabilities by adding a real-time subsystem (RTSS) to Windows NT/2000. Figure 3.8 shows a schematic diagram of a Windows with RTX environment.

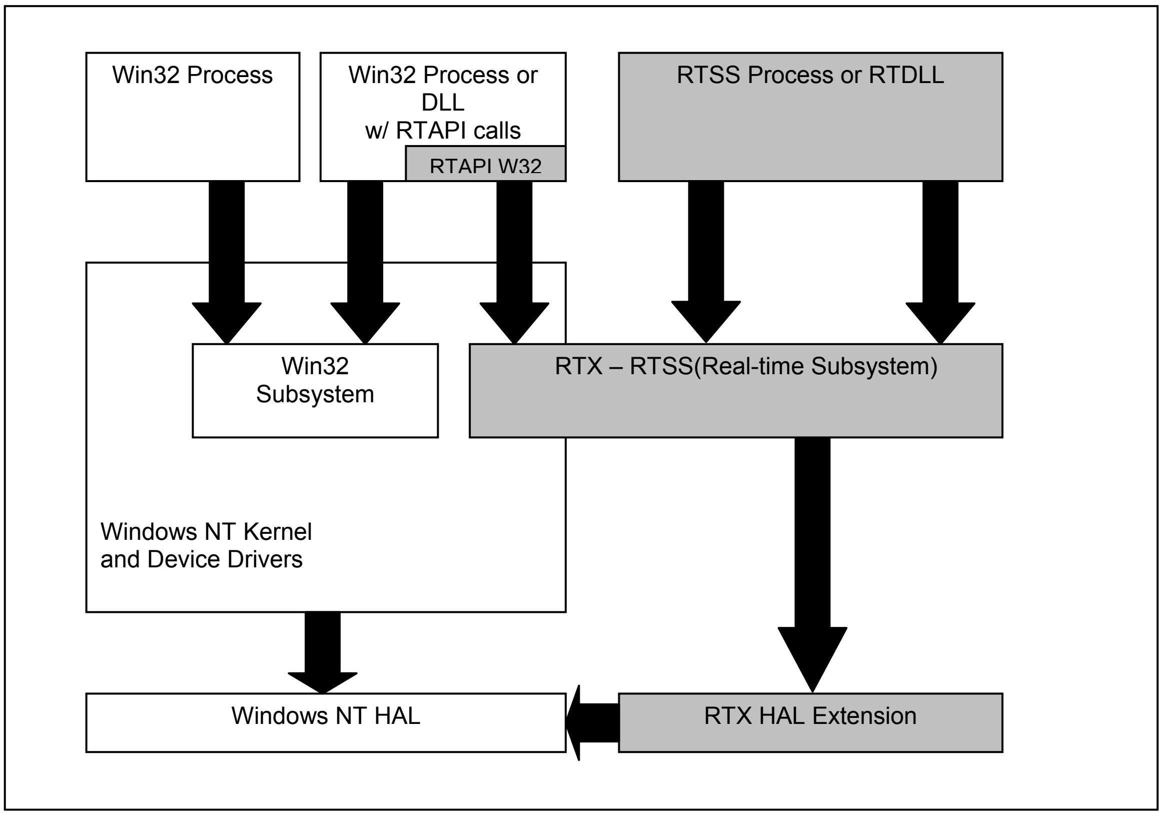

Win32 Process
Win32 Process or DLL w/ RTAPI calls
RTAPI W32
RTSS Process or RTDLL
Win32 Subsystem
RTX – RTSS(Real-time Subsystem)
Windows NT Kernel and Device Drivers
Windows NT HAL
RTX HAL Extension

Figure 3.8 – An overview of how RTX works with Windows NT/2000

Programs are written using Microsoft's Visual C++ 6.0. RTX supports a set of Win32 API (Application Programming Interface) functions, and provides its own set of functions known as RTAPI (Real-Time API). Consequently, user-friendly Win32 programs can be easily developed with real-time capabilities by combining the two APIs.

The RTSS also supports Inter-Process Communication (IPC) objects that can be manipulated by either RTSS or Win32 processes; this enables simple and standard communication and synchronization between real-time and non-real-time programs. Finally, the RTSS provides other time-critical services, such as clocks and timers and interrupt management, to the RTSS processes.

The RTSS is able to

- Preempt Windows NT/2000 anywhere, at least outside critical NT interrupt-processing code.
- Defer Windows NT/2000 interrupts and faults while running real-time tasks.
- Process real-time interrupts while running real-time tasks.

These tools are essential for developing a real-time program. Since RTX provides library functions for Visual C++, there is interaction between the various real-time processes and the user interfaces developed using Visual C++. Hence various software tools can be developed for use with the robot.

3.5 Summary

The quadruped used in this project is built based on the pantograph leg design, with sensors to guide it in its movement. These sensors include proximity sensors for obstacle detection and force sensors that trigger off when the foot of the robot touches the ground. Previously, a low-level controller was developed for motor control with a high-level controller to supervise control – the Behaviour-based controller. The design of the robot was analysed to assist in the development of the current controller.

The goals of the current project were defined. The aim is to develop a robust controller for the robot with sound software methodology. The platform and software used for the development of this controller were discussed along with the software requirements and design specification. Also included in this chapter is a brief mention of the software tools used – Venturcom’s RTX and Microsoft Visual C++.

Chapter 4

CONTROL ARCHITECTURE AND HARDWARE CONTROLLER

4.1 Programs Developed

The controller consists of two main parts – the global control and the hardware control. The hardware control directly communicates with the hardware and calculates the control output to the various motors. The hardware control requires deterministic and quick response to ensure the actuators and sensor reading are current. It forms the base of the entire robot controller.

This chapter explains two components of hardware control – the RTX process that performs data acquisition on the hardware and the fuzzy controller used to calculate the output voltage to the motors.

4.2 Hardware Interface Program

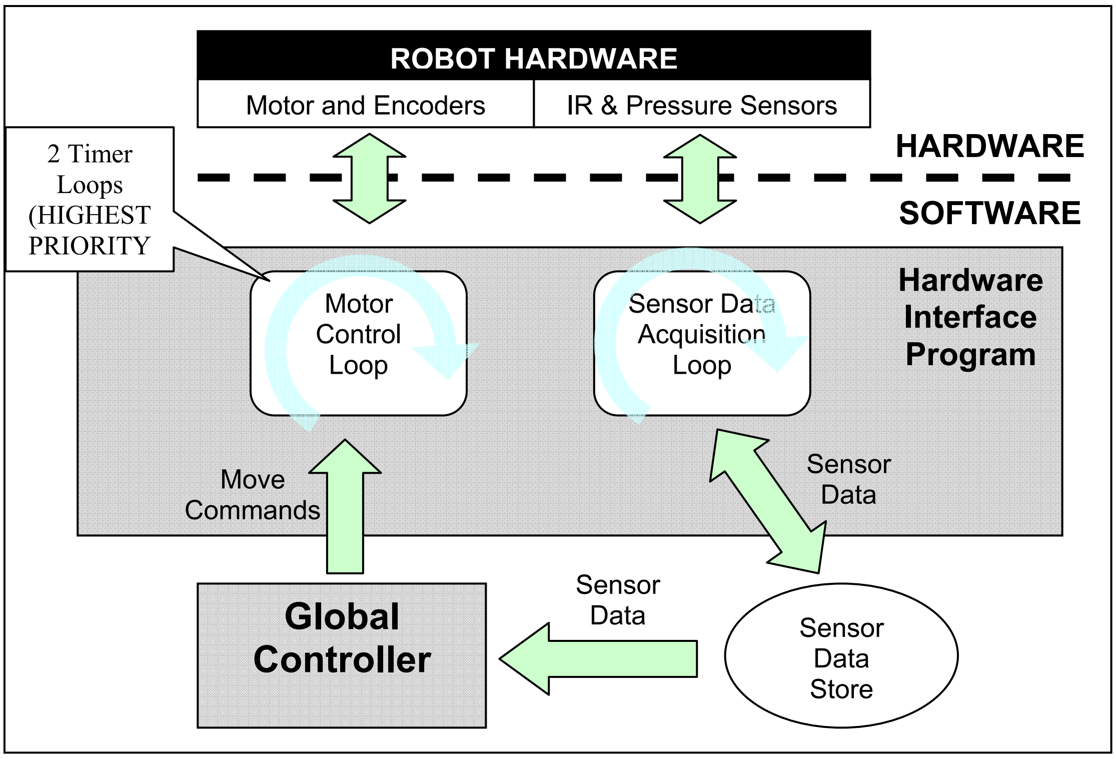


Figure 4.1 –The diagram shows the hardware interface program with 2 control loops. The motor control loop handles control of the different motors, and the sensor data acquisition loop collects sensor data used by the global controller.

This program takes care of the communication between the hardware and the other programs. There are two loops in this program, the motor control loop and the sensor data acquisition loop (Figure 4.1). The first loop takes care of the motor and encoders required for motor control. The second loop collects data from the sensors which are not as critical as the motor control loop. This includes data from the proximity sensors and the force sensors. This is stored in shared memory where it is read by the global controller. Both loops have a period of five milliseconds. All these timer processes have higher priority than the Windows processes and thus are not interrupted by Windows processes. Both timer processes have the same priority in this system (using the RTX real-time extension function).

4.3 Fuzzy Controller Program

The hardware controller in this robot controls the individual motors of the robot. The dynamics involved in control of the motor requires a controller that changes to suit the environment accordingly.

Dynamic control condition for the robot lifting up from the ground is different from when its leg is off the ground. The controller must take into consideration these factors. A PID controller does not work well when the control condition changes as the control environment is different for when the foot is on the ground and when it is not.

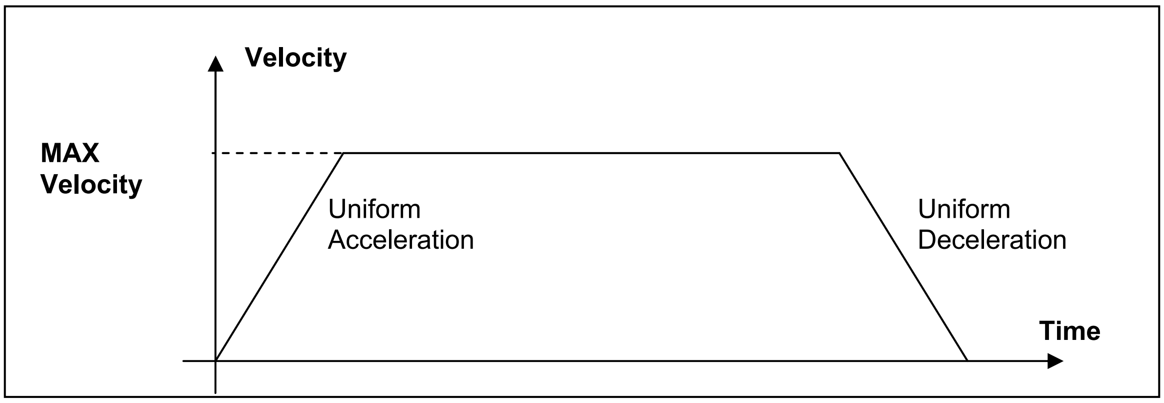


Figure 4.2 – Velocity profile diagram

The controller produces a velocity profile as show in Figure 4.2. The controller must be able to initially accelerate uniformly to a maximum velocity, maintain the maximum velocity until the joint moves close to desired position where the motor decelerates smoothly to a complete stop at the desired position without overshoot.

A colleague, Kong Yong Min, developed a fuzzy controller for his hexapod [Kong 2003]. His controller was modified and was used for the quadruped. An explanation of the fuzzy controller is given in Appendix A.

When humans walk, the distance travelled cannot be accurately measured by calculating the number of steps taken, especially if the terrain is uneven. Therefore, when a robot walks, there is no requirement that the distance travelled with each step is measured. The aim of walking is to move in the specified direction; hence, the error associated with the motors during walking is tolerable. Since legged locomotion does not require precision in the movement of the legs, the acceptable range for the axis positional error was set to ±2 mm. This value was selected based on the minimum stable condition for the fuzzy controller, and it is acceptable value for legged motion. Stable motion of the motors on the different legs were obtained.

4.4 Global Control

The global controller is the brain of the robot. This is the robot's ability to move intelligently and also to work itself out of situations using the available sensory feedback and various algorithms. It acts above the hardware control to ensure the robustness and stability of robot's gait. Two global controllers were developed in this project – the Behaviour-based controller and the Central Pattern Generator controller. These controllers will be discussed in the following two chapters.

4.5 Summary

The controller consists of three parts, the two hardware controllers (the hardware interface and the fuzzy controller), and the global controller. The hardware controller handles control of the motor based on the fuzzy controller. The results of the fuzzy controller achieved stable motion of the individual axes hence the development of the global controllers was possible. Two global controllers were developed in this project, the Behaviour-based controller and the Central Pattern Generator controller, which will be elaborated in the following two chapters.

CHAPTER 5

BEHAVIOUR BASED APPROACH

5.1 Introduction

There are many control methods for legged robotics that can be divided in two fundamental categories – planner-based control and reactive control. In the robotics field, researchers use variation of these two controllers to control legged locomotion. The behaviour-based controller, a hybrid of the two controllers, is presented in this chapter. This method of control was popularised by Brooks with his work on the Behavior Language [Brooks 1990]. The behaviour-based controller was first used as a controller for wheeled mobile robots [Brooks 1986] and has been modified for legged robot locomotion [Wettergreen 1995a].

The behaviour-based controller is an incremental module-based controller that uses a combination of simple modules to build a structure that achieves an overall goal. Each module independently is able to perform simple tasks. Different groups of modules, each perform unique task, integrate to form the control mechanism used for the quadruped.

5.2 Planner and Reactive Approach

The behaviour-based controller is a hybrid between the planner approach and the reactive approach controller. A clear understanding of these two controllers will assist in the comprehension of the development of the behaviour-based controller.

First, the planner-based approach plans the action of the robot according to a centralised world model, which is pre-programmed into the controller. Sensory information verifies the known world model, which in turn generates a sequence of actions that the robot performs [Mat'aric 1992]. This method requires the model to be accurate and the sensors to give precise information.

The reactive approach has minimal internal states and works on looking up the correct action based on an appropriate set of conditions. These conditions are sensory information from the robot. This form of control is computationally efficient. It has the ability to react to situations at the pace of the environment. There are no predictive components and therefore it lacks in the area of run-time flexibility since is has little representational power. This form of control cannot handle complex tasks as it has no memory capability. Knowledge of the control environment is required to design the condition-action pairs.

5.3 What is the Behaviour-based Approach?

5.3.1 Properties of the Behaviour-based Approach

5.3.1.1 As an Entity

Each behaviour module can be viewed as an independent sense-plan-act element which achieves a sub-goal of the entire system, with their unique character. Each module acts as a reflex, which is a collection of rules that react according to the inputs from various sensors, including signal from other modules, the current state and goal of the robot. The module changes state either upon completion of task or on receiving signal from the various other modules. These signals are binary signals that either start or stop the module’s task. One guideline is to select modules that maximize sensory awareness and make small adjustment to the overall movement of the robot, thus increasing the precision of the task.

The other property is that only required sensory information is piped directly into the modules, hence sensory information is task related. This reduces the amount of processing required by each module. There is no centralized controller that combines all sensory data to provide a global model of the environment for the system.

5.3.1.2 As Part of the System

The initial step taken in developing the behaviour-based controller is to breakdown the overall goal of the system into different manageable and observable sub-goals or behaviours. The development of this controller progresses from the bottom-up, i.e. a collection of simple reactive rules are first developed and the combination forms the complex task modules of the system.

There is no central mechanism that coordinates the different modules in the system. The interaction between the various independent components and their characteristics define the tasks in the system. The integration of the different sub-goals working together defines the role played by the system. The links define how the different modules interaction with each other and are between the modules are added when required. The integrity of the entire system must be maintained when planning the interaction between modules.

As the system grows in size, the complexities involved in the communications may become insurmountable. This complexity is reduced by minimizing the interaction between the various modules in the system. This is achieved as the modules obtain most of their information, either through the sensing of the environment, or from the robot's current state and goals, which does not require communication with other modules. As such the communication packets between modules are kept as a simple binary signal.

The behaviour-based controller is an incremental controller. The design of the controller starts with a group of modules that perform a simple single task. This group of modules can form a module of its own with its own inputs, outputs and characteristic. These simple combinations are termed as "black boxes" where its content is not important but the interface and overall ability of the function is. This allows different developers to develop different black boxes and combine them to provide a complete controller by defining the interface between the various black boxes.

5.3.2 Importance in Implementation

As a behaviour-based system depends on interaction between the different behaviours, it is difficult to determine the action of the system. The coordination of all the behaviours and synthesizing of the results has always been a problem especially when the system increases in size. There are no rules or guidelines that help the user connect the different modules together. Therefore, various methods have been used to reduce the complexity of the network and increase predictability of the system so as to improve the performance of the controller. Brooks laid down the following guidelines to integrate behaviour-based system ([Brooks 1991a], [Flynn 1989]).

- Coherence –There should not be conflicts between the many behaviours acting simultaneously. There should be coherence in action and goal of the system as a whole to an observer.
- Salience – Active behaviours should react to situations that the system is currently facing and not be concerned with things that are not happening.
- Adequacy – The selection method should ensure that the long term goals of the system are met no matter what happens. The controller might face situations where it digresses from the goal. It must realise the error and bring the robot back in track.

The next section gives an overview of the different implementations of the behaviour-based approach.

5.3.3 Relationship between Behaviours

One problem with behaviour-based approach is the coordination of the different modules where there are conflicts in the interest of the different modules. There are various methods used to overcome this problem. These methods are known as Action Selection Mechanisms (ASMs) which can be divided into two groups, known as arbitration and command fusion. Arbitration methods are suitable for behaviour selection which include methods such as Discrete Event Systems [Koŝecká 1993], the Subsumption architecture [Brooks 1991a], and activation networks [Maes 1989]. Command fusion is suitable for coordinating the activities between various simultaneously active modules. These include the potential-fields methods [Arkin 1989], multiple objective action selection [Pirjanian 2000] and Payton-Roseblatt voting approach [Rosenblatt 1997].

5.3.3.1 Priority based

Since many behaviour modules act independently, there might be conflicts within the system. A priority-based mechanism allows high importance behaviours to take precedence over lower priority behaviours.

For example, the avoidance behaviour should have a higher priority than normal walking behaviour. If the robot continues walking with the obstacle in front of it, it would cause damage to the robot. The robot must stop immediately to assess the situation before making another move. This method ensures that the relevant behaviours take precedence immediately.

5.3.3.2 Hierarchical based

This method is an organisational method where the behaviour modules are spilt into self-contained functioning groups or layers. These layers can be considered as single modules each with its set of inputs and outputs that connects either to other layers or other modules in the system. Each layer can perform more complex tasks but provide the same structure as an individual module on its own. This provides a systematic method to organise the complex structure of this controller.

Connecting and coordinating four layers that each move a single leg can move four legged robots. Other modules that perform obstacle avoidance, navigation and clearing obstacles can be added which supersedes the previous four modules to take control of the robot as required. A more robust and autonomous system can be built by combining these different layers.

5.3.3.3 Other Methods

On top of those mentioned, other methods to improve controllability and stability of behaviour-based controllers include:

- parameterization [Likhachev 2002] to guide the behaviour,
- summation of the weighted sum of the output of behaviours [Arkin 1989a],

- decision making prioritized based on environmental stimuli (sensors) and task at hand [Xu 1997],
- using neural networks [Al-Jumaily 1999], fuzzy logic, reinforcement techniques [Digney 1993] and genetic algorithms,
- selectively enabling and disabling the behaviours,
- filtering the inputs and outputs to change the response, and
- enabling them to learn to improve performance [Maes 1990].

5.4 Various Approaches

A major part of the author's work is based on the following two implementation of the behaviour-based approach – Brooks' Subsumption Architecture and Wettergreen's work on behaviour-based controllers.

5.4.1 Brooks' Subsumption Architecture

Brooks R. has done extensive work on behaviour-based controllers ([Brooks 1991], [Brooks 1991a]). In his work, Brooks designed a behaviour-based robot controller for a mobile wheel robot, which was later implemented on various robots.

The behaviour-based approach is an integration of the planner-based and reactive-based approach. Unlike the planner-based approach, the behaviour-based approach works with a map of conditions to actions. Although this is a property of reactive systems, it differs in that the behaviour-based controllers use different forms of internal representations and performs computation within the modules to determine the correct action. This is a modification made to the reactive approach. Another distinct property of the behaviour-based system is its distributed nature where it consists of parallel running interconnected modules executing independently.

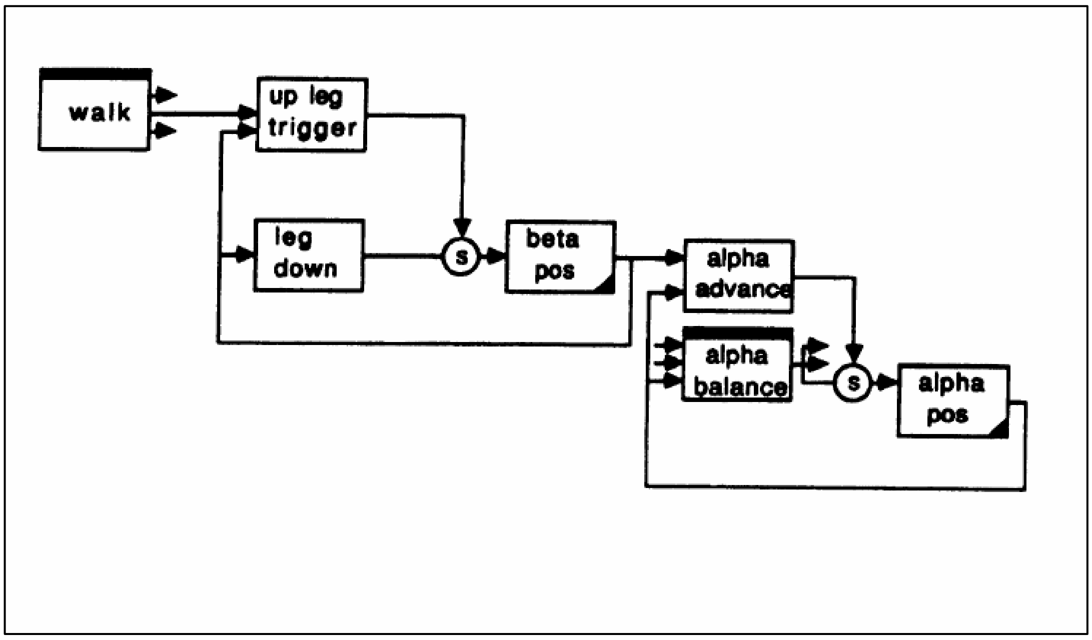


Figure 5.1 – An example of Brook's behaviour-based controller represented [Brooks 1989]. The various tasks (FSMs) are represented as modules with interactions between the tasks are denoted by the arrows.

Brooks' controller uses a layered control system, which he coined "the Subsumption Architecture". Each basic unit of the controller is a module based on finite state machines augmented by internal registers and timers. The controller is made up of layers of these modules (Figure 5.1). Each layer of the controller consists of a group of module where different layers are built to perform different tasks. The inputs to these systems are sensory signals that tie directly to reflex action performed by the robot. Different layers can subsume the behaviour of other layers by suppressing their output.

The aim is to build a network of modules that forms an abstraction of a specified task. Once these task-specific layers are created, they can be treated as modules of their own - a black box with a specific task, and its own set of inputs and outputs. The system improves in capability as more independent task oriented layers are added. A robot controller can be developed by combining and connecting the different layers and the individual modules.

Subsuming layers can be useful at times but may not work all the time. Some task requires that the lower layer and higher layer subsume each other but this is not possible with the hierarchy in place. A multi-layer architecture is not suited for this case.

5.4.2 Wettergreen's Work on Behaviour based Control

Wettergreen's work [Wettergreen 1995] surveys the field of behaviour-based controllers. He implemented a behaviour-based controller for an exploration legged robot. He expanded on Brooks' work to develop controllers for statically-stable walking robots.

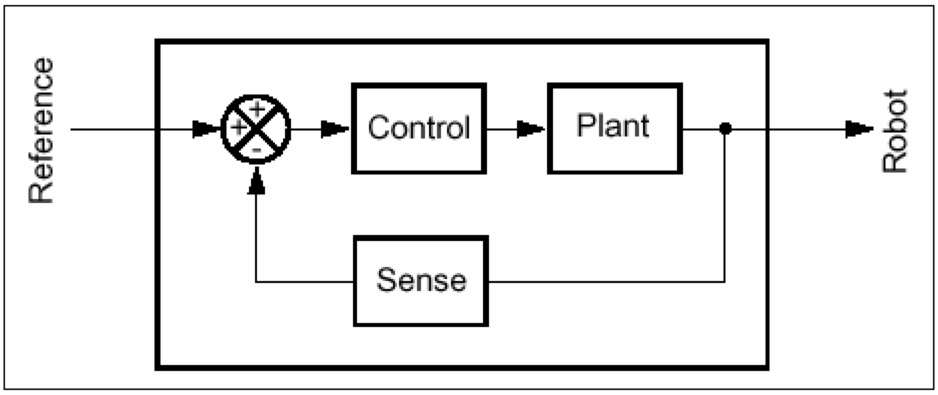


Figure 5.2 - A block diagram of a typical behaviour.

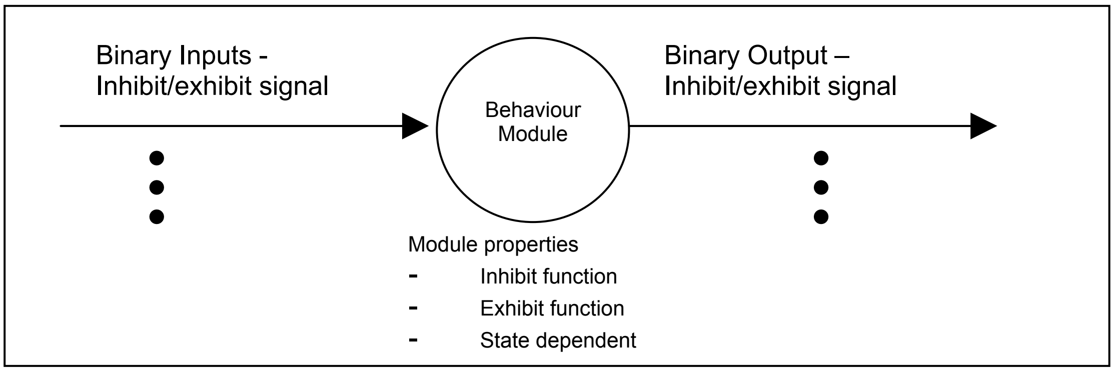


Figure 5.3- Typical module with binary inputs and outputs

A behaviour module can be thought of as a simple feedback control loop as shown in Figure 5.2 with a set of different registers and properties as shown in Figure 5.3. The individual module has the following properties:

1) Each module performs a unique task. Each has a set of binary input and output signal lines that transmit or receive inhibitory or exhibitory signals. These signals are asynchronous messages that are not tied to any clock. Only inhibit and exhibit signals are communicated to the other modules.
2) All behaviours share a standard module structure. The modules are structured where functions are executed upon receiving a signal from other modules or changes in sensor data. There are two action functions in each module, an inhibit function and an exhibit function, which acts depending on the signal received.
3) Every behaviour module is independent of each other.
4) There are two ways to cause as action in the behaviour, either by a change of state of the behaviour or a signal sent to the behaviour.
5) There are arbitration mechanisms to take care of conflicting modules as well.

Wettergreen produced a series of behaviours to construct a walking gait controller for a few legged robots. The controller was used to control a hexapod for exploration of volcano craters. It was able to autonomously traverse through the rugged terrain. It was capable of clearing obstacles and manoeuvring itself to avoid them. It was also able to maintain stability by shifting all legs and was able to perform different gaits depending on the situation, changing them on the fly.

Wettergreen observed the walking control system for animals. It is a combination of pattern behaviour and reflexive coordination. Reflexive action is the feedback mechanism with information obtained from various sensors in the system. All animals and insects have a neural architecture in a single layer. This can therefore get chaotic when there are so many neurons in the system. The way to deal with this problem is to place groups of neurons in a multi-level architecture. Thus Wettergreen developed his controller in multiple layers, similar to Brooks' Subsumption architecture.

5.5 Advantages and Disadvantages

Since the behaviour-based system is a hybrid reactive system, it responds rapidly to environmental changes. Therefore, it is robust in the real world. It also requires less demanding computation. Another advantage of the behaviour-based approach is its modular structure. The modules are created separately since each module can operate independently of the other modules. By combining different behaviours, new behaviours with different functionalities are created. There is no need to be concerned with sequence of events with the behaviour-based approach. As long as all sense and react pairs are defined properly, the behaviour modules will take care of the movement or reaction of the system.

The behaviour-based approach faces difficulties in system modularity, state representation and integration of world model. Certain tasks such as transportation, cleaning and assisting disabled people require knowledge of the environment. This is not present in the behaviour-based approach. Another disadvantage of the behaviour-based approach is that the complexity of the system increases as the structure of the system grows in size. This is not a desirable trait in control. Brooks solved this problem by splitting the different task into different layers. By doing so, new modules are created and the amount of connections between these new modules reduces as they are hidden from the modules outside the group. Lastly, the behaviour-based controller relies on the precision of the sensor reading, actuation, computation and other reading from the system which might cause problems as there are various shortcomings associated with the devices used.

5.6 Proposed Approach

A modified form of the architecture for the behaviour-based controller based on Wettergreen's and Brooks' work was developed in this thesis.

5.6.1 General Properties

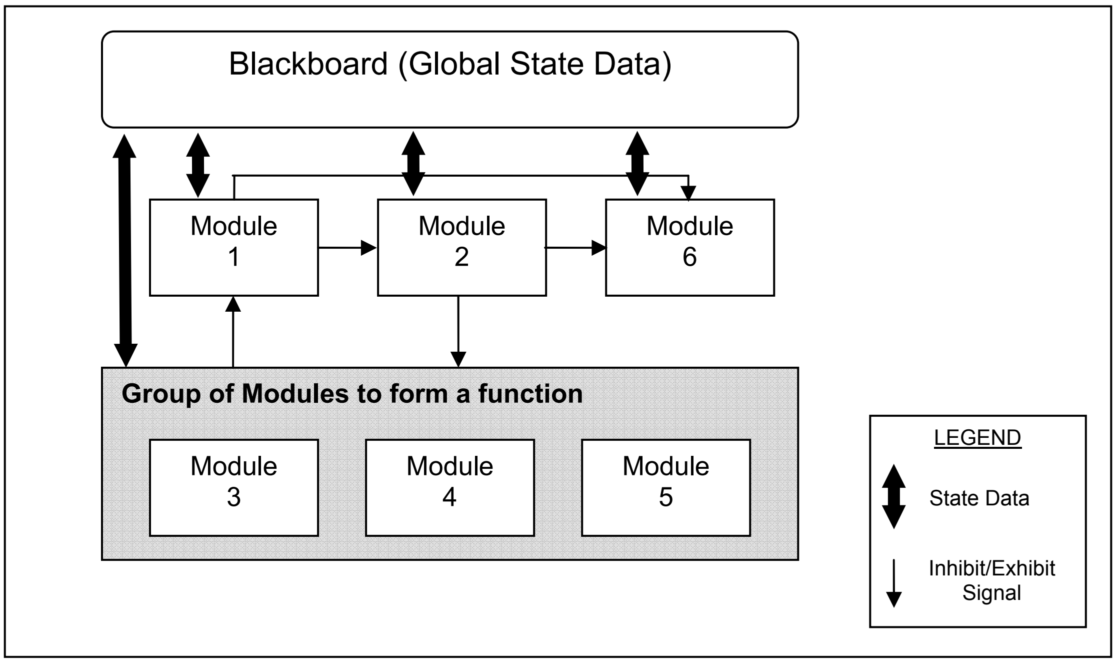


Figure 5.4 –An example of the Behaviour based Controller.

Similar the all behaviour-based controller, the design of the controller starts with a basic module that performs a simple task. Once a group of simple modules are constructed, the combination and connection of various modules together form complex task modules (Figure 5.4).

Brooks' behaviour-based model does not permit communication of state data between modules. This is resolved using a blackboard model which will be explained later. A module can query about the properties of other modules. Only information required by all the modules is shared. This information include the current state, and the inhibit/exhibit status of the modules.

5.6.2 Properties of a Module

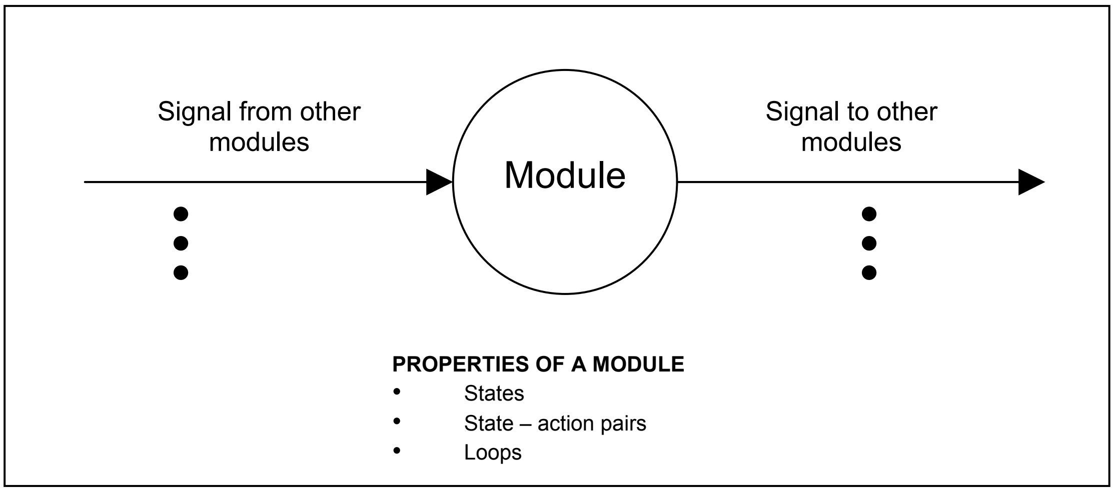


Figure 5.5 - A diagram of a typical module in this system

Each module has its own character – it has its own set of states and actions. Actions are mapped to the different states in the module. Changes in a state occur either due to a completion of a task, or a signal from another module. A typical module is shown in Figure 5.5.

5.6.3 Message Passing

A group of modules is not only defined by the combination of their individual traits, but also by the way they are connected. Content of the messages passed between modules are kept as minimal as possible. Therefore, message packets contained only three parameters, the first two to identify the sender and receiver, whereas the last parameter is the inhibit/exhibit binary signal. An exhibit signal wakes the module from its suspended state, starting the mapped task. An inhibit signal stops the receiving module from continuing with its current task and goes into IDLE state.

The *Raise Leg* module is used as an example to demonstrate how message passing works. During the static walking, a quadruped can only raise one leg at a time. Therefore, whenever one of the *Raise Leg* modules is exhibited, it inhibits all other *Raise Leg* modules, to prevent the robot from falling over. Similarly, the legs should not move individually when it is on the ground. The *Forward Leg* module can only be exhibited by the *Raise Leg* module. Therefore, the inhibition of the *Raise Leg* module prevents the *Forward Leg* module from being exhibited. Message passing is state dependable. All modules are programmed with a set of signals to transmit to other modules.

5.6.4 State Dependency

This system is considered a state-driven or state-dependent system. The activation of actions of the module happens only when the state changes. Every module has a set of unique states. Each action is specific to individual modules depending on its task. The initial state of all modules is the IDLE state. The only way for the system to change to another state is for it to be triggered by an external signal or a sensor or a completion of an action. An inhibit signal from other module will immediately change the module's state to IDLE. An exhibit signal will cause a module to change its state to ACTIVE. This rule always applies unless there are special circumstances.

There are four states defined in the *Forward Leg* module – ACTIVE, MOVING, IDLE, and DONE. An exhibit signal from another module changes the state of the *Forward Leg* module. Upon receiving the exhibit signal, it changes from the IDLE state to the ACTIVE state. In the ACTIVE state, the module sets the destination for the respective motors to move the leg forward and changes state to the MOVING state. The ACTIVE states allow for exhibiting or inhibiting other modules and coordination between the different modules. The module remains in the MOVING state until all motors move to their respective goals before switching to the DONE state. Once in the DONE state the module performs sends a command to stop all the motors from running and send out the respective signals to other modules before proceeding back to the IDLE state.

5.6.5 Loops

There are two loops running in each module. The first being the continuous loop which runs at intervals, like the timer in Brooks' Subsumption Architecture. The other is a triggered loop that reacts upon receiving a message.

The period for the continuous loop is set to one millisecond. The module will behave according to the current state of the module. The continuous loop can also be tasked to check for a change in sensor reading or to complete an action required by the module.

```
void behaviorCtsProcess(){
      while(true){
            executeCtsFunction(); // contains functions that are
                                  //active depending on the state
      }
}
```

Listing 5.1 – Continuous loop function structure. There is no wait function as the loop executes continuously.

```
void pendonMessageQueueForExhibitOrInhibit(message){
      WaitForMessage();
      registryUpdate(message); //updates the registry either with
                               // inhibit or exhibit signal
      behaviourTriggeredProcess(message);
```

Listing 5.2 – Triggered loop function structure. This function waits for an event before updating the registry and performing an action.

The triggered loop is the event-triggered loop. It is triggered by a signal from another module. There are two functions in this `behaviourTriggeredProcess` function, one activated by inhibiting signals and the other by exhibiting signals. The function, `behaviorCtsProcess,` in the continuous loop acts upon changes in state. This function contains state-action pairs, where an action occurs depending on the state of the module. The state mechanism is explained in the next section.

5.6.6 State Table

Another feature of the behaviour module is that it contains a state table. A state table contains four parameters: the inhibit/exhibit signal received, the current state, the previous state and the resulting state. The module keeps track of its current state and its previous state. When a module receives an inhibit/exhibit signal, the module checks the signal with the current state and previous state. If the data matches an entry in the table, the module changes to the corresponding next state in the table. Table 5.1 shows an example of a state table, the *Raise Leg* state table.

Every time a change of state occurs, the modules check the state table and act accordingly. Upon receiving the inhibit signal, the *Raise Leg* module changes state to

STOP without considering the previous and current state. If the current state is START and the state completes its task, it will switch state to MOVING. As can be seen, there are two conditions for a state change; upon completion of task within a state or upon receiving an inhibit/exhibit signal.

Inhibit/Exhibit	Previous State	Current State	*Next State*
Inhibit	X	X	***STOP***
Exhibit	X	X	***START***
X	X	STOP	***DONE***
X	X	START	***MOVING***
X	X	MOVING	***DONE***
X	X	DONE	***IDLE***

X – don't care state

Table 5.1 - State table for the Raise Leg module.

5.6.7 Inhibit/Exhibit Table

The inhibit/exhibit table has a list of states and with the set of inhibit/exhibit signals to be sent out. A set of inhibit/exhibit signals is sent out to other modules listed in the table depending on the current state of the module. The module looks up the inhibit/exhibit table for the signal to send out to the respective module. A mapping table is constructed in the program for each inhibit/exhibit table. This table can be constructed manually using an addInhibitRef function (Listing 5.3). This function is initialised in the constructor of the module class.

```
BackLeg1_Module:: BackLeg1_Module:: (){ // constructor of Back Leg 1
                                        //Module
…
addInhibitRef(FORWARD_LEG_1,INHIBIT); // Inhibit Forward Leg 1 Module
addInhibitRef(AVOID_LEG_1,EXHIBIT); // Exhibit Avoid Leg 1 Module
…
}
```

Listing 5.3 – Adding inhibit and exhibit signal to the module.

Referring to Table 5.2, if the current state of the *Lower Leg* module is START, inhibit signals will be sent to both the *Raise Leg* module and the *Forward Leg* module. Once the *Lower Leg* module completes its move (DONE state), the *Raise Leg* module of the next leg is exhibited.

Current State	Inhibit Signal to be Sent	Exhibit Signal to be Sent
IDLE		
START	***RAISELEG, FORWARDLEG***	
MOVING		
DONE		***RAISELEG(next leg)***

Table 5.2 – Inhibit/exhibit table for the Lower Leg module

5.6.8 Blackboard model

The behaviour-based architecture is highly distributed, with each module performing a specific task. Therefore sharing of system states and knowledge has been inconvenient. The Blackboard model serves the purpose of keeping a set of module data of the entire system. It ensures an up-to-date database of information for other modules to access. This was implemented in [Xu 1997]. The blackboard model consists of the following three components shown in Figure 5.6.

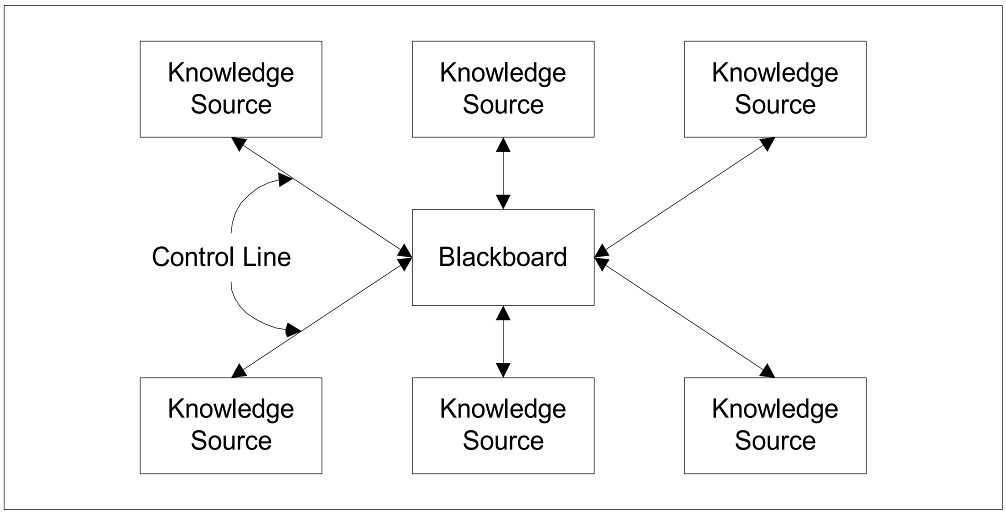


Figure 5.6 – The blackboard model

The ***blackboard*** serves the purpose of keeping a set of data used for problem solving. Data communication only happens through the blackboard.

The ***knowledge sources*** are independent modules that produce change in the data kept in the blackboard. These are the behaviour modules used in the behaviour-based controller.

The ***control lines*** monitor the changes in the blackboard and decide on the next action to be taken.

This is a modification of the behaviour-based concept. Every behaviour should act independently without the knowledge of the actions taken by the other behaviours. Certain problems may arise due to the lack of information about other modules. The blackboard model acts as an information server for the entire modular system in methodical fashion. This centralized system makes it easy to incorporate different data communication processes but information transfer is controlled to prevent total transparency within modules. Modules only require information crucial to performance.

In this current system, the blackboard model displays the following two information – the current state of the module, and the inhibit/exhibit status of the modules. The controller is able to coordinate the various modules that are using common resources.

5.7 Global Control Program

Figure 5.7 shows the structure of the behaviour-based controller program. Since the Hardware Interface and Fuzzy Controller have been explained in the previous chapter, this section will deal only with the behaviour-based controller. Both the Fuzzy Controller and the Behaviour-based Controller send and receive data from the Hardware Interface module and they act independently of each other.

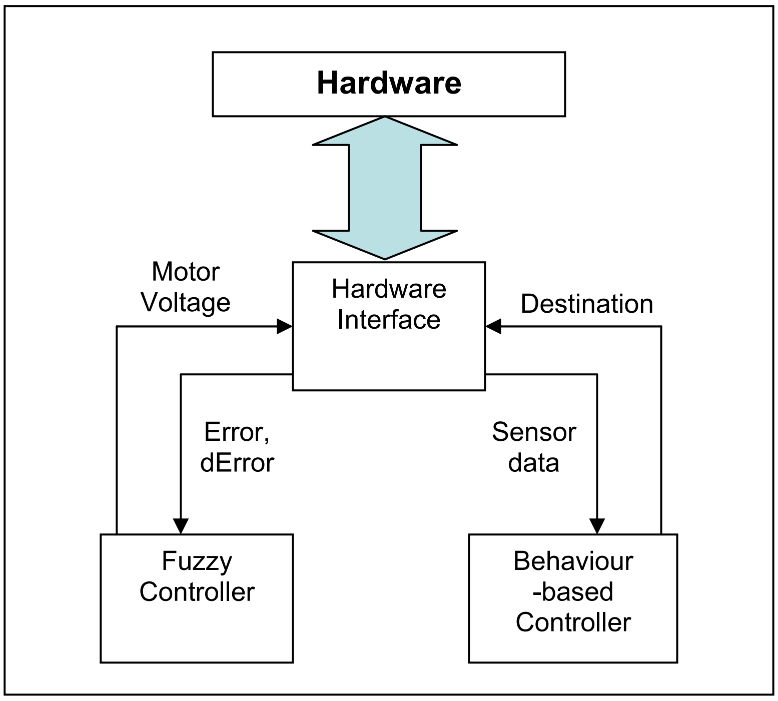


Figure 5.7 – Block diagram of the behaviour-based controller program.

The behaviour-based controller consist three major different parts, each defined as a C++ class. The classes are as follows:

1) Motor class – contains functions that move the different motors (axes) of the robot.
2) Sensor class – contains functions that retrieve information from the different sensors on the robot. These include the force sensors, proximity sensors and the encoders. This module allows code to be added easily for new sensors that are added to the system.
3) FSM class – this is the base class that all the created modules inherit. It contains the different mechanism functions of a module, all of which has been explained in the mechanism of behaviour-based controller, such as the two loops, the message passing mechanism, the state table, the inhibit/exhibit table and the different registry values. A schema and method to create different C++

classes for FSMs for use with the developed program is shown in the Appendix B.

4) Blackboard class – a pool of resources, explained in the Blackboard model section, for all Finite State Machines (FSMs) to retrieve data from.

Note that in the program the modules are known as FSMs. The following introduces the various algorithm used to improve the robustness of the controller.

5.7.1 Centre of Gravity (CG) Testing in the Controller

Modules to move individual legs were developed, and running the combination of these modules forms the gait of the robot. There are many combinations for motion sequencing and a few combinations with different connections between the various modules were tested. Since a quadruped's static walking gait is rather fixed, the order of movement was fixed in the first trial. The leg avoided obstacle by moving back a little and then up and forward. If there was no way to clear the obstacle, the robot would drop the foot and move the robot to the side before attempting to move forward again.

The second version of the behaviour-based controller differs from the first version in terms of order of leg selection. During static walking, the centre of gravity (CG) must fall within the triangle formed by the three legs that support the move.

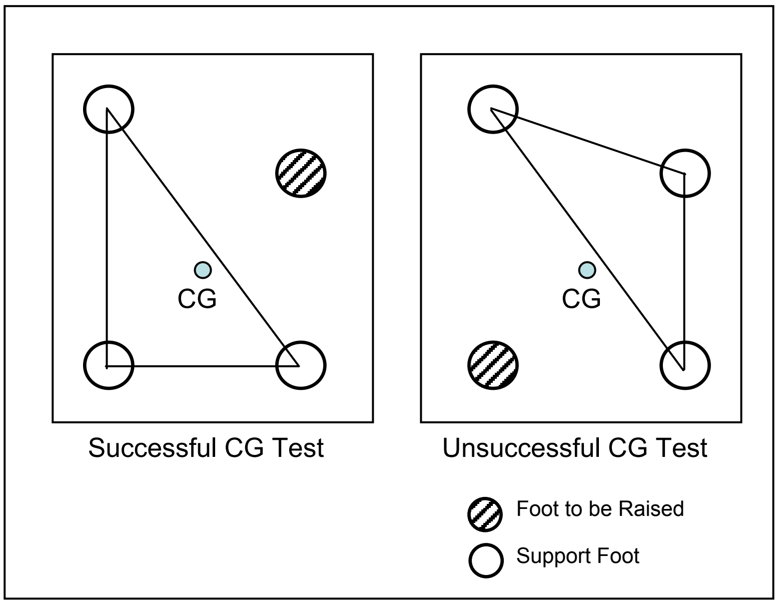


Figure 5.8 – CG test criterion

The new gait selection used a lookup table to find the next foot to move. A lookup table was formed containing the order of the leg movement for gaits used in the first version. The program checks the lookup table to get the next leg to move. Then it checks the CG before the foot is raised. The CG is fixed at the centre of the robot body, so as to simplify calculation. If the CG test is successful, that foot can be raised. If the CG test is unsuccessful, the program refers to the lookup table to obtain the next leg to move. The CG test criterion is shown in Figure 5.8. This process continues until all four legs move forward or it is not possible to move any more legs. The latter means the move is a failure. This can occur when a previously moved foot lowers before the specified position due to an obstacle. This disrupts the support stance of the robot causing failure of the gait. If all four legs successfully move forward, the body then advances forward to complete the move. This is a check and move approach. There are no measures taken when a move is not possible.

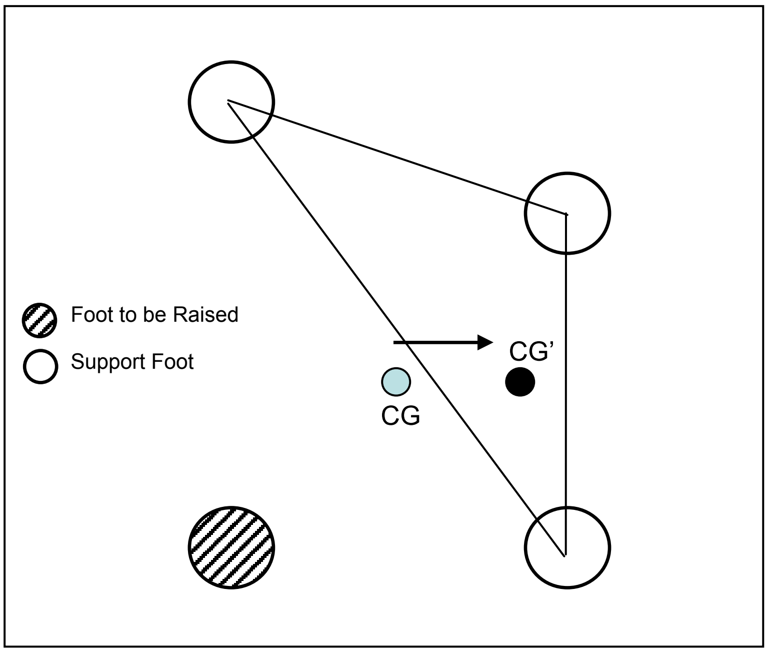


Figure 5.9 – Shifting the body so that the CG test is successful

The following describes a method used to solve the CG instability problem. During certain moves, the body of the robot has to shift either left or right of the move for the CG to fall within the stability triangle as shown in Figure 5.9. This method has been employed in both the walk left and walk right algorithm as the CG shifts from left to right with respect to the direction of move for the robot to complete the move. It was found that the body shifting is not required for the forward and backward move although movement in the left and right direction was found to be more stable.

5.7.2 Modules Available

A number of different modules were developed to achieve robust walking. The different modules developed will be explained briefly in this section.

The primary task of the robot controller is to coordinate walking in all directions. To walk the robot must raise its leg, move the leg and lower the leg. Therefore naturally the first modules to be developed are the *Raise Leg*, *Forward Leg* and *Lower Leg* modules. The robot was able to perform static walking using a simple gait pattern with the coordination of these modules. During static walking, at least three feet must be on the ground at any point in time. Therefore, only one leg can be lifted up at a time. This means that the *Raise Leg* module must inhibit the other *Raise Leg* module, to prevent multiple legs lifting. The order in which the legs move is programmed into the different module. This order is stored in the list that is access by the *Lower Leg* modules. Therefore the *Lower Leg* module will exhibit the next *Raise Leg* module according to the gait list. After all four legs have moved, the body of the robot has to move forward to complete the move. An *Advance Frame* module is added to move the body to complete the gait.

Other modules were added to aid the robot in moving. Basic movement modules have been added. They include:

- Move left/right/forward/back: Based on the gait list, the module moves the robot in the respective direction. It supervises the various *Raise Leg*, *Move Leg* and *Lower Leg* modules to complete the gait.
- Turn left/right: Turns the robot in the left/right direction x degrees around the centre of the robot.

Figure 5.10 shows module relationship activity for forward movement. The leg is first raised, then moved forward and finally lowered to complete the move. Every move is exhibited by the move before. Once the move is completed, the next *Raise Leg* module is exhibited. Note that the *Avoid Leg* module is exhibited by the *Forward Leg* module to avoid obstacles the leg encounters.

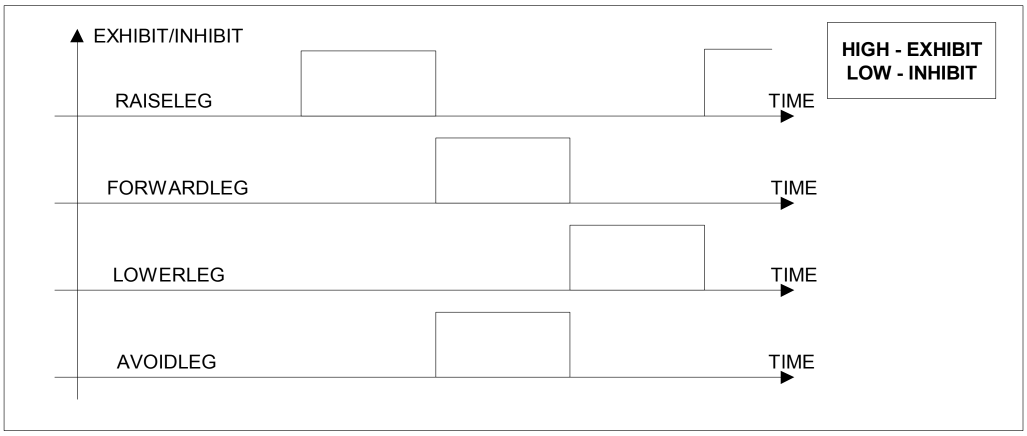


Figure 5.10 – Timeline of a typical Forward Leg move

These following modules were added to adjust the foot of the robot if it is sensed that the move is not possible. This improves the adaptability and hence the robustness of the robot controller.

- *Avoid*: When a *Move* module is called, the *Avoid* module is activated as well to monitor the proximity sensors on that particular feet, i.e. there is an obstacle in front. If a proximity sensor is triggered, the movement stops and the *Avoid Shift* module is exhibited.
- *Avoid Shift*: This module tries to clear the obstacle by shifting the leg back a little, lifting the leg up and forward again before exhibiting the respective *Move* module. If the *Move* module detects an obstacle again, the procedure is repeated.

- *Relocate Leg*: During *Lower Leg*, the force sensors on the foot finds foothold on the ground. If the robot does not find a foothold, the leg will be lifted and moved in a reversed and sequential order to find a foothold.

These algorithms will be explained in the following section.

5.7.3 Various Algorithms Towards Robust Walking

The algorithms developed help the robot clear a number of difficult situations. The controller deals with two situations – first with obstacles in front and secondly with holes in the path of the robot since there are two types of sensors that deal with these forms of obstacle clearing. The *Avoid Leg* module deals with the first situation whereas the *Relocate Leg* module deals with the next situations.

5.7.4 Avoidance Mechanism

The avoidance mechanism handles obstacles that the robot detects in the direction of motion. It is a repeated three step process. A description of this mechanism in cats – the placing reaction can be found in [McMahon 1984]. A blindfolded cat will lift its foot if it detects an object on the dorsum (top part) of the foot (Figure 5.11). The robot action of this move is shown in Figure 5.12.

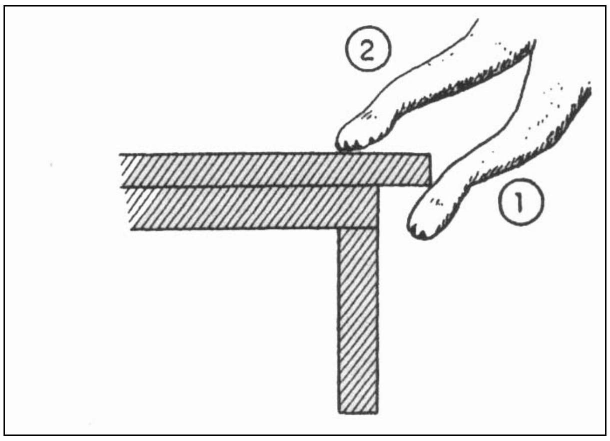


Figure 5.11 – The Placing Reaction

When a foot is moving forward or any direction in the horizontal plane, the Avoid module is activated to receive signal from the proximity sensors. If an obstacle is detected, the *Avoid* module exhibits the *Avoid Shift* module that conducts the three step algorithm to move the foot to clear the obstacle.

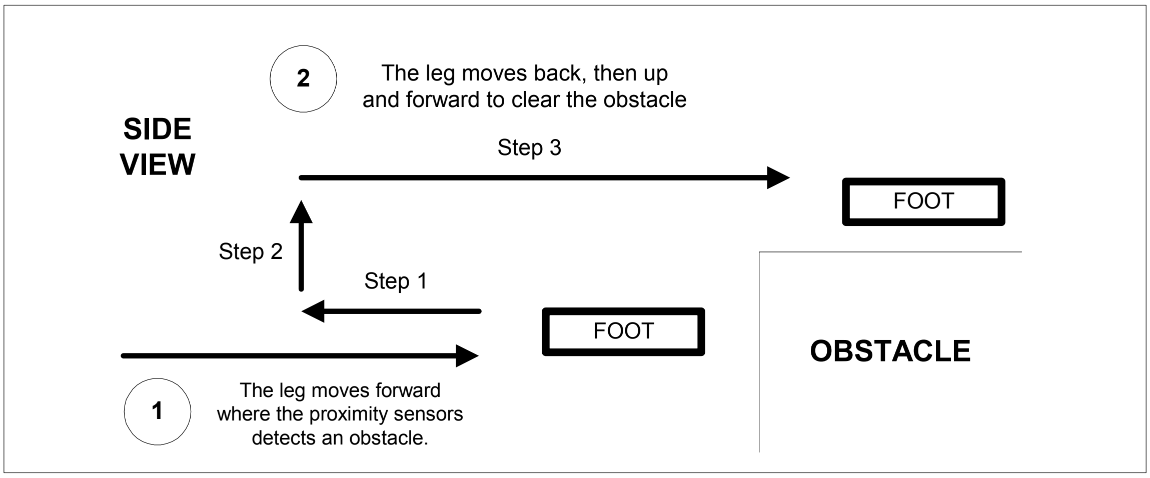

2
The leg moves back, then up and forward to clear the obstacle
SIDE VIEW
Step 3
FOOT
Step 2
Step 1
FOOT
OBSTACLE
1
The leg moves forward where the proximity sensors detects an obstacle.

Figure 5.12 – The Avoid Shift algorithm in action

The three steps involve the foot moving back a short distance, then up and then it tries to move forward to clear the obstacle as shown in Figure 5.12. This action is repeated until either the obstacle is cleared or a threshold distance is reached, i.e. the obstacle cannot be cleared. If the obstacle cannot be cleared, an alternative action must be taken. The robot switches the direction of move. Figure 5.13 shows the sequence of events for a forward leg movement. In the figure the *Avoid* module is activated by all *Move Leg* modules before the move starts. During the move, an obstacle is encountered and the obstacle is cleared by the *Avoid Shift* module.

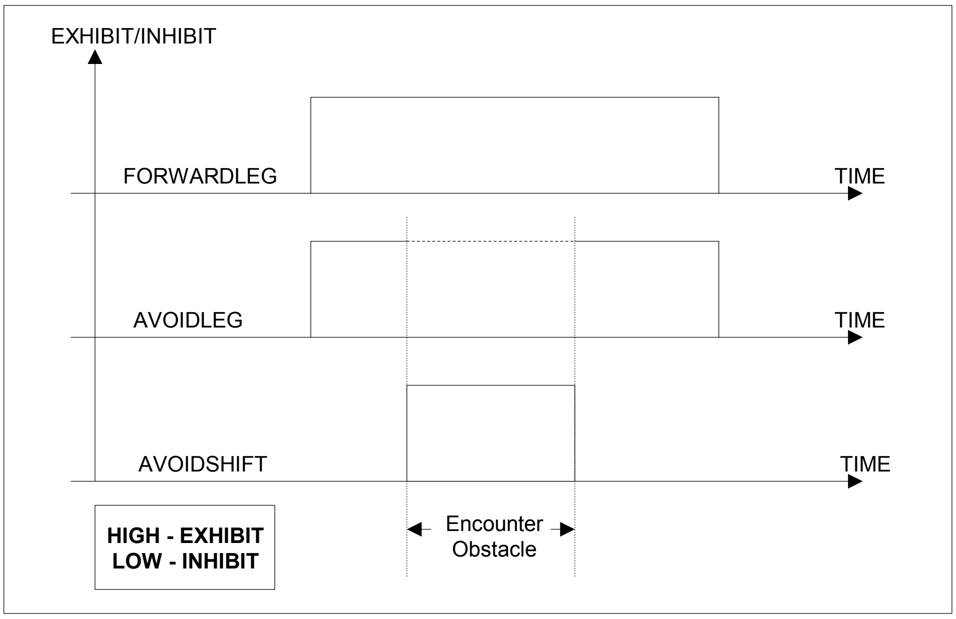


Figure 5.13 – The sequence of events for the Avoid module

5.7.4.1 Relocate Leg

This algorithm allows the robot to deal with situations like potholes and irregularity of the ground. This movement is similar to when the stick insect moves its foot around to find foothold.

During the *Lower Leg* module, the pressure sensor on the foot detects contact and stops moving down when the pressure sensor is activated. A threshold distance in the z-axis is set upon which if the pressure sensor is not activated, the alternative action starts upon which the *Lower Leg* module exhibits the *Relocate Leg* module.

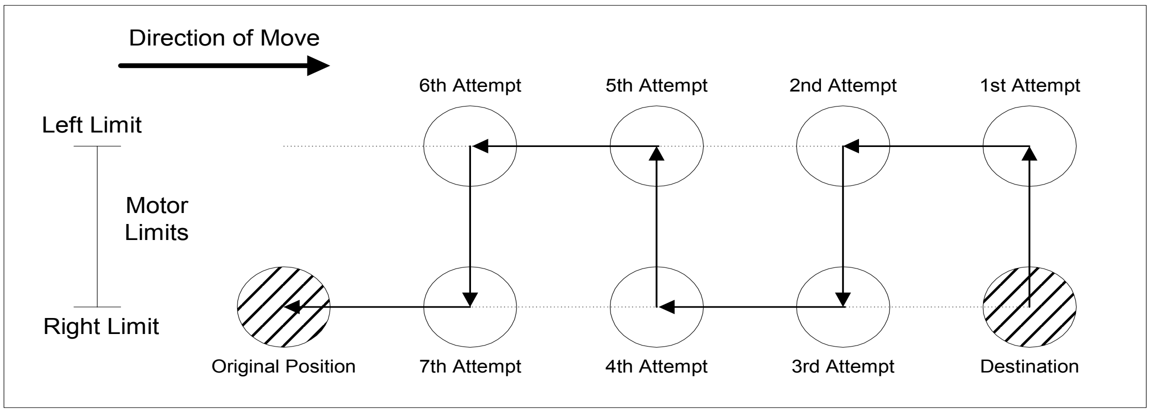


Figure 5.14 – The Relocate Leg Algorithm

The role of the *Relocate Leg* module is to find a foothold for the foot in the vicinity of the target destination of the foot. The algorithm is shown in Figure 5.14. Assuming that the foot is always at the extreme end (either left or right), the algorithm starts by moving the foot in the other extreme direction where it lowers the foot to test the foothold. If it fails to find a foothold, the foot moves a distance behind and tries to probe in both extremes. This test is repeated until either the foot finds a foothold or the foot moves back to the original position.

This algorithm requires the coordination between three modules – the *Lower Leg*, the *Step Back* and the *Relocate Leg* module. The *Lower Leg* module was modified to take into consideration the algorithm. It counts the number of time the *Lower Leg* module is exhibited in the WAIT state. If the occurrence is an odd number, the *Relocate Leg* module is exhibited otherwise the *Step Back* module is exhibited. The flowcharts of the three modules are shown in Figure 5.15. The timeline is shown in Figure 5.16. The time for each action and between each action are variables and can change depending on the situation.

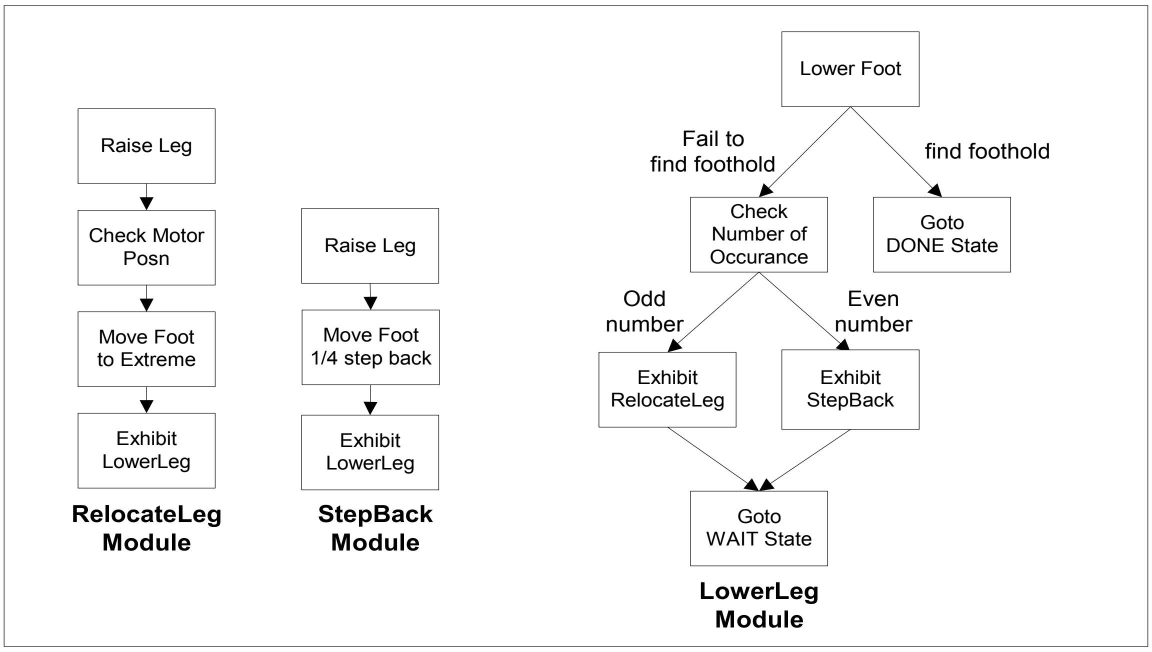


Figure 5.15 - Flowcharts of the three modules involved in the relocate leg algorithm

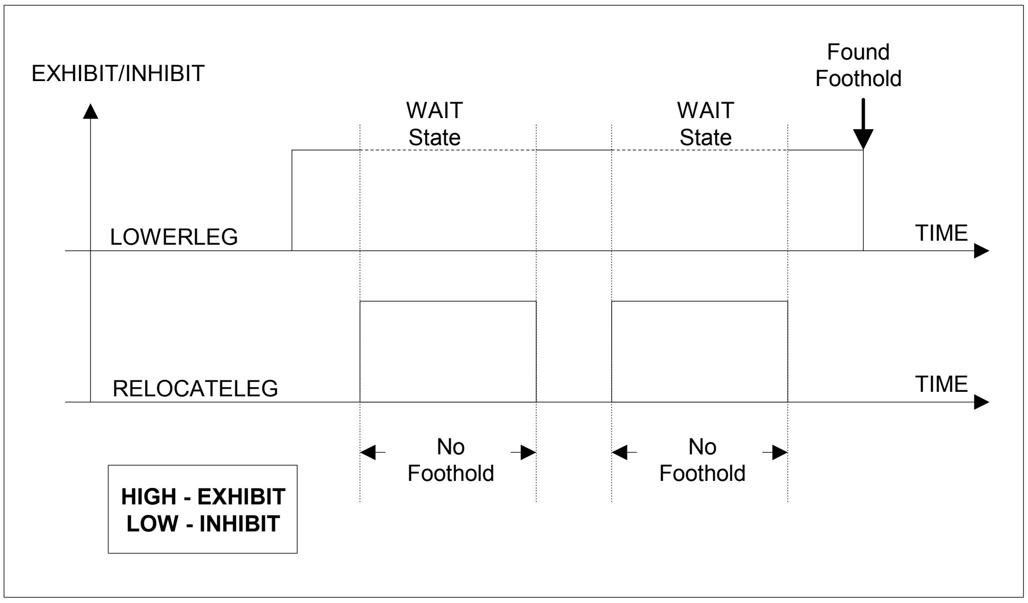


Figure 5.16 – Timeline of the Relocate Leg module

5.8 Observations and Results

The behaviour-based controller performs stable sustained static walking gaits. At present, the robot does not have navigation capabilities or the ability to move to specific goal. The robot is able to move without human intervention once the direction of move is specified. The robot was able to clear obstacles in its path, executing both the *Avoid Leg* and *Relocate Leg* modules.

In this project the following goals have been achieved:

1) a modular control system was created,
2) a schema for creating behaviour modules was created, and
3) a simple robust behaviour-based controller with some avoidance mechanisms was built.

In the early stages of development, a basic group of modules were created to produce simple walking without sensing. Once simple walking was achieved, various modules were created from walking in various directions, and turning the robot. In the initial stage, the gait of the robot was fixed and no checks were conducted to ensure the stability of the robot. The CG test function and the Gait selection function improved the autonomy and flexibility of the controller. Since motion stability is under assessment, time taken to move forward is not a major concern. Hence, the robot has ample time to check and adjust to ensure the stability of the robot is always maintained. Figure 5.17 and 5.18 show the encoder counts of the different axes of the robot for a move right behaviour. The robot was able to perform this stable gait for a prolonged period of time even with disturbance in the system.

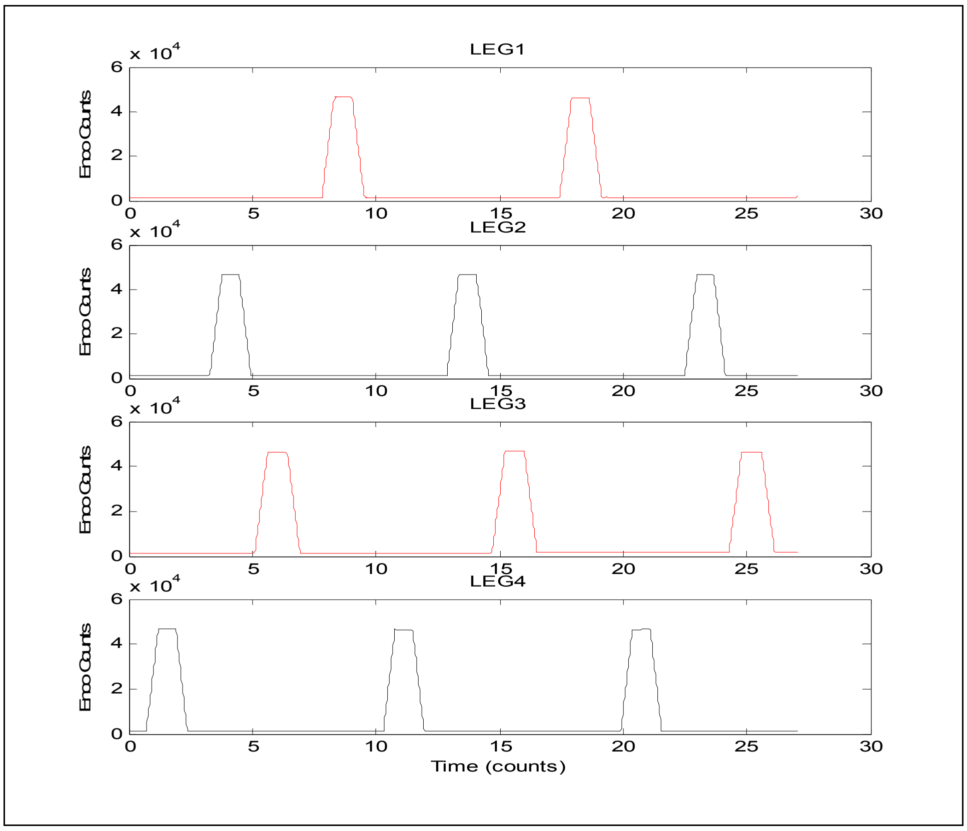


Figure 5.17 – Z–axis encoder readings for 4 legs for walk right gait

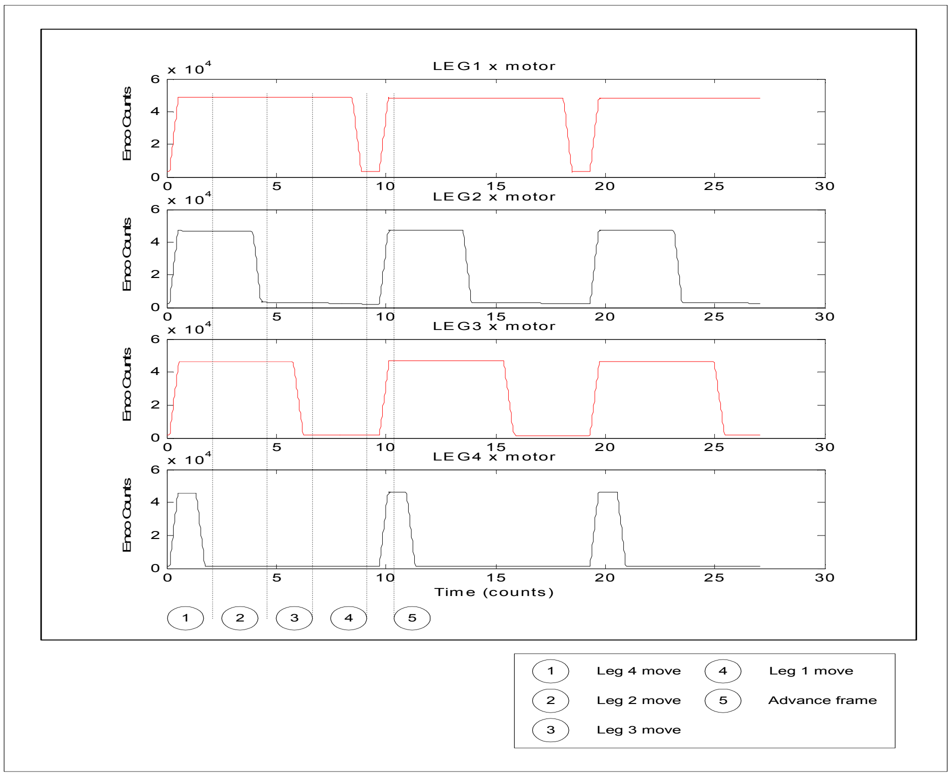


Figure 5.18 - Movement of X axis motors as the robot walks right

The robot performed the relocate leg move when the force sensor was not triggered. The robot could move the leg back as described to find a foothold. The controller was able to coordinate between the different modules. The motion of the robot was not affected by the algorithm. This is aligned with the module independency property of the behaviour-based controller.

The controller's avoid shift algorithm was able to try and clear the obstacle. When the foot is in the raise leg position, the z-axis distance to threshold is small. Therefore, the controller cannot move the foot up too much. Hence, the algorithm is called a few times before threshold is reached.

The algorithms were useful in clearing the various obstacles and difficulties that the robot faced while moving autonomously in an unknown and uneven terrain. As the controller is an incremental modular controller, various modules can be easily added for improvement.

5.9 Summary

The behaviour-based approach is a hybrid controller that is based on both the planner based controller and the reactive based controller. It has the computational efficiency of the reactive based approach while maintaining some perks of planner based approach.

The concept of a behaviour-based controller was explained. It has the following properties,

1) each behaviour is an independent sense-plan-act module,
2) all behaviours work in parallel to achieve a common goal, and
3) a combination of these modules and their connections defines the system.

There are various problems associated with simultaneously active modules. Firstly, there might be conflicts as the different modules share common resources and control the same actuators. Therefore, a method to resolve these differences is required. Various methods were introduced which included Brooks' Subsumption Architecture. Brooks' method was selected to solve those conflicts.

The behaviour-based approach is a modular based system and, therefore, encounters problems with system states and information sharing. Problems arise due to the lack of knowledge of the system module states. The blackboard model is a method implemented to coordinate sharing of information between the various modules in the system.

Lastly, the behaviour-based controller was implemented for simple robust walking. This controller is based on Brooks' Subsumption Architecture and Wettergreen's research on behaviour-based controllers. Various modules were developed and the interaction between these modules managed to achieve robust walking. Various functions were included to improve the autonomy and flexibility of the controller. Three algorithms were also developed to improve the robustness of the controller.

Chapter 6

CENTRAL PATTERN GENERATOR

6.1 Introduction

In the field of robotics, there has always been a fascination with biological systems. Evolution has created creatures that adapt to its environment and terrain perfectly. It is amazing how legged creature locomotion can be so agile and efficient. Hence, the curiosity of man [Farquharson] has led to the study of these biological locomotion systems and the development of biologically inspired robots.

Biologically-inspired robotics aims to study both the biomechanics and neurophysiology of the animals and apply these findings to robotic systems. Research in this area is achieved by studying both vertebrate and invertebrate with relatively simple neural constructs to comprehend their various biological mechanisms. A number of robot controllers were built based on these simple neural systems. The aim of these studies is to achieve controllers that require simple communication with minimal computation. These controllers have even been implemented in simple analog VLSI models.

There are many areas that can be explored in the biological aspect but locomotion is specifically dealt with in this project. Grillner found that there is a mechanism within the spinal system of animals that controls locomotion without connections to the brain

[Grillner 1975]. This system was referred as the Central Pattern Generator (CPG). The CPG generates pattern signals required for muscular control at the limbs to produce legged locomotion. This finding led to research in locomotion, which aims to create a stable sustainable pattern generating system that adheres to the CPG model. In this project, a CPG controller is developed as a locomotion controller for the NUS quadruped.

6.2 Central Pattern Generators in Legged Animals

6.2.1 The Neurological System – Studying Pattern Generation

A motor program is "a set of muscle commands that are structured before a movement begins and can be sent to the muscle with the correct timing so the entire sequence is carried out in the absence of peripheral feedback" ([Keele 1968]). There are many examples of these biological pattern generators in biological systems that are common to most animals. Other systems include the respiratory system, the chewing system, and even the escape behaviour of preys in the wild [Pearson 1993]. Locomotion in animals is a form of a pattern generator that controls motor action. An interest in this study has sparked off the research in the mechanics and behaviour of the locomotion system in legged creatures.

6.2.2 CPG - Neural Studies on Legged Locomotion

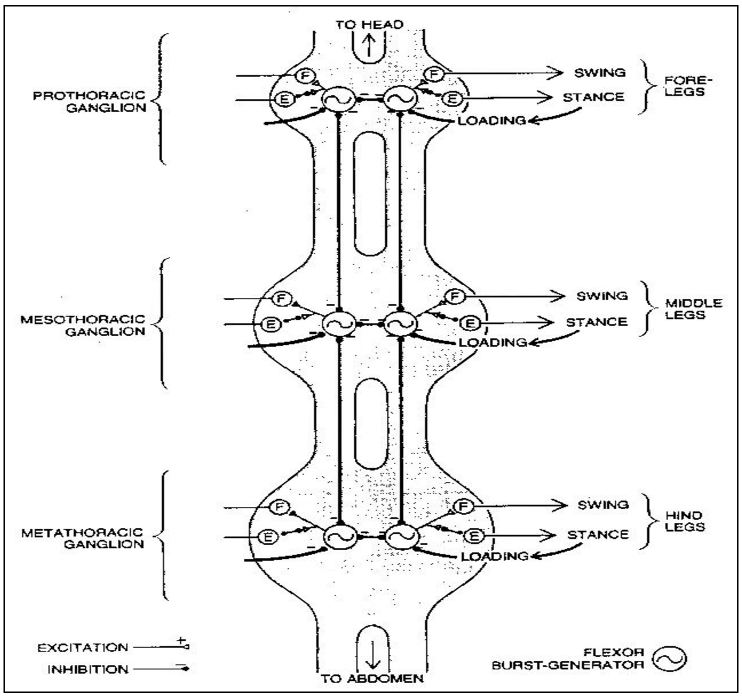


Figure 6.1 – The locomotion neural system of a six legged animal [Grillner 1975]

Experiments were conducted on cats where their spinal cords were severed from their brains [Grillner 1975] to investigate the fact that locomotion is not the responsibility of the brain. These finding showed that these cats were able to generate a walking gait with avoidance and reaction mechanism without information from their brains. This led to the hypothesis that there exists a mechanism within the spinal neural system that coordinates locomotion in legged animals. The spinal cord generates signals that coordinate muscle action that produces locomotion. This mechanism is known as the CPG.

The neural system for legged locomotion is shown in Figure 6.1. The smallest unit of the system consists of a joint, the extensor and flexor muscles. The α motor neuron carries signal from the brain to the muscles. Each muscle receives control signals from the α motor neuron at the spinal cord (Figure 6.2). All this activity is independent of the brain, i.e. walking is autonomously generated by the neural system in the spinal cord.

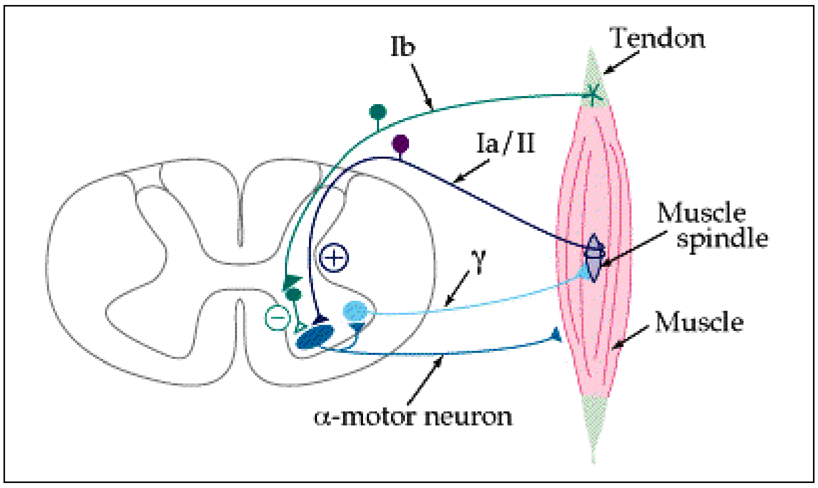


Figure 6.2 –The spinal/CPG connections with muscles[1].

Studies on the CPG have shown that it is a complex system of neural networks that makes the system difficult to model. Most of the connections found within the network were found to be inhibitory. But implementations of CPG controller showed that CPG models do not require such complicated networks to be realised.

6.3 Various CPG Control Approaches

Over the years, this idea has been adopted into implementing controllers for legged robots. The idea is to produce a periodic signal in a controlled manner, which is in turn used for producing the gait pattern. Most CPG controllers control the gait by modifying the gait parameters. The stability of the gait can be improved by feedback

[1] http://thalamus.wustl.edu/course/spinal.html

from the sensory system. Hence, the robot is able to react and change both the speed of locomotion and the type of gait.

There are many methods of implementing the CPG. These include using,

- coupled non-linear oscillatory equations that is based on neurological system,
- mathematical models ([Venkataraman 1996]),
- fuzzy-like rules to achieve this oscillatory movement, or
- neural networks and/or evolutionary methods ([Cruse 1994], [Ijspeert 2001], [Kodjabachian 1995]).

Sensory-Motor Integration

Locomotion requires information including auditory, visual, and other environmental information. Both the motor and sensory systems work together to provide motor information for effective and efficient movement. The study on animal's motor and sensory system has surfaced some intriguing facts. It is crucial that the two systems work together. This integration work with the CPG controller will be elaborated in the following sections.

6.3.1 Cruse's Rule Based Approach

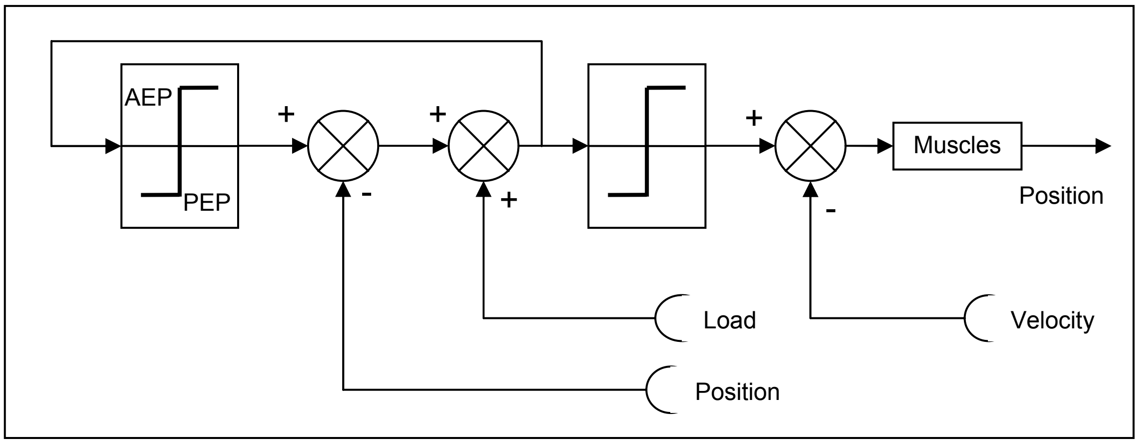


Figure 6.3 – Circuit diagram of the movement of a single leg [Cruse 1991].

Cruse [Cruse 1991] modelled the stick insect to obtain its gait. Using the biological data obtained from the stick insect, he studied the mechanism that coordinates the movement of different legs in order to produce a walking gait. The results of his research showed that a single central pacemaker does not produce the coordination between the legs. Instead, each leg has a local controller and the gait emerges out of the interaction of these local controllers according to a set of simple rules.

Figure 6.3 shows a circuit model of the local leg controller. There are two states for each leg in legged motion, the return stroke and the power stroke. The return stroke involves moving the leg from the anterior extreme position (AEP) to the posterior extreme position (PEP), while the foot is on the ground. And the power stroke moves in the opposite direction with the foot off the ground. Hence, the return stroke is the action that moves the body forward. The different leg positions are shown in Figure 6.4.

There are two parts to the circuit, the first decides on which of the two states the leg is to be and the second consists of a velocity feedback system. The first part of the circuit takes into account the current load and position of the leg before making its decision.

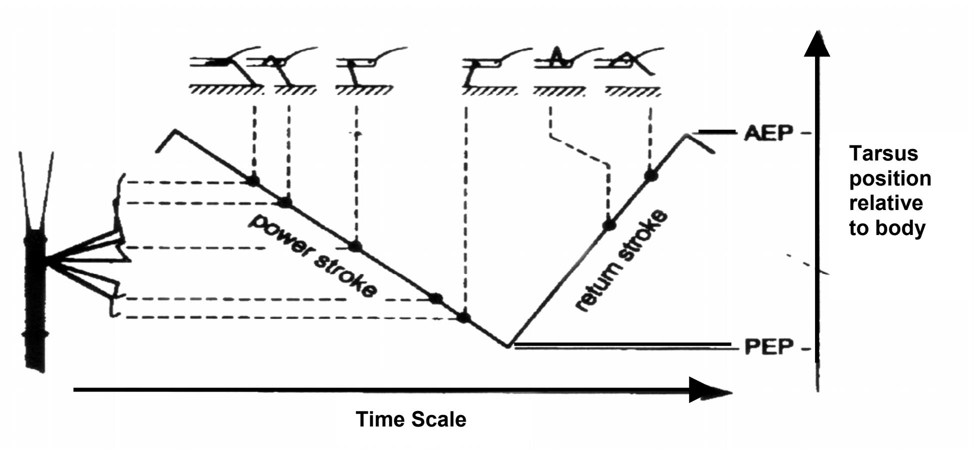
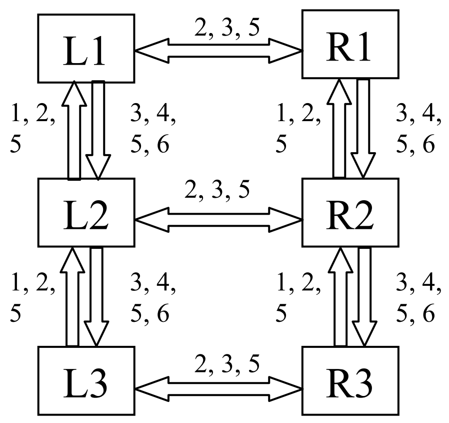


Figure 6.4 – The Cyclic movement of a leg [Cruse 1991].

Cruse used a set of rules to implement the CPG. The activation of individual legs is dependent on a set of rules. There are two classes of these rules: one being the ipsilateral (same side) leg relationships and the other being contralateral (opposite side) leg relationships. There are a total of six simple relationship rules, which are presented in Figure 6.5.



1. Hinders a leg from starting a return stroke as long as the posterior leg is executing a return stroke.
2. Start a return stroke in the leg when the posterior starts the power stroke.
3. Caudal positions excite start of return stroke.
4. Anterior leg position influences position at the end of the return stroke of the posterior leg.
5. Increase in motor output when there is a decrease in speed of the leg in the power stroke.
6. Treading-on-tarsus reflex cause the posterior leg to lift and be placed slightly to the back.

Figure 6.5 – Cruse's rules relationship between legs for a six legged robot [Cruse 1991].

Cruse implemented this set of rules in a neural network ([Cruse 1995], [Cruse 1998]). In his implementation, each leg was controlled by a local sub-network. Each of these control sub-networks consists of three subnets. The first generates the return stroke while the other generates the power. The last subnet controls the previous two subnets to ensure the leg perform the correct action. The gait emerges from the interaction between the different control sub-networks of the different legs. The interaction between the different legs is governed by the rules shown in Figure 6.5.

6.3.2 Neural Networks and Genetic Algorithm

Since the CPG is a biological controller, it is only natural that it is a biologically-derived system. Neural network and genetic algorithm fit this bill perfectly. Neural networks have been used to produce CPG controllers ([Srinivasan 1992], [Chiel 1999], [Beer 1999], [Astma 1999]). Robust control can be attained with neural network CPG controllers ([Hillel 1992], [Kimura 2000]). Genetic algorithms were integrated to allow systems to evolve over time. Genetic algorithm has been used in many cases for optimising solutions. It is used together with neural networks for find optimum number of neurons and their connections. The lamprey's evolution approach [Ijspeert 1999] in the next section will show this work involved in CPG models.

6.3.2.1 Evolution Approach Using Neural Network

Ijspeert and Kodjabachians' study on the lamprey has led to the development of an evolutionary approach to neural network to solve the CPG problem [Ijspeert 1999]. This method uses a genetic programming algorithm to evolve developmental programs which encode a growing dynamic neural network. It uses a genetic programming approach with a developmental encoding known as Simple Geometry Oriented Cellular Encoding (SGOCE). This architecture evolves to control high level characteristics like the speed of locomotion and a change in direction. The aim of this approach is to obtain an automatic generation of a control mechanism for locomotion. This controller imitates the natural process of evolution, allowing the neural network to create its own synaptic connections and the amount of neuron in the network in a systematic way.

This method uses a control structure similar to Cruse's method. Each limp is controlled by a local controller. The coordination of all the controllers determines the gait of the system in control. This method has been used as a controller for a virtual six legged insect with behaviours such as 'gradient following' and 'obstacle avoidance'.

Genetic programming has always been used for optimisation of neural networks. This evolution system gives this implementation the ability to achieve its specified task without having to tune the controller. The only problem with this method is that it takes a while to create the neural network (e.g., approximately 450 CPU hours on an Ultra 1 Model 140s SUN station).

6.3.2.2 Coupled Neural Pattern Generators

The last method discussed here uses coupled neural oscillators to implement the CPG. Through carefully selected parameters and coefficient of the networks, the system is able to exhibit a signal pattern. Studies in oscillatory control include Barnes' work on pulse coupled relaxation oscillators [Barnes 1998] and J.S. Bay and H Hemamis' work on van der Pol oscillators [Bay 1987]. The aim of these experiments is to obtain a stable pattern of oscillation, which can be used to model the CPG of legged animals. The van der Pol oscillators are used to develop a controller for this project.

Coupled van der Pol Oscillators

This model mimics the neurological system of animals. The control outputs either expand or contract the different muscles that are involved in gait generation. Similar to most neural oscillatory models, this model has a group of oscillators that are coupled together. Bay and Hemami found that the simulated output of the system is able to produce a human walking gait [Bay 1987].

This is a network of neurons built with sets of neurons grouped to become individual oscillators. Each oscillator obeys a set of van der Pol equations for manipulation of data. Every oscillator has a set of parameters – phase, frequency and amplitude. Hence, modification of these parameters allows the alteration of gait and its speed.

From the simulation of the model with various sets of parameters, the results show that oscillation for various gaits is achievable. Variations in the gait are obtained by,

i. modifying the amplitude of the signal (i.e., contraction and expansion of muscles as with animals),
ii. modifying frequency for faster and slower gait,
iii. shifting phase between legs from 0 to 180 degrees for transitions between gaits,
iv. using travelling waves as seen in the motion of fishes, and
v. having the ability to inhibit other oscillators in the network from generating motion.

Pribe, Grossberg and Cohen's Oscillator Mode

This CPG model uses physiological mechanisms, such as nerve cell models that obey Hodgkins-Huxley type equations ([Grossberg 1997], [Grossberg 1997a]). A neural network is used to create the model since the model is to be neurologically based. Gait patterns are generated from the interaction between neurons and the control of the parameters that control the interaction and the input to the system.

Yamanishi (1980) has conducted a bimanual finger tapping experiment to find out how rapidly joint combinations are bounded to rhythmic patterns. A human subject is asked to tap keys in time to visual cues across ten relative phases from 0 degree to 360 degrees. During this experiment, the phase difference between the visual cue and the human tapping is measured. The standard deviation of the error for locomotion is lowest when the phase relationship is either near in-phase or pure anti-phase. This means the oscillatory behaviour of the human is biased in favour of in-phase and anti-phase relationship.

Therefore, this in-phase and anti-phase relationships are applied to the inter-limb relationship of the robot. This has produced various stable gaits in simulation.

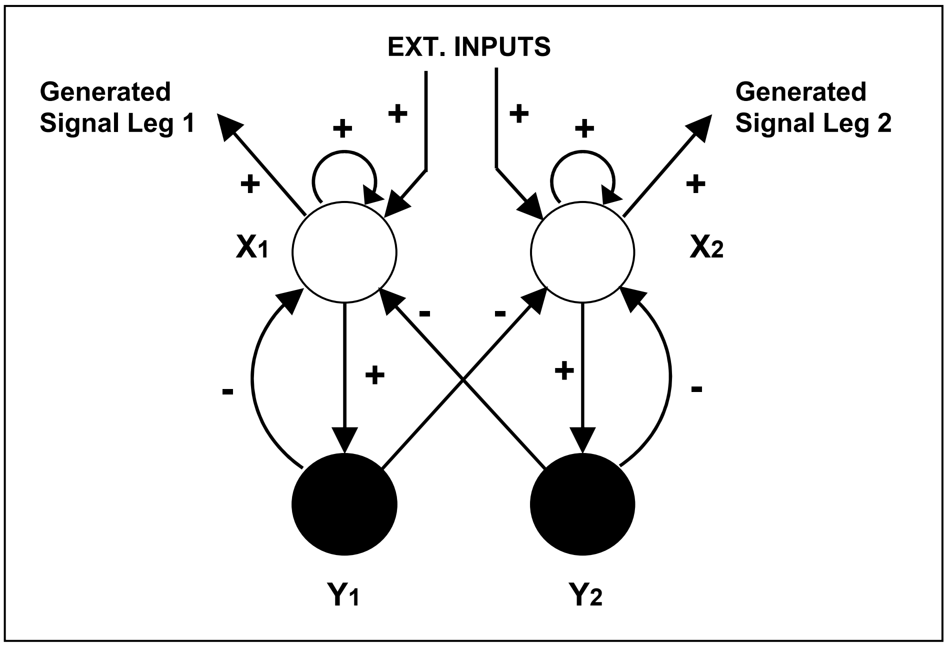


Figure 6.6 - Two channel oscillator of the CPG developed for the robot [Grossberg 1997].

Building the CPG oscillator

The CPG model is a modified version of the Ellias-Grossberg oscillator model. This is a neural model (also known as a cell) where two neurons are used, an excitatory neuron and an inhibitory neuron. Since legged locomotion can only happen with even number of legs, two cells (one cell per leg) are shown in the Figure 6.6. Each cell can be referred to as ‘a channel’. The neurons obey Hodgkin-Huxley type equations where excitatory signals operate faster than the inhibitory signal in a recurrent on-centre off-surround anatomy. Unlike the typical Ellias-Grossberg model where only the excitatory signals are coupled to the membrane equation, in this case, both inhibitory and excitatory signals are coupled to the membrane equation. The majority of the connections are inhibitory connections as these signals govern the characteristics of the cell.

Equations for the Neurons

The CPG model obeys the following Hodgkin-Huxley equations,

$$\frac{d}{dt}x_i = -Ax_i + (B - x_i)[f(x_i) + I_i] - (C + x_i)\sum_j D_{ij}g(y_j) \tag{6.1}$$

and

$$\frac{d}{dt}y_i = -E[(1 - y_i)[x_i]^+ - y_i] \tag{6.2}$$

where

$$[w]^+ = \max(w,0) \tag{6.3}$$

and

$$f(w) = \frac{F_1([w]^+)^2}{F_2 + ([w]^+)^2}\ ,\ g(w) = \frac{G_1([w]^+)^2}{G_2 + ([w]^+)^2} \tag{6.4}$$

Comparing the above equations with the actual Hodgkin-Huxley equations, it can be seen that Equations (6.1) and (6.2) can be rewritten into the following Huxley-Hodgkin notation into Equations (6.5) and (6.6) respectively,

$$C\frac{\partial V}{\partial t} = (V^P - V)g^P + (V^+ - V)g^+ + (V^- - V)g^- \qquad \textbf{(6.5)}$$

Represent	**Equation (6.1) represented as in Equation (6.5)**
Variable voltage	V = x_i
Constant saturation voltages	$V^P = 0$, $V^+ = B$, and $V^- = -C$
Conductances	$g^P = A$, $g^+ = f(x_i)$, and $g^- = D_{ij}g(y_i)$

Table 6.1 – Comparison of Equation (6.1) and Equation (6.4)

$$\frac{d}{dt}y_i = \alpha(x_i)[\beta(x_i) - y_i] \qquad \textbf{(6.6)}$$

where $\alpha(x_i) = 1 + [x_i]^+$ and $\beta(x_i) = \frac{[x_i]^+}{(1 + [x_i]^+)}$

In equation (6.1), $f(x_i)$ acts as the fast excitatory conductance, and $g(x_i)$ as the slow inhibitory conductance. x_i measures the activity of the excitatory neuron ,and y_i the activity of the inhibitory inter-neuron. y_i controls a slow inhibitory intracellular conductance rather than a separate inhibitory inter-neuron. Both the inhibitory and excitatory signals are rectified sigmoid (shown in equation (6.5)).

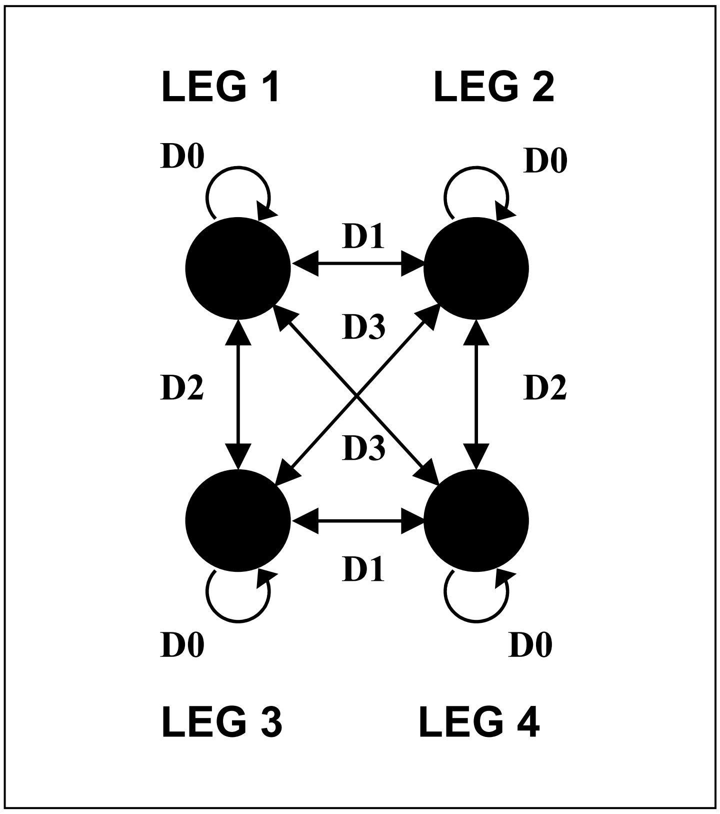


Figure 6.7 – Inhibitory strength diagram for four cells representing four legs

Inhibitory term	**Inhibitory Strength Representation**
D0	Self-inhibitory
D1	Front-Front, Rear-Rear
D2	Front-Rear and Rear-Front ipsilateral
D3	Front-Rear and Rear-Front contralateral

Table 6.2 – Inhibitory strength representation

The excitatory signal excites itself but the inhibitory signal acts on all cells depending on the strength D_{ij}**.** The list of all the inhibitory strength is shown in Table 6.2 and a schematic diagram is shown in Figure 6.7. I_i is the input signal to the system. Equation (6.6) shows that the conductance y_i is controlled by $\alpha(x_i)$ and approaches an asymptote $\beta(x_i)$. The two terms are dependent on the variable voltage term, x_i. (B - x_i) and (C + x_i) in equation (6.1) are shunting terms and (1 – y_i) is a shunting term in equation (6.2) that is not present in the original Ellias-Grossberg's set of equations.

The gait has a tendency of switching from an anti-phase relationship to an in-phase relationship as the oscillating frequency increases. The oscillatory signal is dependent on the input signal. A change in the inhibitory cross-coupling strengths $\mathbf{D}_{ij}$ together with an increase in the self-inhibitory strength $\mathbf{D}_{ii}$ affects the system phase relationship as well.

A signal is used to not only activate the gait but trigger gait transition by modification of the amplitude and the phase difference of the signal. By generating different in-phase and anti-phase signals between the different neurons (i.e. legs), different gaits at different speed can be achieved. Altering the input **I** signal with the different parameter in various ranges, and modulation of the inhibitory signals allows alteration of the gait online. Therefore the controller is able to switch quickly between gaits.

6.4 Advantages and Disadvantages

Man marvel at the beauty of animals running around in the wild. Different animals walk with different gait, each with its own style. Wettergreen states that animals evolve to a locomotion that suits their need but may not necessarily be efficient [Wettergreen 1995]. Animals are able to fall down, roll over, trip, and get up again. But the mechanism and electronics on the robot are delicate and will damage if the robot falls over. Therefore animals do not necessarily make the best models for legged robots. Although the robot lacks the mobility and agility of the animal, it can achieve more efficient movement. Robots still lack the computational and sensory processing capability of animals. Therefore direct adaptation of animal features to a robotic system is not likely to work.

The neurological systems of simple creatures are still impossible to fully comprehend. At this point in time, it is still impossible to say the neural networks that replicate the networks of creatures are accurate and perform the same way as the actual thing. Beer found that extensive revisions on the basic walking network are needed to coordinate stopping, starting, and turning [Beer 1998]. Simple tasks as such require fine tuning over 500 parameters by trial and error.

Animals use CPG to produce rhythmic gait motion. This may be a problem for robots as these systems are difficult to tune and verify. Tuning by entrainment requires human guidance and interaction in teaching/tuning. It offers the best hope for robots, and this is also the method by which animals learn how to walk. It also has been shown that CPG controllers cannot produce consistent slow gaits, while animals can. Therefore there are a lot of problems associated with developing a robotic system based on biological systems.

The advantage of implementing a naturally inspired system like the CPG controller does outweigh the mentioned disadvantages, as this is a nature-inspired controller, evolved and optimized to perform locomotion of any sort. At this point in time, engineered robots can never outperform an animal in handling natural terrain. If a successful CPG controller is implemented with an adequate robotic body structure, it will bring forth the reality of an animal-like robot. The disadvantages can be treated as mountains that have yet to be conquered.

6.5 Proposed Approach

There are various methods used to develop the CPG controller, each having its advantages. But the neural model developed by Pribe, Grossberg and Cohen was chosen for the development of our CPG controller. This method generates a pattern that can be modified by changing the various parameters used. This signal is passed to an interpreter that generates the corresponding motor command. The signal generated is meant for muscle-like actuators. This means that the output signals are binary representing either the expansion or contraction of muscles. This expansion and contraction of muscles move the different joints thereby performing locomotion. Since the NUS quadruped does not have muscle-like actuators, the binary signal represents the two strokes, the return stroke and the power stroke.

6.5.1 Oscillator Signal Generator Model

The Pribe, Grossberg and Cohen's CPG model is used in the development of the CPG controller. This model is used to produce oscillatory signals for gait generation. Movement of each leg is divided into two phases, the power stroke and the return stroke. The power stroke moves the feet from the anterior position to the posterior position while the foot is on the ground. On the other hand, the return stroke moves the feet from the posterior position to the anterior position while the foot is above the ground. Figure 6.4 shows a cycle from power stroke to return stroke.

The model for the four-legged robot is the four channel neural simulator shown in Figure 6.8. Each channel represents a leg and has an excitatory and an inhibitory neuron. All equations are described in the section above (Equations for the Neurons). All excitatory neurons are inhibited by all inhibitory neurons in the model and are gated by the cross-coupling and self-inhibitory strength, $\mathbf{D}_{ij}$ and $\mathbf{D}_{ii}$ respectively. Modulation of the gait is obtained from changing the parameters of the equations used (as described in the section above, Equations for the Neurons) and the parameters of the inputs. Each gait is identified by its own set of parameters, with the values of the parameters obtained from observing the simulations.

The outputs from the Y-neurons are the inhibitory signal and used as feedback signal, while the outputs from all X-neurons are excitatory signals and are used as signal to the four legs for gait generation.

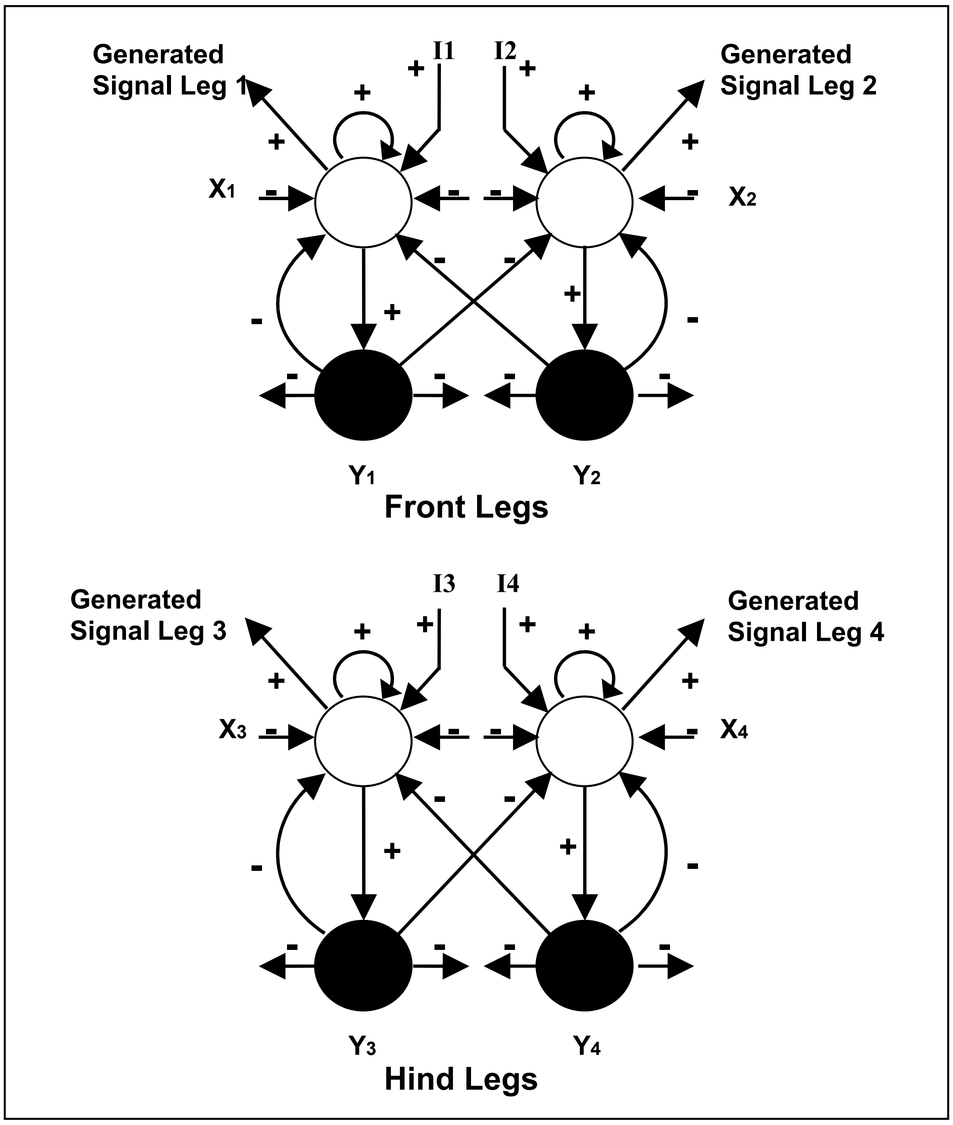


Figure 6.8 –A four legged neuron model of the CPG.

Inhibitory term	Inhibitory Strength Representation
D0	0.95
D1	0.255
D2	0.3
D3	0.3

Table 6.3 – Inhibitory Strengths used in the program

6.5.2 Runga-Kutta Method for solving ODEs

As a note, the Huxley-Hodgkins equations used in the oscillatory generator are a set of nonlinear Ordinary Differential Equations (ODEs). These equations have to be solved by an approximation method since there is no other method that solves these equations trivially so as to reduce the computational time and effort. Therefore, these ODEs are solved using a fourth-order Runga-Kutta method. This method can be found in most mathematical textbooks available. The C functions used to solve the equations are obtained from this program – "Numerical Recipes in C" [Numerical Recipe].

6.6 The CPG Program

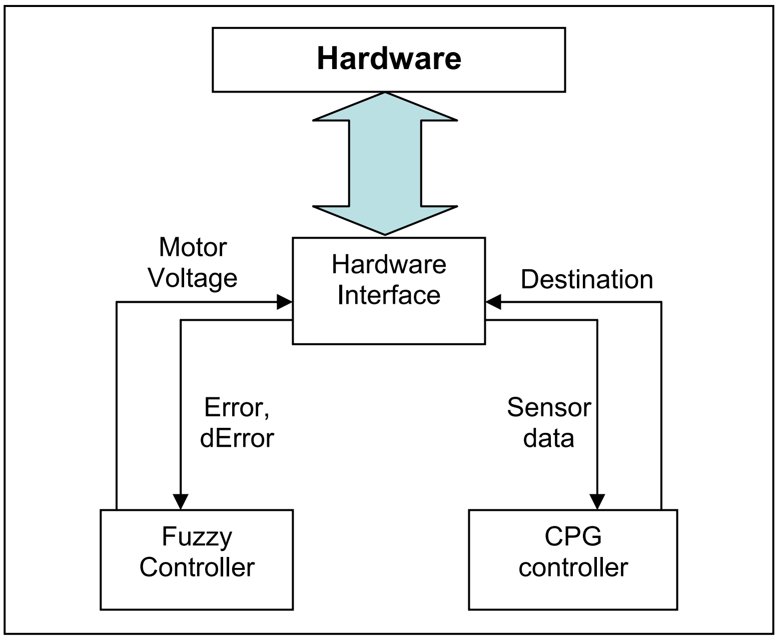


Figure 6.9 – Block diagram of entire CPG controller program

This program is developed reusing the two low-level control modules developed for the behaviour-based controller. And both the hardware interface and fuzzy controller part of the program are reused in the CPG controller. The block diagram of the controller is shown in Figure 6.9 and is a similar program structure with the behaviour-based controller replaced.

The CPG controller is divided into two modules. The first module generates the rhythmic signal (Signal Generator) while the other translates this signal to motor commands (Motor Command Interpreter). The Pseudo Code is shown in Listing 6.1. he signal is generated using the Pribe, Grossberg and Cohen's CPG model. The Motor Command Interpreter receives signal, interprets it and passes the information to the low level controller.

```
Signal Generator:
      Input parameters, step function;
      Calculate Derivatives using Runga-Kutta
      Obtain gait signal for Four Legs
      Stored gait signal in Circular Buffer
Motor Command Interpreter:
      Input gait signal;
      Generate joint motion command
```

Listing 6.1 – Pseudo code for Signal Generator and Motor Command Interpreter.

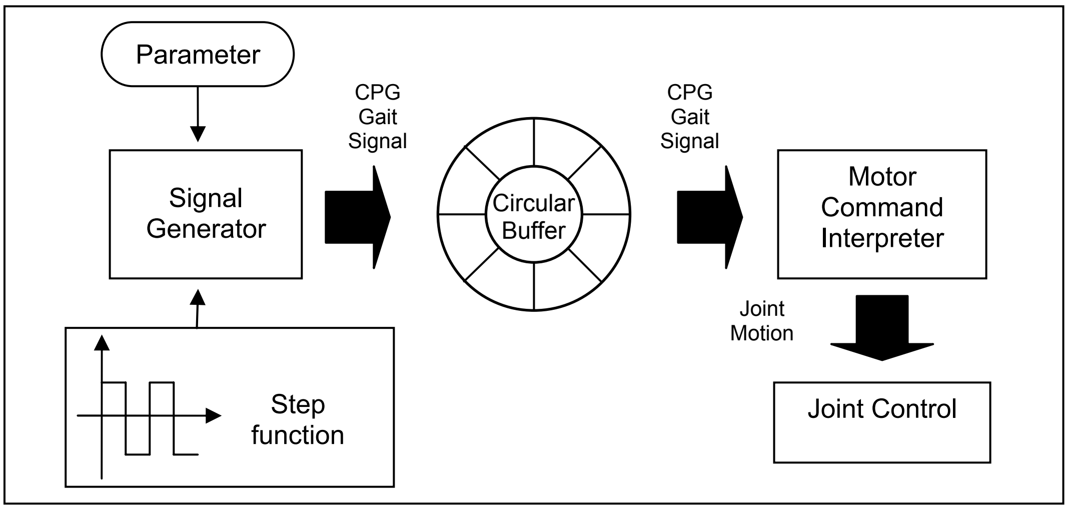


Figure 6.10 – Block diagram of CPG controller

The Signal Generator generates the oscillatory signal obtained from the solution to the Hodgkins-Huxley equations. The signal generated is stored in a buffer to prevent discontinuity in the signal fed to the motor. This buffer data is then fed to the second part of the program, the Motor Command Interpreter, which in turn transforms the signal to movement of the legs. This is shown in Figure 6.10. Since data is written at the start of the buffer and read at the end of the buffer, there is delay in transmission of data from the signal generator to the motor command interpreter. The Motor Command Interpreter reads from the signal buffer and executes either the return stroke or the power stroke accordingly to the respective leg. This delay is proportional to the size of the buffer. Hence the size of the buffer is small.

6.6.1 Circular Buffer

The buffer used is a circular buffer, which gets written by the Signal Generator, and read by the Motor Command Interpreter. There is a "stop-write" feature that stops writing when the buffer is full. A term is removed from the buffer whenever a read occurs. This will in turn cause the Signal Generator to write the next term into the buffer. Since signal generation writes at a higher rate than the reading of the Motor Command Interpreter, the "stop-write" feature of the buffer prevents the buffer from overflowing. The data moves at the pace of the slower of the reader and writer, which in this case is the Motor Command Interpreter.

The CPG controller is able to perform both static and dynamic walking. Changes in gaits are possible on the fly by altering the parameters online. These changes can be altered using the user interface provided. Sets of parameter, representing different gaits, are recorded which can be recalled when that particular gait is needed. Similarly these can be called using buttons provided on the user interface.

Stability requirements for dynamic and static locomotion are different (refer to Section 2.5.2). Therefore, different algorithms are required for dynamic and static gaits. And since static stability is only an issue with static gait, precautionary functions are included to ensure static stability before moving a leg. These functions are also present in the behaviour-based controller.

The program is written in such a way that a move has to be completed before the next move can carry on. Each move can consist of movement of more than one leg depending on the signal generated. The buffer reader ensures that each move is completed before it accepts the next move.

The robust algorithm used is the same algorithm used in the behaviour-based controller. The avoidance mechanism and the foot placement mechanism are reused in this controller. Since the program does not allow overriding moves, there is no time constraint on each move, allowing it to perform other tasks during a move.

6.7 Results

The results of two gaits are shown here. The first is a gait for static walking while the second gait is a simulation of dynamic walking. It is mentioned previously that a high in the signal represents the return stroke and a low represents a power stroke. The interpretation of the signal is altered for the static gait (first set of results). The high signal triggers the leg to start its move, which moves the foot from the back position to the forward position. This gives the robot time to run the various precautionary algorithms during the move. The next leg is signalled to move once the leg has completed its task and the body advances forward once all four feet have completed the move forward. This set of signal can be applied to different robots depending on the strokes defined for different robots.

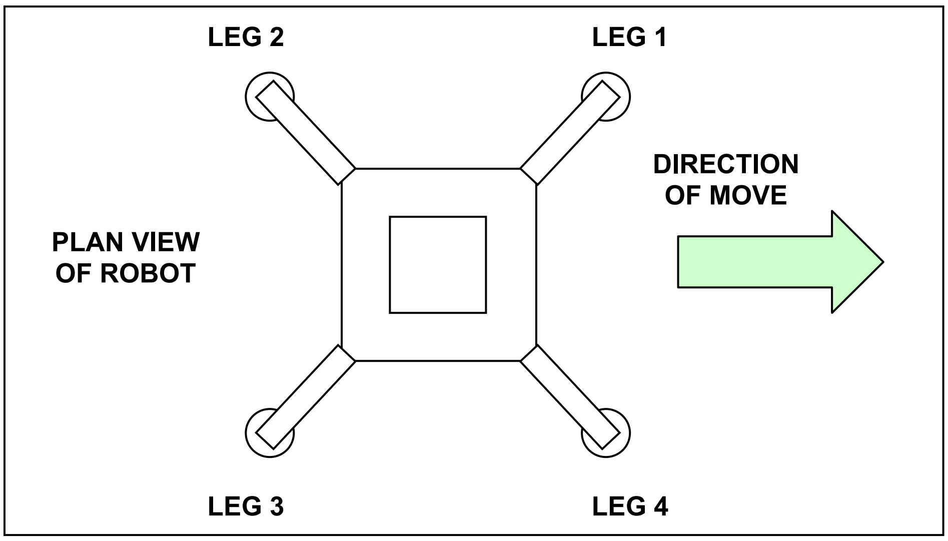


Figure 6.11 – Leg number convention.

Figure 6.11 shows the numbering convention for the legs of the robot, where leg 1 and leg 4 are the front legs, and leg 2 and leg 3 are the hind legs. Figure 6.12 and Figure 6.13 shows the signal for all four legs for a static gait. This is a gait pattern where leg one moves first, followed by legs four, two and lastly three. The sequence of leg movement can be changed by altering the parameter of the equations. The robot is able to perform a prolonged walking gait. Since the robot completed moving a leg before it moved other legs, the gait generated is stable and the robot is able to run the different checks to prevent instability.

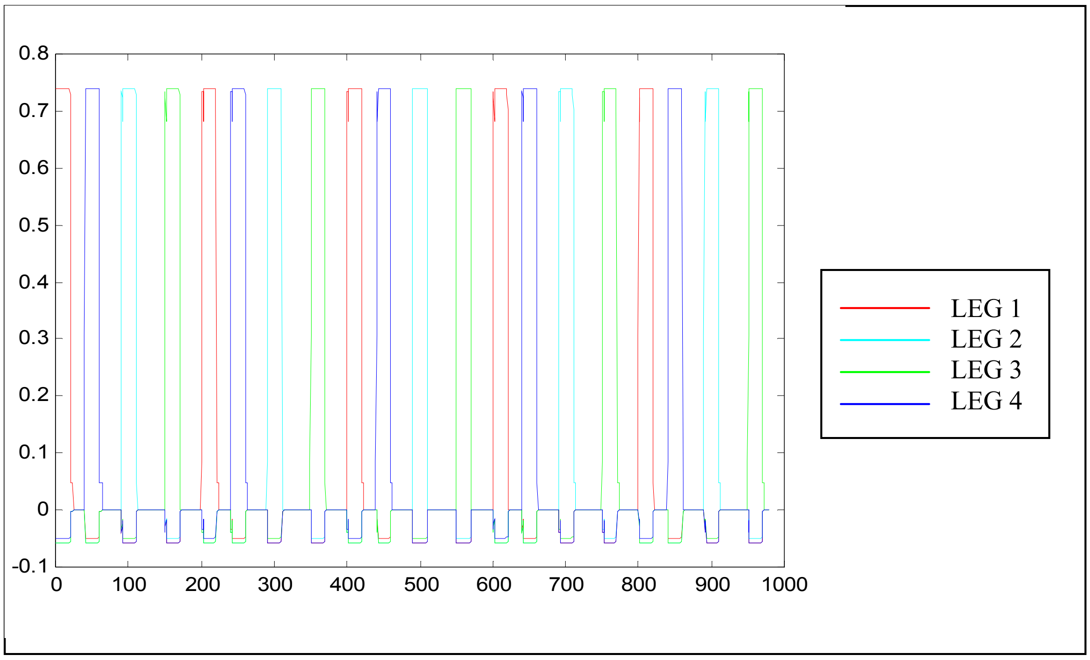


Figure 6.12 – Static gait diagram of the trigger signal produced by the CPG controller

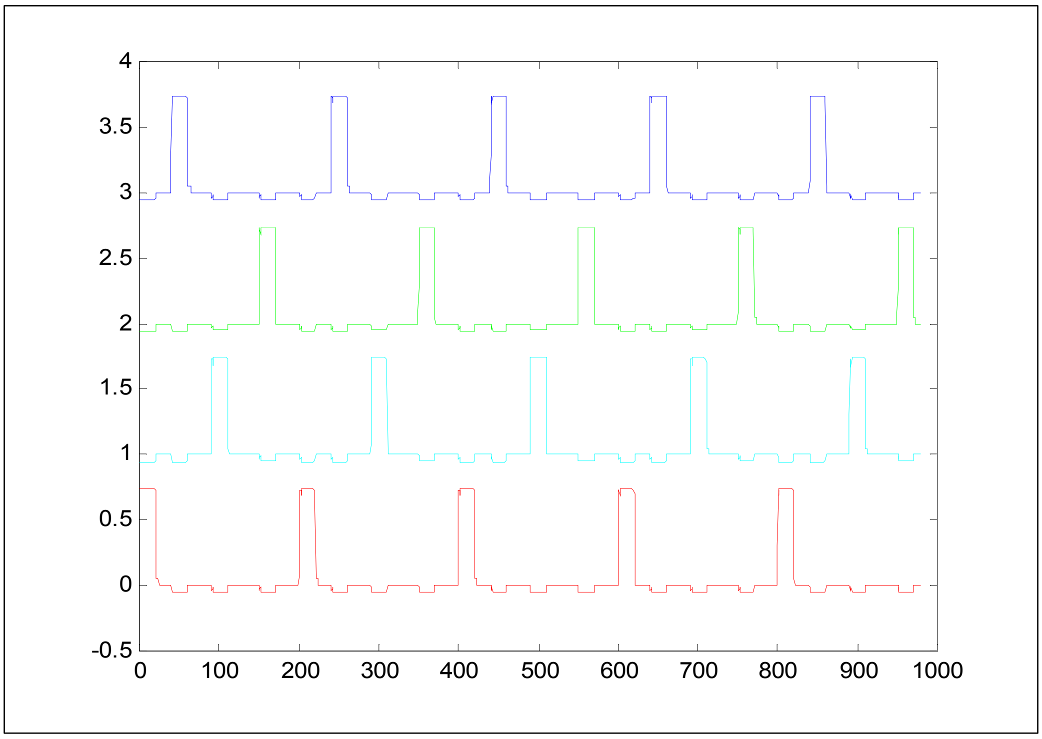


Figure 6.13 – Split signal diagram for static gait.

Figure 6.14Figure 6.14 and Figure 6.15 show a simulation of dynamic walking, with first movement by leg one and three, followed by two and four. This is a simulation of the equations, as dynamic walking cannot be implemented on the quadruped, due to the robot's mechanical design constraints. The speed of motion of the legs cannot produce a stable dynamic gait. The signal is fed to the robot while the robot is suspended to observe the movement. As a dynamic gait does not require checks for stability, the signal is fed directly to the robot. The signal interpretation of the dynamic gait is set to execute the return stroke on receiving a high signal and executing the power stroke on the low signal. The controller is able to sustain the gait over a long period of time.

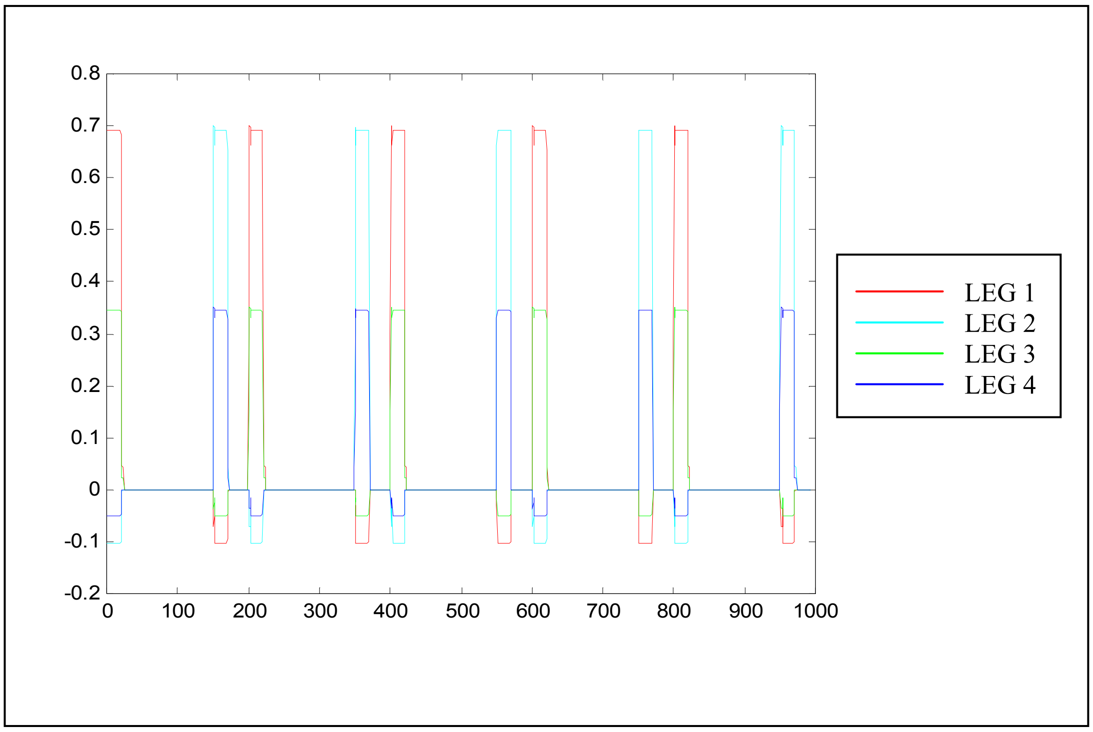


Figure 6.14 – Simulated trot gait for a quadruped.

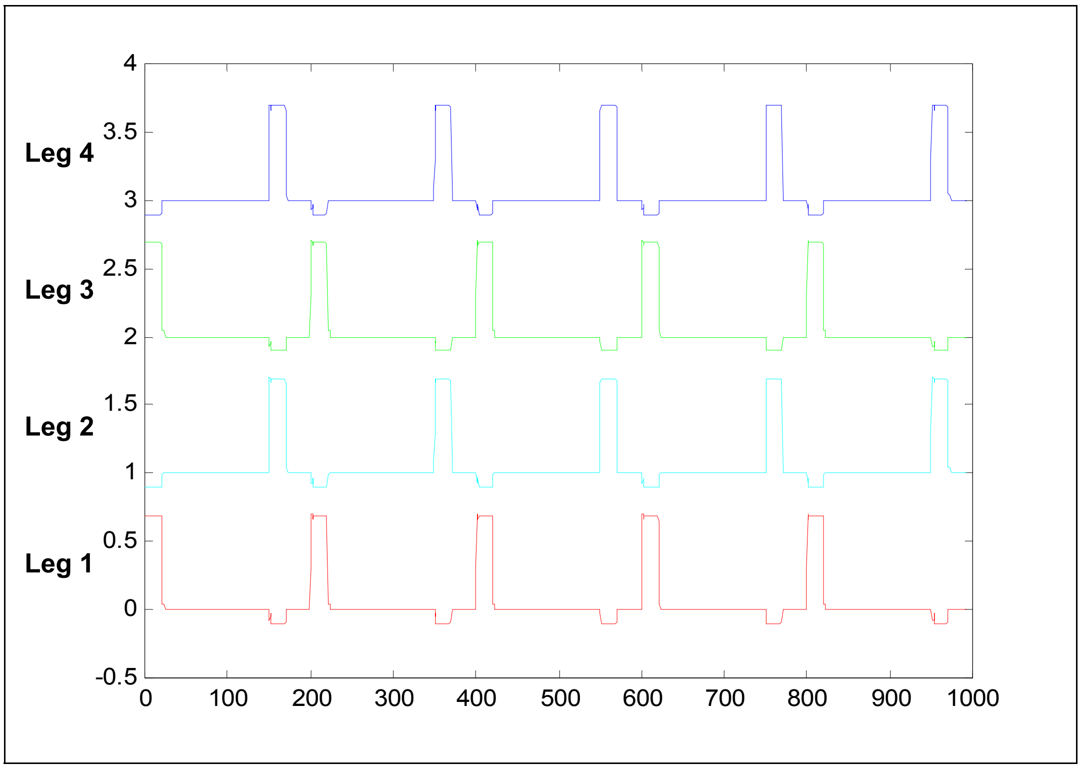


Figure 6.15 – Split signal diagram for simulated trot gait for a quadruped

6.8 Summary

The CPG controller is derived from studies of the spinal system of animals. There exists a system away from the brain that coordinates legged locomotion of animals. This system generates a pattern that coordinates the movement between the different legs of the animals. Different research groups have developed different methods of achieving the CPG. In this thesis, the controller is developed using a group of differential equations that mimic biological systems.

The controller consists of two parts, one generating the pattern signal and the other interpreting the signal for motor movement. Changes in the signal can be made by altering different parameters. This controller is able to produce both static and dynamic gait signals but only static walking was possible. Different sets of parameters can produce the different gaits, while changes in the gait can happen online using the interface provided.

The same low level controller is used for this controller. The robot is able to perform online change in gait when the parameters are altered online. Two versions of the CPG controller is developed, one to cater for static walking while the other for dynamic walking. The controller produces simple static walking and the dynamic walking gait is tested on the robot while it is suspended. Both static and dynamic gaits are stable and sustainable.

Chapter 7

FUTURE RECOMMENDATIONS

7.1 Merging the Two Approaches

This project uses two methods to develop a robust walking controller - the CPG method and the behaviour-based approach. The advantages and disadvantages of these methods have been presented in their respective chapters. Since the two controllers achieve legged locomotion differently, both having their own advantages, thus combining the two approaches may provide a better solution. But both have their distinct characteristic, and there are problems involved in combining the two approaches. This chapter aims to combine the two approaches and develop a controller that takes the advantages of the two controllers and remove the disadvantages. The various methods that can be employed to the integration are presented.

7.1.1 The Advantages of the Two Approaches

These two approaches are results of the studies on biological systems. Humans have yet to comprehend how biological systems work, but in the world of artificial intelligence and engineering, various theories and algorithms have been coined to obtain the results evident in biological systems. Biological systems are

1) fast to react (fast reflexes),
2) able to adapt to the environment well,
3) fast learner, and

4) able to handle changes in the environment quickly.

The two controllers are two biologically inspired controller mimicking different parts of the biological system - one based on the pattern generators present in the spinal system, the other based on reasoning and action.

The behaviour-based controller is an incremental modular controller with interacting modules to provide the control required. This is an advantage as it allows the developer to add different features to the controller. However, the structure of the behaviour-based system is larger and the overhead is greater than that of the CPG. It used more resources as the number of modules required to perform simple locomotion is large, hence the interactions between the modules are higher as well.

On the other hand, the CPG uses signals generated from a pattern generator to control the different legs to produce locomotion. The CPG controller is a simple low-level locomotion controller. This controller is not a modular controller and therefore adding algorithms to the controller is difficult. The advantage of the CPG controller is its ability to change the gait online by changing the parameters of the system. Dynamic walking can even be achieved using this controller. Crucially, this method has very little overhead and is an efficient method of producing legged locomotion.

7.1.2 Solving their differences

The CPG controller controls the movement of the legs within the spinal system. On the other hand, the behaviour-based controller works like the brain of the system.

Therefore if a combined controller is implemented, the CPG should be used for the low level control while the behaviour-based controller acts as the high level controller. This model is a representation of how the human locomotion system works. The human spinal system produces rhythmic signals, which is modulated by brain signals to produce robust locomotion. Therefore, the behaviour-based controller plays the role of the observer, sensing peculiarities in the environment.

7.1.3 Various Implementations

7.1.3.1 Behaviour-based Modulated CPG Controller

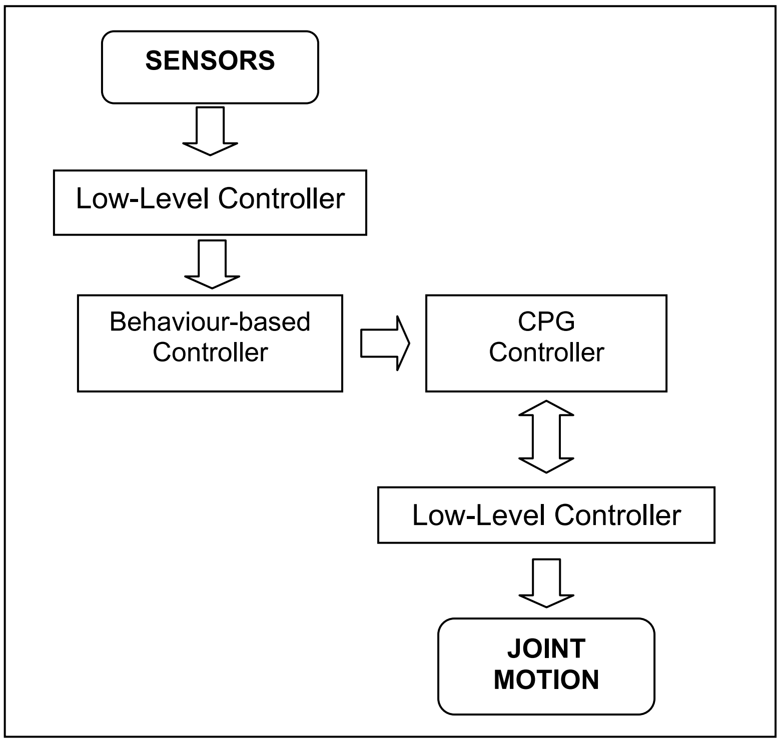


Figure 7.1 – Behaviour-based controller modulates the CPG controller by altering the CPG parameters.

This first method uses the behaviour-based controller to modulate the parameters in the CPG controller (Figure 7.1). The CPG controller acts as a low level controller. The behaviour-based controller now relinquishes control of all the joints and instead modifies the parameters of the CPG. The CPG controller will take full control over the joint movements. With the combined controller, the CPG works normally when there are no obstacles or problems in locomotion. The behaviour-based controller sends corrective signals to the CPG controller when it senses error in the locomotion.

The behaviour-based controller changes the gait pattern and controls the speed of the locomotion by modifying the parameters of the CPG controller. The behaviour-based controller is not able to obtain the same reaction time as before as it does not have control over low level commands of the robot. More parameters are required for the behaviour-based controller to have better control over the CPG controller.

Since there are many parameter in the CPG controller, modulation of these signals is difficult. Different sensors may be used to trigger changes in the different parameters but many trials have to be conducted before the parameters changes for the various sensors can be found. Secondly, the parameters control the speed and type of gait but not the actions of the robot. With the different difficulties faced, the control mechanism of the behaviour-based control becomes complicated.

7.1.3.2 A CPG Module within the Behaviour-based Controller

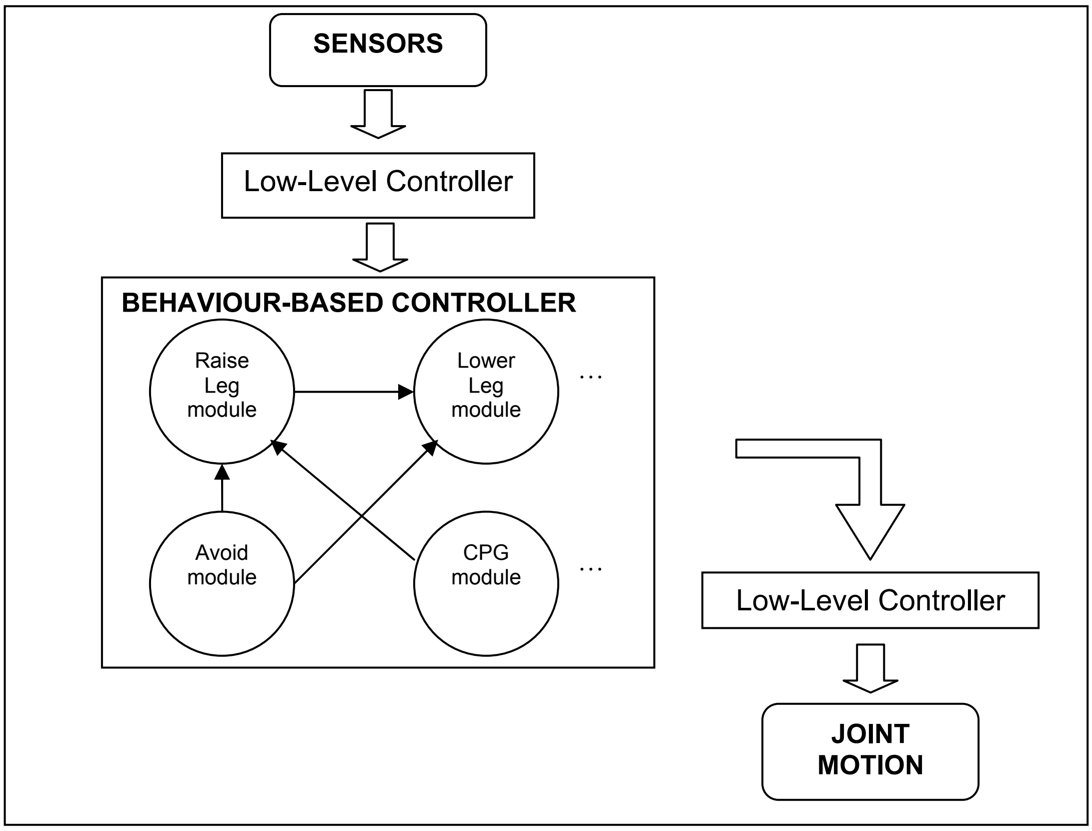


Figure 7.2 – The CPG controller is a module within the behaviour-based controller.

With this method, the CPG is added to the behaviour-based controller as a module. The CPG module takes over the control of basic locomotion, with the various parameters controlling the gait pattern (Figure 7.2). The various reaction modules, such as the *Relocate Leg* and the *Avoid* module, developed in the original behaviour-based controller will still be used. These reaction modules take the role of subsuming the CPG module when various obstacles are encountered. These modules will rectify the situation before returning controller back to the CPG controller.

The problem with this controller is that the coordination between the various modules is difficult as the gait is controlled by a single module. The behaviour-based controller requires that as long as a task can be spilt, it should be into various manageable modules. If the module takes control of the entire locomotion, the flexibility of the controller degrades.

The *Avoid* mechanism is only turned on when the leg is moving forward (after the leg is raised). The *Relocate Leg* mechanism is turned on when the leg is in the lower leg phase. These must be considered in the CPG module. Will the CPG module be in full control of all existing leg movement modules (*Raise Leg, Forward Leg* and *Lower Leg*)? This defeats the purpose of using the CPG controller. The goal is to develop a low level module that is subsumable by other modules, i.e. to stop the module in its path and allow it to carry on once the corrective action has been taken. But these corrective modules can only be turned during certain movements made by the CPG module. There must be coordination between the CPG module and the corrective modules, which increases the dependency the modules have on each other, and alters the rules used for the behaviour-based controller. As can be seen, the coordination between the various modules is difficult as the CPG module takes on too many roles of locomotion.

7.1.3.3 Parallel Integration Approach

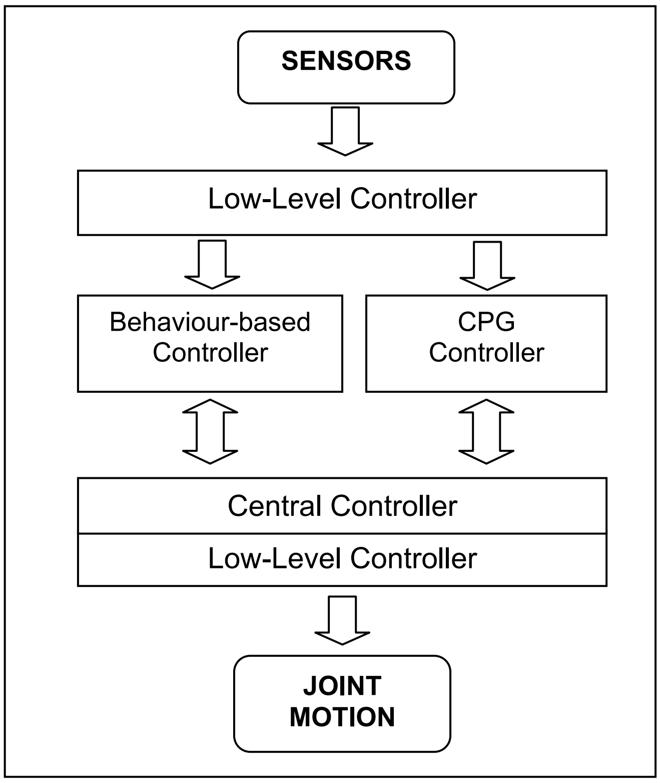


Figure 7.3 – Behaviour-based and CPG controller working in parallel. The central controller takes input from both controllers and decides on the action required.

Another approach is for the two controllers to work parallel to each other (Figure 7.3). The two controllers both have low-level controller which sends their signal to a central controller. These signals can be either be an additive signal or a subtractive signal to the central controller. The central controller combines the two signals from the two controllers and decides on the correct action to take. There are no communications between the two controllers and therefore sensory feedback is important. The two individual controllers make decisions based on sensory feedback. The central controller can be implemented as a neural network based controller.

The problem with this controller is that the two controllers require corrective action to be taken. Since it depends on the central controller to move the robot, the controllers are at its mercy, to decide if the correct action is taken. This is the dilemma is similar to two person being tied together with one wanting to move left and the other wanting to move right. Both persons need to move in the respective directions to achieve its goal but only one can be satisfied.

7.1.4 Summing Up the Combined Controller Approaches

Integration of the two controllers has the following advantages.

1) The CPG controller enhances the robustness of the behaviour-based controller.
2) The CPG controller reduces the complexity and size of the behaviour-based controller.
3) The behaviour-based controller's modular approach makes it easy to add features to the CPG controller.

The three different methods of integration have been explored but not implemented as the implementation is beyond the scope of this work and an area of future work. These methods capture the advantages of the two approaches but realise the conflicts between the two methods. The first implementation is a serial type of integration where the behaviour-based controller modulates the CPG controller only if there are changes in the environment. The second places the CPG mechanism within the behaviour-based controller, and lastly the third is a parallel approach where the two controllers run concurrently with a central process that integrates the signal from both controllers.

It is difficult to combine these two controllers as they require quick reaction to work properly. Both controllers cannot take control over locomotion as they differ in the method of control. Since these two controllers are considered active reactive controllers, they must react to changes detected in the environment. Since only one controller can take control over the joints, the other controller cannot be optimally utilised.

7.2 Other Recommendations

More autonomy can be developed by including a navigation system to the robotic system. Since there are no sensors onboard that gives an indication that the robot is instable, no form of recovery mechanism can be implemented. This can be achieved with an inclinometer.

More features can be included as more sensors are added to the system. Since a modular-based system has been implemented, more modules can be included as more are invented. Other features such as landmark detection and representation can be included in the system at a later point in time. As the legs can be can quite wobbly when it is stationary, the robot will trip if it is pushed a little on the side. With the inclinometer, the tilt change in the robot's body can be detected and the foot can be lowered to recover from this instability.

7.2.1 Navigation Features

The robot should build a world model using the various sensors available on the robot. The controller can use a set of parameters to obtain a rough 3D map of the environment. This would involve noting the height, various obstacles, door, and walls. This representation should not contain too much information to reduce processing power required. Information that is relevant can guide the robot through the surrounding terrain to avoid obstacles encountered before, predicting the shortest passable route. This guidance not only produces effective locomotion but also avoids entrapment of the robot. Planning can speed the locomotion of the robot as the robot can choose the terrain it traverses through.

7.2.2 Portability of Controller

A modular software structure has been created. This modularity should be used to port the controller to other legged robots. The portability of the controller has to be improved. Building a module controller that suits all legged robots of any configuration as long as the actuators and sensor interface are defined properly.

[Fujita 1999] is an example of a modular reconfigurable robot with software to suppose the different configuration of the robot. This work can pave the way for modular robotic work or modification of current work to be more modular. This allows different students to work using the same language with different configured robots. All legged robots should use a single platform of development as this allows students to work closely without having to switch between the different platforms they are working on. Students working on different robots are able to understand what the developer is trying to achieve with his robot.

7.2.3 Memory

Humans and animals remember features in the environment and also methods in dealing with different situations. This ability is useful, much like learning in neural networks, as if the robot is left to move on its own, it should gain "experience" in clearing obstacles, and finding its way around the environment. The robot should have a memory where these details are stored. The way humans and robots perceive differ in the sense that humans think in very abstract terms whereas robots require specifics to associate objects.

There are examples where the robot builds a three dimensional map of the environment as it moves around the area. There are also robots that learning to clear obstacles better as the robot practices clearing them. These different features can be integrated into the robotics system to improve the robot's ability to survive in the real world.

7.3 Conclusion

Working on this project has given the author insight about the various aspects of legged locomotion, both in the engineering and biological sense. There are many areas that can be dealt with in terms of walking gait analysis, controller design and legged robot design. This project has explained the many factors that contribute towards the success of building a robust robot. But the robot controller is the objective and area of research in this project.

Although the robot has a set controllers that have been developed by previous researchers, the entire software controller has been redesigned. The system is ported to the Windows NT platform and hence many changes are necessary. A new controller is designed and developed to increase the portability and capacity of the robot controller software.

The study of the various controllers developed for legged robots has led to the development of two types of controllers. The first controller is the behaviour-based controller based on Brooks' Subsumption architecture. A set of basic modules are developed to perform basic locomotion. Modules that interact with sensors are added to improve the stability of walking. Modules are developed to move in all directions and turn both left and right. Other modules are created to improve on the robustness of the controller. These modules include modules that handle situations where the robot is faced with an obstacle or a pothole. These modules are created depending on the sensors available to the robot.

The second is the CPG controller that mimics the biological pattern generator, which produces locomotion in animals. This controller is effective in producing both static and dynamic gaits, with the gaits sustained over a long period of time.

The two controllers, behaviour-based and CPG, achieved stable walking locomotion. The walking algorithm used included the ability to overcome obstacles, find foot placement and work out a gait on its own if the current gait failed. In terms of robustness of the controller, the aim of this project is achieved.

In conclusion, humans and animals have walked the earth since creation and therefore the area of walking should not be considered foreign to us. Yet there seems to be a difference between knowing and application. There are more to consider when applying the knowledge. Researchers have made actuators and sensors that work like the human motor and sensory system, but looking at the current research in legged locomotion, there are not many robots that can achieve human-like or animal-like locomotion. The real challenge is to comprehend and apply the knowledge on machines.

The popularity of legged robots is growing as commercial interest in this field grows. But to date there are no robots that can closely resemble an animal in terms of walking. It lacks the dexterity, agility, learning skills and recovery mechanism. Examples of legged robots available that are well-known to the public are Sony’s AIBO, Honda’s biped (Asimo) and various other mobile robots that do not come close to mimicking human and animals in terms of movement.

Research in legged locomotion has great potential and a lot can be discovered as yet by researchers, as there are many areas to consider in terms of robustness of legged robots. This project has only touched the surface of this field. There is still much more to be done in terms of design, sensors, and algorithm.

Even as the world watches in amazement at the wonders of technological advancement of legged robots, the hope is that this field will mature as more is learnt about legged locomotion and its application.

REFERENCES

[Al-Jumaily 1999]	Al-Jumaily, A.A. & Amin, S.H.M. Fuzzy Logic Based Behaviors Blending for Intelligent Reactive Navigation of Walking Robot. 5th International Symposium on Signal Processing and its Application, 1999, Brisbane, Australia, pp. 155-158.
[Arkin 1989]	Arkin, A.C. Towards the Unification of Navigational Planning and Reactive Control. Proceedings of the AAAI Spring Symposium on Robot Navigation, 1989, pp. 1-5.
[Arkin 1989a]	Arkin, A.C. Motor Schema-based Mobile Robot Navigation. International Journal of Robotics Research, Vol. 8(4), pp. 92-112, 1989.
[Atienza 1998]	Atienza, R.O. An AI-enhanced Control System for a Four-legged Robot. MEng Thesis, Department of Mechanical Engineering, National University of Singapore, 1998.
[Atienza 2001]	Atienza, R. O. and M. H. Ang Jr. A Flexible Control Architecture for Mobile Robots: an Application for a Walking Robot. Journal of Intelligent and Robotic Systems, Vol. 30 No. 1, pp 29-48, 2001.
[Atsma 1999]	Atsma, W.J. Genetic Algorithms for Central Pattern Generator Evolution – Making Stiqman Walk. World Web Web, http://www.computer.org/books/stiquito/neurons/neurons.htm.
[Bares 1999]	Bares J.E. & Wettergreen D.S. Dante II: Technical Description, Results, and Lessons Learned. International Journal of Robotics Research, Vol. 18, no. 7, pp. 621--649, July 1999.
[Barnes 1998]	Barnes, D.P. Hexapodal Robot Locomotion over Uneven Terrain. Proceedings of IEEE International Conference on Control Applications, September 1998, Italy, pp. 441-445.
[Bay 1987]	Bay, J.S. & Hemami, H. Modeling of a Neural Pattern Generator with Coupled Nonlinear Oscillators. IEEE Transactions on Biomedical Engineering, Vol. BME-34, No 4, April 1987, pp. 297-306.
[Beer 1998]	Beer, R.D., Chiel, H.J., Quinn R.D., & Ritzmann R.E. Biorobotic Approaches to the Study of Motor Systems. Opinion in Neurobiology, Vol. 8, pp. 777 – 782, 1998.
[Beer 1999]	Beer, R.D., Chiel, H.J., & Gallagher J.C. Evolution and Analysis of Model CPGs for Walking II. General Principles and Individual Variability. Journal of Computational Neuroscience, Vol. 7, pp. 119-147, 1999.

[Briones 1994] Briones, L., Bustamante, P., & Serna, M. A. Wall-climbing Robot for Inspection in Nuclear Power Plants. IEEE International Conference on Robotics and Automation, pp. 1409—1414. 1994.

[Brooks 1986] Brooks, R.A. A Robust Layered Control System for a Mobile Robot. IEEE Journal of Robotics and Automation, Vol. 2(1), pp. 14-23, 1986.

[Brooks 1989] Brooks, R.A. A Robot that Walks; Emergent Behaviors from a Carefully Evolved Network. In Neural Computation, Vol. 1(2), pp. 253-262, 1989.

[Brooks 1990] Brooks, R.A. The Behavior Language. Technical Report 1227, MIT AI Lab, 1990.

[Brooks 1991] Brooks, R.A. Intelligence Without Representation. Artificial Intelligence, Vol. 47, pp. 137-159, 1991.

[Brooks 1991a] Brooks, R.A. Integrated Systems Based on Behaviors. ACM SIGART Bulletin, Vol. 2(4), pp. 46-50, 1991.

[Brooks 1994] Brooks, R.A. & Stein, L.A. Building Brains for Bodies. Autonomous Robots, Vol. 1(1), pp. 7-25, 1994.

[Cheong 1998] Cheong, C.H. The Design and Building of a Four-legged Walking Vehicle. MEng Thesis, Department of Mechanical Engineering, National University of Singapore, 1998.

[Chiel 1999] Chiel, H.J., Beer, R.D., & Gallagher, J.C. Evolution and Analysis of Model CPGs for Walking I. Dynamic Modules. Journal of Computational Neuroscience, Vol. 7, pp. 99-118, 1999.

[Cohen 2002] Cohen, A.H. & Boothe, D.L. Sensorimotor Interactions: Principles Derived from Central Pattern Generators. In Handbook of Brain Theory and Neural Networks, Ed: Arbib, M.. MIT Press, Cambridge, MA, 2002.

[Cruse 1994] Cruse, H., Bartling, C.H., Cymbalyuk, G., Dean, J., & Dreifert, M. A Neural Net Controller for a Six-legged Walking System. From Perception to Action Conference, Ed: Gaussier, P. & Nicoud, J.-D.. IEEE Computer Society Press, pp. 55-65, 1994.

[Cruse 1991] Cruse, H, Dean, J, Muller, U & Schmitz, J. The Stick Insect as a Walking Robot. In Proceedings 5th International Conference on Advanced Robotics, Vol. 2, 1991, Pisa, Italy, pp. 936--940.

[Cruse 1995] Cruse, H., Brunn, D.E., Bartling, C., Dean, J., Dreifert, M., Kindermann, T., & Schmitz, J., Walking: A Complex Behaviour

	Controlled by Simple Networks. Adaptive Behaviour 3(4), pp. 385-418, 1995.
[Cruse 1998]	Cruse, H., Kindermann, T., & Dean, J. A Biologically Motivated Controller for a Six-legged Walking System. Proceedings of the 24th Annual Conference of the IEEE Industrial Electronics Society, September 1998, Aachen, Vol. 4, pp. 2168-2173.
[Digney 1993]	Digney, B.L. & Gupta, M.M. A Distributed Adaptive Control System for a Quadruped Mobile Robot. From Animals to Animats 3, Proceedings of the 3rd International Conference on Simlulation of Adaptive Behavior, August 1994, Brighton, England, pp. 344-353.
[Farquharson]	Farquharson, A.S.L. On the Gait of Animals By Aristotle. Written 350 B.C.E (Translated). World Wide Web Publication, http://classics.mit.edu/Aristotle/gait_anim.html.
[Flynn 1989]	Flynn, A.M. & Brooks, R. A. Building Robots: Expectations and Experiences. IEEE/RSJ International Workshop on Intelligent Robots and Systems, September 1989, Tsujuba, Japan, pp. 236-243.
[Fujita 1999]	Fujita, M., Kitano, H., & Kageyama, K. A Reconfigurable Robot Platform. Robotics and Autonomous Systems, Vol. 29, pp. 119-132, 1999.
[Grillner 1975]	Grillner, S. Locomotion in Vertebrates: Central Mechanisms and Reflex Interaction. Physiol. Rev., Vol. 55, pp. 247-304, 1975.
[Grossberg 1997]	Grossberg, S., Pribe, C. & Cohen, M.A. Neural Control of Interlimb Oscillations, I. Human Bimanual Coordination. Biological Cybernetics, Vol. 77, pp. 131-140, 1997.
[Grossberg 1997a]	Grossberg, S., Pribe, C. & Cohen, M.A. Neural Control of Interlimb Oscillations, II. Human Bimanual and Coordination. Biological Cybernetics, Vol. 77, pp. 131-140, 1997.
[Hillel 1992]	Hillel, J.C., Beer, R.D., Quinn, R.D., & Espenschied, K.S. Robustness of a Distributed Neural Network Controller for Locomotion in a Hexapod Robot. IEEE Transactions on Robotics and Automation, Vol 8(3), June 1992.
[Ijspeert 2001]	Ijspeert, A.J. A Connectionist Central Pattern Generator for the Aquatic and Terrestrial Gaits of a Simulated Salamander. Biological Cybernetics, Vol. 84(5), pp. 331-348, 2001.

[Ijspeert 1999]	Ijspeert, A.J. & Kodjabachian, J. Evolution and Development of a Central Pattern Generator for the Swimming of a Lamprey. Research Paper No 926, Department of Artificial Intelligence, University of Edinburgh, Edinburgh, 1999.
[Jantzen 1998]	Jantzen, J. Design of Fuzzy Controllers. Technical University of Denmark: Dept. of Automation, Technical Report No 98-E-864.1998.
[Keele 1968]	Keele, S.W. Movement Control in Skilled Motor Performance. Psychological Bulletin, Vol. 70(6), pp. 387-403. 1968.
[Kimura 2000]	Kimura, H. & Yasuhiro, F. Adaptive Dynamic Walking of the Quadruped on Irregular Terrain – Autonomous Adaptation Using Neural System Model. Proceedings of the 2000 IEEE International Conference on Robotics & Automation, April 2000, San Francisco, CA, pp.2037-2042.
[Kodjabachian 1995]	Kodjabachian, J. & Meyer, J-A. Evolution and Development of Control Architectures in Animats. Robotics and Autonomous Systems, Vol. 16(2-4), pp. 161-182. 1995.
[Kong 2003]	Kong, Y.M. Control of a Walking Robot Using Computational Intelligence. MEng Thesis, Department of Mechanical Engineering, National University of Singapore, 2002.
[Koŝecka 1993]	Koŝecka, J. & Bajcst, R. Discrete Event Systems for Autonomous Mobile Agents. Proceedings of the Symposium on Intelligent Robotic Systems, 1993, Zakopane, pp. 21 – 31.
[Krotkov 1992]	Krotkov, E. & Simmons, R. Performance of a Six-legged Planetary Rover: Power, Positioning, and Autonomous Walking. IEEE International Conference on Robotics and Automation, Vol. 1, pp. 169 – 174, May, 1992.
[Pirjanian 2000]	Pirjanian, P. Multiple Objective Behavior-based Control. Robotics and Autonomous Systems 31, pp. 53-60. 2000.
[Likhachev 2002]	Likhachev, M. Kaess, M. & Arkin, R. C. Learning Behavioral Parameterization Using Spatio-temporal Case-based Reasoning. Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Vol. 2, pp. 1282-1289. 2002.
[Maes 1989]	Maes, P. How to do the Right Thing. Connection Science Journal, Vol. 1(3), pp. 291-323. 1989.
[Maes 1990]	Maes, P. & Brooks, R.A. Learning to Coordinate Behaviors. In Proceedings of the Eighth National Conf. on AI, pp. 896-802. 1990.

[Mat'aric 1992] Mat'aric, M.J. Behavior-Based Control: Main Properties and Implications. Proceedings of the IEEE International Conference on Robotics and Autonomation, Workshop on Architectures for Intelligent Control Systems, May 1992, Nice, France, pp. 46-54.

[Mat'aric 1997] Mataric, M.J. Behavior-Based Control: Examples from Navigation, Learning, and Group Behavior. Journal of Experimental and Theoretical Artificial Intelligence, special issue on Software Architectures for Physical Agents, 9(2-3), H. Hexmoor, I. Horswill, and D. Kortenkamp, eds., pp. 323-336, 1997.

[Matsuoka 1985] Matsuoka, K. Sustained Oscillations Generated by Mutually Inhibiting Neurons with Adaptation. Biological Cybernetics, Vol. 52, pp. 367-376. 1985.

[Nagakubo 1994] Nagakubo, A. & Hirose, S. Walking and Running of the Quadruped Wall-climbing Robot. Proceedings of the IEEE Conference on Robotics and Automation, pp. 10051012, 1994, San Diego, pp. 10051012.

[Nelson 1997] Nelson, G.M., Quinn, R.D., Bachmann, R.J., Flannigan, W.C., Ritzmann, R.E., & Watson, J.T. Design and Simulation of a Cockroach-like Hexapod Robot. IEEE International Conference on Robotics and Automation, April 1997, Albuquerque, New Mexico, pp.1106-1111.

[Nomani 2002] Nonami, K. Development of mine detection robot COMET-II and COMET-III. SICE Proceedings of the 41st SICE Annual Conference, Vol. 1, pp. 346-351, August 2002.

[Numerical Recipe] Press, W.H., Saul, A.T., & Vetterling, W.T. Numerical Recipes in C, The Art of Scientific Computing (2nd Edition). pp. 710-714, Cambridge University Press. 1988.

[Pearson 1993] Pearson, K.G. Common Principles of Motor Control in Vertebrates and Invertebrates. Annual Rev. Neuroscience, Vol. 16, pp. 265-297. 1993.

[Pratt 1995] Pratt, G.A. & Williamson, M.M. Series Elastic Actuators. Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Vol. 1, pp. 399-406, July 1995.

[Rosenblatt 1997] Rosenblatt, J. DAMN: A Distributed Architecture for Mobile Navigation. Journal of Experimental and Theoretical Artificial Intelligence, Vol. 9, No. 2 / 3, pp. 339-360. 1997.

[RTX User] Venturcom, Inc., RTX 5.0 User's Guide.

[Srinivasan 1992] Srinivasan, S., Gander, E., & Wood, H.C. A Movement Pattern Generator Model using Artificial Neural Networks. IEEE Transactions on Biomedical Engineering, Vol. 39, No 7, pp. 716-722. July 1992.

[Venkataraman 1996] Venkataraman, S.T. A Model of Legged Locomotion Gaits. Proceedings of the 1996 IEEE International Conference on Robotics and Automation, April 1996, Minneapolis, USA.

[Xu 1997] Xu, H. & Brussel, H.V. A Behaviour-based Blackboard Architecture for Reactive and Efficient Task Execution of an Autonomous Robot. Robotics and Autonomous Systems 22, pp. 115-132. 1997.

[Wettergreen 1995] Wettergreen, D. Robotic Walking in Natural Terrain – Gait Planning and Behavior-based Control for Statically-stable Walking Robots. Doctoral Dissertation, Tech. Report CMU-RI-TR-95-42, Robotics Institute, Carnegie Mellon University. December 1995.

[Wettergreen 1995a] Wettergreen, D.; Pangels, H.; Bares, J. Behavior-based Gait Execution for the Dante II Walking Robot. Proceedings of the 1995 IEEE International Conference on Intelligent Robots and Systems, Volume 3, pp. 274-279, August 1995.

[Yim 2000] Yim, M., Duff, D. & Roufas, K. PolyBot: a Modular Reconfigurable Robot. Proceedings of the 2000 IEEE Conference on Robotics and Automation, pp. 514-520, San Francisco, 2000.

APPENDIX A

Modified Sugeno-type Controller

The fuzzy controller is a Sugeno-type fuzzy controller [Jantzen 1998]. This method of inference is widely used, as it is computationally efficient and is able to handle different operating conditions of dynamic nonlinear systems.

The Sugeno-type inference method [Jantzen 1998] uses output membership functions that are either linear or constants. Taking a two input one output system as an example, a typical rule is as follows,

If ***x₁*** is **A** and ***x₂*** is **B** then

$$y = (p * x_1 + q * x_2) + k \qquad \textbf{(A.1)}$$

where x_1 and x_2 are inputs, ***A*** and ***B*** are fuzzy sets, y is the output for a rule, ***p*** and ***q*** are calculated weight values and ***k*** is a constant. The output, y, to every rule is a crisp constant. The output is a weighted sum of the individual rule outputs divided by the total weight,

$$Output = \frac{\sum w_i y_i}{\sum w_i} \qquad \textbf{(A.2)}$$

The calculation is not computationally intensive as shown by the simple formula used.

In this project, the term on the right of Equation (A.1) is modified to a simpler version. Instead of taking individual weighted sum, the equation has been changed to the following,

$$Modified\_Output = \frac{\sum W_i V_{ij}}{\sum W_i} \qquad \textbf{(A.3)}$$

where the weight W_i is calculated based on Equation (A.4), which depends on the two inputs. $\boldsymbol{V_{ij}}$ is a constant that depends on a lookup table that is a function of the two inputs.

$$\mathrm{W_i = AND(weight(x_{1i}), weight(x_{2i}))} \qquad \textbf{(A.4)}$$

where weight($\boldsymbol{x}$) is the weight of $\boldsymbol{x}$ obtained from a fuzzy membership function.

and $\quad AND(A,B) = \min(A,B)$

Therefore, this equation gives the average of the weighted sum of the various memberships of the membership function.

In this project, the two inputs are error (*Error*) and rate of change in error (*dError*), and the output is the motor voltage. Error is defined as the difference in the desired position and the current position of the motor. Change in error is defined as the rate of the change of error per unit second.

Using the modified equation (Equation (A.3)) as the output of the controller, the voltage of the motor is calculated. Both weights in Equation (A.4) needed for the W term need fuzzy membership function. The two membership functions are shown in Figure A.1.

The weight value (*Error* term) is obtained from the membership function from the top figure and the weight value (*dError* term) is obtained from the membership function in the bottom figure. Constants a, b, c, d, a', b', c', and d' are used in the membership functions of the fuzzy controller. The controller is tuned by adjusting the values of these constants. There are overlaps in the triangles of the membership function to ensure a smooth transition between the points on the graph.

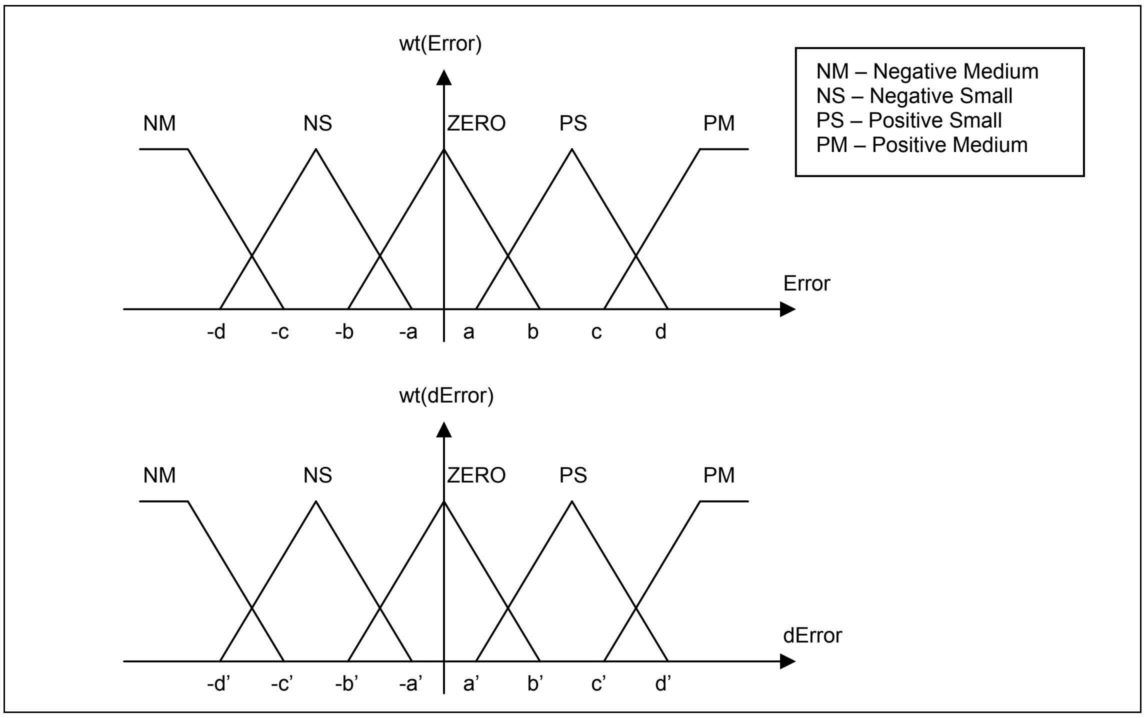


Figure A.1 – Membership functions of *Error* and *dError*

		DError				
		NM	NS	Z	PS	PM
Error	NM	NM	NM	NS	NS	NS
	NS	NS	NS	NS	Z	Z
	Z	Z	Z	Z	PS	PS
	PS	Z	Z	PS	PS	PM
	PM	PS	PS	PS	PM	PM

Table A.1 – Lookup table for V value

represents the fuzzy rules used in the fuzzy controller. For example,

Rule 1 - *If* Error *is* ***Negative Medium*** *and* dError *is* ***Zero*** *then Voltage =* ***Negative Small***

or

Rule 2 - *If* Error *is* ***Positive Small*** *and* dError *is* ***Negative Small*** *then Voltage =* ***Zero***

All together 25 rules are used to obtain the voltage for the controller. The positive and negative values dictate the direction, which the motor moves. The V values used in Equation (A.3) are a set of crisp values that range from –0.8 to 0.8 volts. There are five values – Negative Medium, Negative Small, Zero, Positive Small and Positive Medium. These values are found in Table A.2.

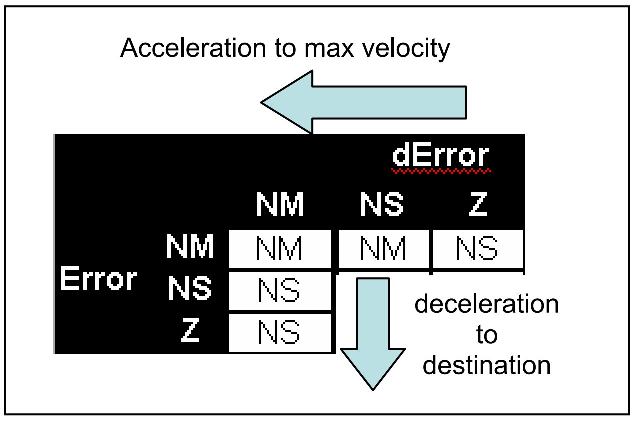
Representation	Value
Negative Medium	-0.8
Negative Small	-0.3
Zero	0.0
Positive Small	0.3
Positive Medium	0.8

Table A.2– Values of the voltage.

The following is an explanation of how the lookup table was derived. Example rule 1 states that if motor is far from the objective (i.e. *Error* = NM) and velocity is zero (i.e. *dError* = Zero), then slowly accelerate to maximum velocity (i.e. apply NS voltage) from the centre to the left of the table.

For example rule 2 states that if motor is near the objective (i.e. *Error* = PS) and velocity is small (i.e. *dError* = NS), then set motor voltage to zero for the motor to decelerate (i.e. apply Zero voltage).

Figure A.3 shows the method in which the different values of the table are chosen. If the motor is still and the error is increasing to a negative medium value, accelerate the motor to the max voltage. Once error reduces to a negative small value, decelerate the motor until error is zero. Since friction affects the motor's movement, the voltage is not zero when error is zero.



		dError		
		NM	NS	Z
Error	NM	NM	NM	NS
	NS	NS		
	Z	NS		

Figure A.3 – Method to read the table using human reasoning

APPENDIX B

Creating a Module in the Behaviour-based Controller

This is a long process that the user has to go through to create a module in the behaviour-based controller. Note that a FSM is equivalent to module.

Step 1: Declarations

Things to place in '***definition.h***'

1) The FSM ID declaration.
2) All FSM states declaration.
3) Event Messages declaration.

Things to place in '***include_fsm.h***'

Include the fsm's header file into this file.

Things to place in '***process.h***'

Declare the prototype of the 2 threads for the continuous loop and the triggered loop.

Things to place in '***process.cpp***'

Declare the function for the 2 threads for the continuous loop and the triggered loop. These 2 loops will run concurrently.

Step 2: Create the different objects.

Things to place in '***new_behave.cpp***'

1) Create ***new*** object of FSM in main function.
2) Create the trigger event.
3) Create the 2 threads for the triggered process and the continuous process for the module.

Things to place in '***fsm.cpp***'

1) Declare extern hEvent declaration of the Event handle for the FSM.
2) Add the Event created to the ***sendMessage()*** function in FSM class.

Things to place in the newly created fsm file '***name_fsm.cpp***'

Copy an old FSM source file file to modify.

1) Create Registry with the ID for the FSM and the initial state of the FSM.

   ```
   createRegistry(AVOID3, IDLE);
   ```

2) Build the state table. Input to table include the exhibit/inhibit signal, previous, current and next state;

   ```
   ps_tbl = (STATE_TABLEPTR)malloc(sizeof(STATE_TABLE));

   ps_tbl = NULL;

   buildStateTable(&ps_tbl,INHIBIT,DONT_CARE,DONT_CARE,STOP);

   buildStateTable(&ps_tbl,EXHIBIT,DONT_CARE,DONT_CARE,START);
   ```

3) Create inhibit/exhibit list. First declare a pointer to the inhibit/exhibit table. Create an inhibit state pointer to each term in the table for each state. Terms are added to the ref pointer which is in turn added to the inhibit/exhibit table. In the example below, START is the only state during which an inhibit/exhibit signal is sent out. START inhibits two other FSMs, i.e. RAISE_LEG2 and

FORWARD_LEG2.. After the inihibit ref pter is created, we add it to the inhibit/exhibit table.

```
pi_tbl = NULL; //init inhibit/exhibit table pter

INHIBIT_REFPTR pi_r_start; //state pter

pi_r_start = (INHIBIT_REFPTR)malloc(sizeof(INHIBIT_REF));

pi_r_start = NULL; //init inhibit/exhibit ref pter

addInhibitRef(&pi_r_start,RAISE_LEG2,INHIBIT);   //add   inhibit
term for START state

addInhibitRef(&pi_r_start,FORWARD_LEG2,INHIBIT);  //add  inhibit
term for START state

addInhibitTable(&pi_tbl,START,pi_r_start);                 //add
inhibit/exhibit ref to the table
```

4) ALL ***pendonMessageQueueForExOrIn*** function are the same. The only difference is that that the event of every FSM is different.

5) This is where all the FSMs' functions differ. Both ***executeTriggeredFunction*** and ***executeCtsFunction*** are custom written for all FSMs. Depending on the state, decide on the action taken by that particular FSM.

Compile the module and the behaviour-based controller is updated.