

DEVELOPING A RAPID AND STABLE PARAMETRIC QUADRUPED
LOCOMOTION FOR AIBO ROBOTS

by
Tekin Meriçli

Submitted to the Faculty of Engineering
Department of Computer Engineering
Bachelor of Science

Undergraduate Program in Computer Engineering
Marmara University
Göztepe, 2005

DEVELOPING A RAPID AND STABLE PARAMETRIC QUADRUPED
LOCOMOTION FOR AIBO ROBOTS

APPROVED BY:

Assoc. Prof. Haluk Topçuoğlu
(Thesis Supervisor)

Prof. H. Levent Akın
(Thesis Co-supervisor)

DATE OF APPROVAL: 10.06.2005

ABSTRACT

DEVELOPING A RAPID AND STABLE PARAMETRIC QUADRUPED LOCOMOTION FOR AIBO ROBOTS

For a robot, the ability to get from one place to another is one of the most basic skills. However, locomotion on legged robots is a challenging multidimensional control problem. It requires the specification and coordination of motions in all of the robot's legs while accounting for factors such as stability and surface friction. Especially when the robot's aim is to chase a ball, dribble it as fast as it can, and shoot it towards the opponent goal, the importance of a rapid locomotion capability increases.

Robot soccer is a challenging research domain that is appropriate for studying a large spectrum of issues of relevance to the development of complete autonomous agents. A robot must have some capabilities in order to be able to play soccer. These capabilities can be listed as vision, locomotion, localization, planning, and communication. Although all of these capabilities have vital roles in playing soccer, locomotion is probably the most important one since a motionless soccer player cannot achieve its goal even if it has great planning, vision, localization, and communication skills.

The aim of this work is to develop a fast and stable parametric locomotion for Sony Aibo robots by using genetic algorithms for parameter optimization.

ÖZET

AIBO ROBOTLARI İÇİN HIZLI VE KARARLI PARAMETRİK DÖRT BACAĞLI HAREKET MODÜLÜ GELİŞTİRİLMESİ

Bir robot için bir yerden başka bir yere gidebilme yetisi en temel becerilerden biridir. Bununla birlikte, bacaklı robotlarda hareket oldukça uğraştırıcı çok boyutlu bir kontrol problemidir. Bir yandan denge ve yüzeydeki sürtünme gibi faktörler hesaba katılırken, bir yandan da robotun bütün bacaklarının hareketlerinin belirlenmesini ve koordinasyonunun sağlanmasını gerektirir. Özellikle robotun amacı bir topu kovalamak, onu olabildiğince hızlı bir şekilde sürmek, ve rakip kaleye doğru şut çekmek olduğunda hızlı hareket yetisinin önemi artar.

Tamamen özerk etmenlerin geliştirilmesine yönelik geniş bir alana yayılan konular üzerinde çalışmak için robot futbolu oldukça zorlayıcı bir araştırma alanıdır. Bir robot futbol oynayabilmek için bazı yetilere sahip olmalıdır. Bu yetiler görüş, hareket, yerini belirleme, planlama, ve iletişim olarak listelenebilir. Futbol oynamada bütün bu yetilerin hayati rolleri olmasına rağmen, çok iyi planlama, görüş, yerini belirleme, ve iletişim yetilerine sahip olsa bile hareketsiz bir futbol oyuncusu amacına ulaşamayacağından, muhtemelen hareket bu yetilerin en önemlisidir.

Bu çalışmanın amacı parametre optimizasyonu için genetik algoritmaları kullanarak Sony Aibo robotları için hızlı ve kararlı parametrik bir hareket modülü geliştirmektir.

TABLE OF CONTENTS

ABSTRACT	iii
ÖZET	iv
LIST OF SYMBOLS/ABBREVIATIONS	vii
LIST OF FIGURES	viii
LIST OF TABLES	xi
ACKNOWLEDGEMENTS	xii
1. INTRODUCTION	1
1.1. Outline	2
2. BACKGROUND	3
2.1. Robot Soccer	3
2.2. Sony Aibo Robot	5
3. GENETIC ALGORITHMS	6
3.1. Operators	7
3.1.1. Reproduction	7
3.1.2. Crossover	8
3.1.3. Mutation	9
3.2. Fitness Function	9
3.3. Initial Population	10
3.4. Stopping Criteria	10
3.5. Summary	10
4. RELATED WORK	12
4.1. Kinematic Model	12
4.2. Parameters	13
4.3. Locus	14
4.4. Walking Styles	14
4.5. Omnidirectional Motion	17
5. PROPOSED APPROACH	18
5.1. Kinematic Model	18
5.1.1. Inverse Kinematics	18

5.2. Walking	20
5.2.1. Representing the Locus	20
5.3. Object-oriented design of the locomotion module	21
5.4. Parameter Optimization	23
6. EXPERIMENTAL STUDY	28
6.1. Environments	28
6.1.1. Webots	28
6.2. Results	29
7. CONCLUSION & FUTUREWORK	32
APPENDIX A: Code Used in Parameter Optimization Process	33
REFERENCES	36

LIST OF SYMBOLS/ABBREVIATIONS

GA	Genetic Algorithms
UNSW	University of New South Wales

LIST OF FIGURES

Figure 2.1.	Field setup for RoboCup Legged League [1].	4
Figure 2.2.	ERS-210 and ERS-7.	5
Figure 3.1.	A sample roulette wheel for a generation having 4 chromosomes. .	7
Figure 3.2.	An example of crossover operation on two solution strings.	8
Figure 3.3.	An example of a mutation operation on a solution string.	9
Figure 4.1.	Kinematic model of an Aibo leg developed by rUNSWift [10]. . . .	13
Figure 4.2.	Rectangular locus used by rUNSWift [10].	15
Figure 4.3.	The loci produced by minimization technique in the sagittal plane, shown in bold. The dotted lines show the basic rectangular loci. .	15
Figure 4.4.	Different timing of each legs motion results in different walking styles [10].	16
Figure 4.5.	(a) High stance and (b) low stance.	16
Figure 4.6.	Using legs as the wheels of a shopping cart [12].	17
Figure 4.7.	Forward, sideways, and turn components are added vectorally to pbtain a resulting vector; which indicates the direction and limit of the paw movement [11].	17
Figure 5.1.	Simple kinematic model representation for front right leg.	19

Figure 5.2.	Resulting motions with different combinations of forward, sideways, and turnCW parameters: (a) only forward, (b) only sideways, (c) only turnCW, (d) forward and turnCW together [10].	20
Figure 5.3.	Movement of the paw on a (a) rectangular locus and a (b) half elliptic locus.	21
Figure 5.4.	Proposed locus in the shape of a hermite curve.	21
Figure 5.5.	Class diagram showing the relations between classes used in the locomotion module.	22
Figure 5.6.	Parameters related to initial paw locations [10].	24
Figure 5.7.	String representation of walking parameters.	24
Figure 5.8.	A sample string.	24
Figure 5.9.	A sample chromosome seperated into three parts. These parts are related to number of waypoints on the locus, shape of the front and back loci, and initial paw locations, respectively.	26
Figure 5.10.	Technique used for the crossover operation in our approach.	26
Figure 5.11.	Pseudo-code implementation of the proposed approach.	27
Figure 6.1.	A screenshot from Webots simulator environment showing an Aibo ERS-210 on its walking test platform.	28
Figure 6.2.	A screenshot from the simulator environment during the training process.	29

Figure 6.3.	Average and best fitness values of 30 generations each having 50 individuals.	29
Figure 6.4.	Average and best fitness values of 50 generations each having 100 individuals.	30
Figure 6.5.	Resulting parameter set that provided best fitness value.	30

LIST OF TABLES

Table 4.1.	Default values of the walking parameters used by ParaWalk [11]. .	14
Table 6.1.	Walking engine vs. Performance.	31

ACKNOWLEDGEMENTS

To my family...

Especially to my elder brother Çetin Meriçli for his guidance, support, and belief.

1. INTRODUCTION

A robot is a mechanical device which performs automated tasks, either according to direct human supervision, a pre-defined program, or a set of general guidelines, using artificial intelligence techniques. The word “robot” was coined by Czech playwright Karl Capek in his play R.U.R (Rossum’s Universal Robots), which opened in Prague in 1921. The word is derived from ‘robota’ which is the Czech word for forced labor. Later, the word usually used for human-made artificial creatures that can accomplish a task in which it is programmed to do.

Robots are designed for a multitude of applications. For example, in manufacturing, they are used for welding, riveting, scraping and painting. They are also deployed for demolition, fire and bomb fighting, nuclear site inspection, industrial cleaning, laboratory use, medical surgery, agriculture, forestry, and planet exploration. Increasingly, more artificial intelligence is being added. For example, some robots can identify objects in a pile, select the objects in the appropriate sequence and assemble them into a unit. As the robots become smarter, the tasks that they can accomplish become more complex. Today, some mobile robots are such smart that they can even play soccer which is one of the most complex activities that a humanbeing performs. It requires great vision, locomotion, localization, planning, and communication capabilities; however, locomotion is probably the most important capability for a mobile robot to achieve its goal as a soccer player.

The robots that have the ability to move is called mobile robots and there are many ways for making a robot mobile, one of which is using legs. Currently one popular research platform for legged locomotion is the Sony Aibo robot [Section 2.2] which is used in the annual RoboCup [1] 4-legged soccer competitions. In robot soccer domain, the speed of individual robots is a very important factor in determining the success of a team. Since the default gait developed by Sony for Aibo is fairly slow, there has been many research activities within the RoboCup community to develop improved locomotion for the Aibo robots.

Until recently, using hand-tuned parameters for improving locomotion capability of Aibo was a widely used technique. However, the process of hand-tuning a parameterized gait was time-consuming and required a good deal of human expertise. Furthermore, it was necessary to tune new parameter sets when the robot's hardware and/or the surface on which it is to walk changes. One alternative to hand-tuning a parameterized gait is to use machine learning techniques to automate the searching process for obtaining the best parameters.

In this work, there are two main objectives. First objective is to develop an inverse kinematics based, fully object-oriented, parametric motion engine for Aibo. Additionally, this engine includes a higher-level representation of walking action that deals with the trajectories of the Aibos four feet through three-dimensional space, which are called the *loci*. The second objective is to determine the necessary parameters for making an Aibo move smoothly, and obtaining the optimal values of these parameters by using Genetic Algorithms (GA) [Chapter 3].

1.1. Outline

A brief information about robot soccer, RoboCup [Section 2.1], and Sony Aibo robots [Section 2.2] is given as background in Chapter 2. Chapter 3 explains how Genetic Algorithms work. In Chapter 4, research done on quadruped locomotion so far is explained in detail. Chapter 5 explains our approach to the problem; how the kinematic model of the robot is constructed, what are the parameters to be used by our motion engine, and how these parameters are to be optimized by using GA techniques. Experiment environments and results obtained after the experiments are mentioned in Chapter 6. Finally, we conclude in Chapter 7 by summarizing the work done and pointing the possible future work.

2. BACKGROUND

In this chapter, a brief information about robot soccer, RoboCup organization, and specifications of a Sony Aibo robot is given.

2.1. Robot Soccer

Robot soccer is an example of such complex tasks for which multiple agents need to collaborate in an adversarial environment to achieve specific objectives. The main objective is scoring more goals than the opponent team and win the match as in real soccer. In order to achieve this objective, we must have a team consisting of good soccer players, and being a good soccer player requires some capabilities, such as, vision, locomotion, localization, planning, and communication [2].

- Vision: This module is responsible for information extraction from received camera frame. The vision process starts with receiving a camera frame and ends with a world model consisting of a collection of visual percepts.
- Locomotion: This module is responsible for all kind of movements of the robot. It includes all walking, dribbling, kicking actions and also some special actions such as cheer and sadness indicators.
- Localization: This process can be summarized as searching for the answer of the question "Where am I?".
- Planning: Planner is the decision making module of the robot and forms the highest level in the hierarchy. Robot acts according to the behaviors decided by the planner module.
- Communication: Communication module is responsible for inter-robot communication which uses messages that describe the states of each robot on the field.

RoboCup is an international joint project to promote AI, robotics, and related fields. It is an attempt to foster AI and intelligent robotics research by providing a standard problem where wide range of technologies can be integrated and examined.

RoboCup chose to use soccer game as a central topic of research, aiming at innovations to be applied for socially significant problems and industries. The ultimate goal of the RoboCup project is:

By 2050, develop a team of fully autonomous humanoid robots that can win against the human world champion team in soccer.

In order for a robot team to actually perform a soccer game, various technologies must be incorporated including: design principles of autonomous agents, multi-agent collaboration, strategy acquisition, real-time reasoning, robotics, and sensor-fusion. RoboCup is a task for a team of multiple fast-moving robots under a dynamic environment. RoboCup also offers a software platform for research on the software aspects of RoboCup. One of the major application of RoboCup technologies is a search and rescue in large scale disaster. RoboCup initiated RoboCupRescue project to specifically promote research in socially significant issues.

Currently, there exist a number of different RoboCup soccer leagues that focus on different aspects of this challenge. The Sony Four-Legged Robot League is one of them. In this league, teams consisting of four Sony Aibo robots each play on a field of 6 m x 4 m which is illustrated in Figure 2.1. The robots operate fully autonomously, i.e. there is no external control, neither by humans nor by computers [1].



Figure 2.1. Field setup for RoboCup Legged League [1].

2.2. Sony Aibo Robot

The Sony Aibo ERS-210 is a commercially available robot that is equipped with a color CMOS camera and an optional ethernet card that can be used for wireless communication. The Aibo is a quadruped robot, and has three degrees of freedom in each of its four legs. Totally it has 18 degrees of freedom with a continuous range of motion and a variety of other output mechanisms including sound and LEDs. Also an Aibo robot has various sensors such as an accelerometer, an infrared proximity sensor, a total of 8 touch sensors (two on the head which measures pressure in micro Newton, one on its chin, one on its back, and 4 under each paw), stereo microphones on the sides of its head, and joint position and load sensing mechanism in addition to its CMOS camera [3]. An ERS-210 has an 64bit RISC processor with a clock speed of 192 MHz, and an ERS-7 has an 64 bit RISC processor with a clock speed of 576 MHz. An ERS-210 and an ERS-7 is shown in Figure 2.2



Figure 2.2. ERS-210 and ERS-7.

An operating system named Aperios, which is developed by Sony, runs on Aibo robots. Aperios is organized as a set of cooperating processes, called objects in Aperios, which communicate via message passing, and it provides system services.

”OPEN-R” is the interface that Sony is promoting for the entertainment robot systems to expand the capabilities of entertainment robots. This interface is layered and optimized to enable efficient development of hardware and software for robots. The OPEN-R SDK discloses the specifications of the interface between the system layer and the application layer. These specifications are essential knowledge for developing robot software [4].

3. GENETIC ALGORITHMS

Genetic algorithms (GA) are a part of evolutionary computing, which is a rapidly growing area of artificial intelligence. GA are inspired by Darwin's theory of evolution; simply said, problems are solved by an evolutionary process resulting in a best (fittest) solution (survivor) - in other words, the solution is evolved. In every generation, a new set of artificial creatures (strings) is created using bits and pieces of the fittest of the old; an occasional new part is tried for good measure. They efficiently exploit historical information to speculate on new search points with expected improved performance [5].

GA works with strings which are called *chromosomes* and each chromosome represents an *individual*. Strings are constructed by arranging necessary parameters in a row. Every individual has a property called *fitness*, which indicates the strength of the individual, that is how strongly does it conform to the situation to be obtained, or in other words, the goodness of solution. Typically a higher fitness value represents a better solution.

A set of chromosomes is called a *population*. To form a new population (the next generation), individuals are selected according to their fitness. The simplest selection technique is the fitness-proportionate selection, where individuals are selected with a probability proportional to their relative fitness. This ensures that the expected number of times an individual is chosen is approximately proportional to its relative performance in the population. Thus, high-fitness good individuals stand a better chance of reproducing, while low-fitness ones are more likely to disappear. Producing new generations by applying operators of GA to previous generation iteratively continues until either a certain convergence level of the system is reached (a solution / a set of solutions with a fitness value over a certain threshold is produced) or a given iteration number is exceeded [5].

Parameters of a solution of the system are represented as a string of numbers

in GA. The most common representation method is to use binary numbers. In this approach, each parameter is one single bit. In real number / integer encoding, each gene is consist of a real number / integer. Different methods might give different results for different problems; hence, string representation method is problem dependent.

3.1. Operators

Reproduction, Crossover and Mutation are main three operators used in GA. These operators are used for creating new generations and finding an optimal solution by investigating each individual. The probabilities of these operators of being used affect the characteristics of the output very much.

3.1.1. Reproduction

Reproduction is a process in which individual strings are copied into the next generation according to their fitness function values. This process makes sure that the strings with a higher fitness function value have a higher probability of contributing one or more offspring in the next generation. This operator, of course, is an artificial version of natural selection, a Darwinian survival of the fittest among string creatures [5].

Probably the easiest way of implementing the reproduction operator is to create a biased roulette wheel where each current string in the population has a roulette wheel slot sized in proportion to its fitness. The wheel is "spun" and the marble falls into one of the slots. The wheel is spun to select each chromosome that is to mate. A sample roulette wheel is illustrated in Figure 3.1.

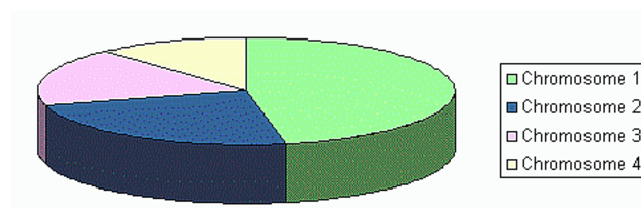


Figure 3.1. A sample roulette wheel for a generation having 4 chromosomes.

3.1.2. Crossover

The offspring individuals get their genetic information from their parents. Some parts of the genetic information of each parent are carried to their offspring after reproduction. It's provided that the individuals in each offspring carry some characteristics of their parents since new individuals are created by exchanging some parts of the parent chromosomes. This means that some portion of the characteristics from each parent is inherited to the child. This operation in the evolutionary process is called *crossover*. Figure 3.2 gives an example crossover operation on two solution strings consisting of integers.

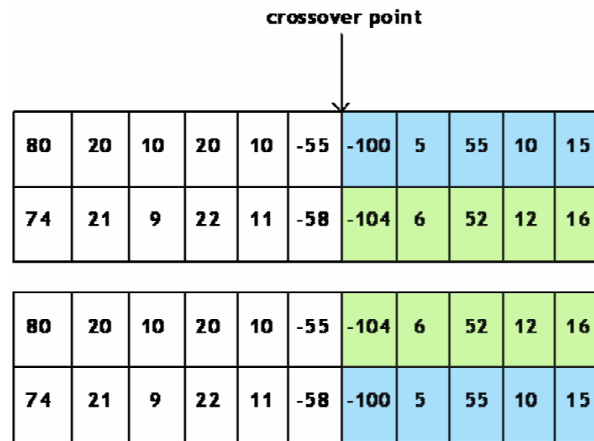


Figure 3.2. An example of crossover operation on two solution strings.

Figure 3.2 shows a sample single point crossover operation. Two new individuals are created by exchanging the parts of the chromosomes separated by the crossover point, and after that operation these new individuals carry the genetic information taken from both of the parents.

Crossover probability can be introduced to indicate how often crossover operation will be performed. If crossover probability is 100 per cent, then all offspring is made by crossover. If it is 0 per cent, whole new generation is made from exact copies of chromosomes from old population.

3.1.3. Mutation

Mutation is a genetic operator that alters one or more gene values in a chromosome from its initial state. This can result in entirely new gene values being added to the gene pool. With these new gene values, the GA may be able to arrive at better solution than was previously possible. Mutation is an important part of the genetic search as it helps to prevent the population from getting stuck at any local optima. Mutation occurs during evolution according to a user-definable *mutation probability*. It indicates how often the parts of a chromosome will be mutated. This probability should usually be set fairly low (0.01 is a good first choice). If mutation probability is 100 per cent, whole chromosome is changed, if it is 0 per cent, nothing is changed. Mutation is made to prevent falling GA into local extreme, but it should not occur very often, because then GA will change to random search.

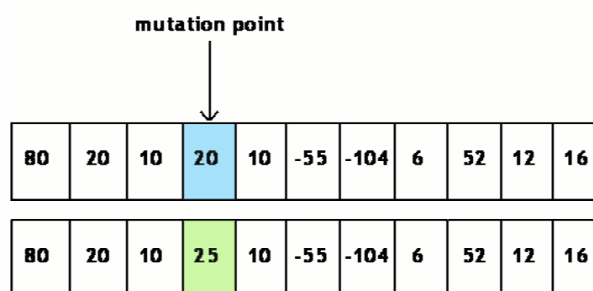


Figure 3.3. An example of a mutation operation on a solution string.

3.2. Fitness Function

A fitness function evaluates each solution to decide whether it will contribute to the next generation of solutions. In other words, the fitness value is an attribute related to an individual's strength or the probability of survival and reproduction of this individual. A fitness function is usually a form of the utility function of the system that is tried to be optimized.

3.3. Initial Population

The distribution of fitness values of individuals over population highly depends on the predecessor population since every generation is created from previous generation. However, the first generation to be the parent of all offsprings should initially be created in such a way that it will produce children having high fitness values. Therefore, creation of the initial population needs careful consideration. A pure random based approach can be used but it may produce individuals having quite low fitness values due to the nature of randomness. This situation can be removed by creating the individuals of initial population not only randomly but also based on some information about solution space; that is, by using some heuristics. However, the level of heuristics should be considered carefully too because creating individuals that are very similar to each other according to their fitness values may lead the system to premature convergence or local optima.

3.4. Stopping Criteria

Stopping criteria are the criteria that the GA engine decides stopping. There can be three stopping criteria:

- Stopping after a certain number of generations
- Stopping after founding global optimum solution
- Stopping in the case of the change in the convergence of the system is below a given threshold value in the last n generations.

Each of these criteria can be used alone as well as a combination of more than one of them.

3.5. Summary

Given a clearly defined problem to be solved and a bit string representation for candidate solutions, a simple GA works as follows:

1. Start with a randomly generated population of n l -bit chromosomes (candidate solutions to a problem).
2. Calculate the fitness (x) of each chromosome x in the population.
3. Repeat the following steps until n offspring have been created:
 - a. Select a pair of parent chromosomes from the current population, the probability of selection being an increasing function of fitness. Selection is done "with replacement," meaning that the same chromosome can be selected more than once to become a parent.
 - b. With probability p_c (the "crossover probability" or "crossover rate"), cross over the pair at a randomly chosen point (chosen with uniform probability) to form two offspring. If no crossover takes place, form two offspring that are exact copies of their respective parents. (Note that here the crossover rate is defined to be the probability that two parents will cross over in a single point. There are also "multi-point crossover" versions of the GA in which the crossover rate for a pair of parents is the number of points at which a crossover takes place.)
 - c. Mutate the two offspring at each locus with probability p_m (the mutation probability or mutation rate), and place the resulting chromosomes in the new population. If n is odd, one new population member can be discarded at random.
4. Replace the current population with the new population.
5. Go to step 2.

Each iteration of this process is called a generation. A GA is typically iterated for anywhere from 50 to 500 or more generations. The entire set of generations is called a run. At the end of a run there are often one or more highly fit chromosomes in the population. Since randomness plays a large role in each run, two runs with different random-number seeds will generally produce different detailed behaviors [6].

4. RELATED WORK

At the lowest level, the Aibo's gait is determined by a series of joint positions for the three joints in each of its legs. Hornby et al. [7] used GA to learn a set of low level parameters that described joint velocities and body position to develop a gait as an early attempt. More recently, trajectories of the Aibos four feet through three-dimensional space have been used to develop a higher level representation for Aibo's gait. An inverse kinematics calculation is then used to convert these trajectories into joint angles. Among higher-level approaches, most of the differences between gaits that have been developed for the Aibo stem from the shape of the loci through which the feet pass and the exact parameterizations of those loci. For example, a team from the University of New South Wales achieved the fastest known hand-tuned gait using the high-level approach described above with trapezoidal loci. They subsequently generated an even faster walk via learning [8]. A team from Germany created a flexible gait implementation that allows them to use a variety of different shapes of loci [9].

Among these approaches, ParaWalk, which is developed by rUNSWift 4-legged robot soccer team of UNSW (The University of New South Wales), is the first walking routine that uses parameters for describing the walking action. It is a very flexible and robust walking routine and operates on a set of supplied parameters such as the height of the chest of the robot from the ground, the number of waypoints needed to complete a full step cycle, etc. These parameters can be defined in such a way to produce particular stances and walking styles.

In this chapter, the structure of ParaWalk is explained.

4.1. Kinematic Model

Kinematic model of an Aibo leg designed by rUNSWift approximates each segment of the leg to be a right triangle as shown in Figure 4.1 and this makes the necessary calculations more complex. Also the coordinate system they used is kind of different;

according to their coordinate system, x axis grows towards front of the robot, y axis grows towards the ground, and z axis grows towards the left side of the robot, parallel to the ground.

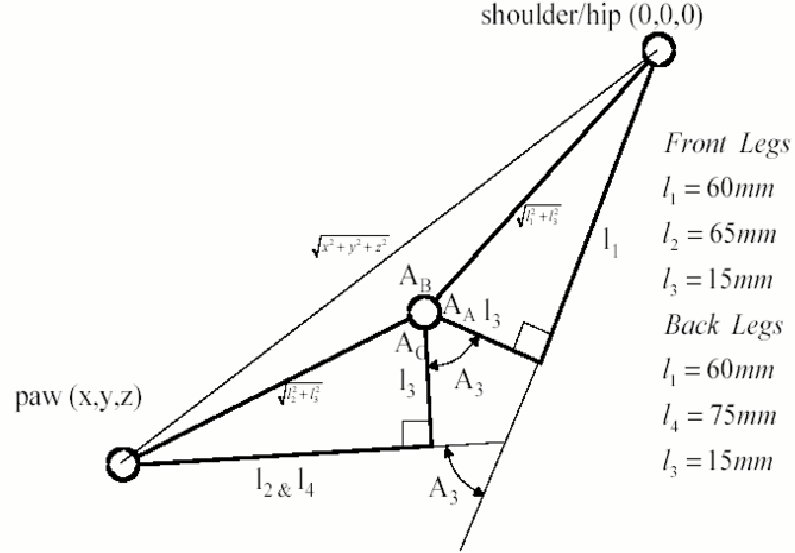


Figure 4.1. Kinematic model of an Aibo leg developed by rUNSWift [10].

4.2. Parameters

Walking parameters used by rUNSWift can be listed as;

- Step duration related
 - PG: Number of steps needed to complete one full step (i.e. the paw comes back to its initial position).
- Rectangular locus related
 - hdF: Height of the rectangle used as front locus from the ground.
 - hdB: Height of the rectangle used as back locus from the ground.
- Initial paw locations related
 - hF: Height of the chest of the robot from ground.
 - hB: Height of the back of the robot from ground.
 - fs0: Sideway distance of the paws of the front legs from shoulder.
 - ff0: Forward distance of the paws of the front legs from shoulder.

bs0: Sideway distance of the paws of the rear legs from shoulder.

bf0: Forward distance of the paws of the rear legs from shoulder.

The default values of these parameters are shown in Table 4.1.

PG	use 65 as default. Can be any +ve multiple of 5.
hF	use 73 mm as default. Limited by physical constraints.
hB	use 97 mm as default. Limited by physical constraints.
hdF	use 16 mm as default. Limited by physical constraints. Must be ≥ 0
hdB	use 19 mm as default. Limited by physical constraints. Must be ≥ 0
ff0	use 33 mm as default. Limited by physical constraints.
fs0	use 20 mm as default. Limited by physical constraints.
bf0	use -35 mm as default. Limited by physical constraints.
bs0	use 20 mm as default. Limited by physical constraints.

Table 4.1. Default values of the walking parameters used by ParaWalk [11].

4.3. Locus

ParaWalk initially used rectangular locus. Rectangular locus means that the paw draws a rectangle on the air when performing the walking action. In ParaWalk, the heights of forward and back locus are included in the parameter list. Since the widths of rectangles are calculated according to the locus limits, they are not needed to be used as separate parameters. Simple representation of rectangular locus used by rUNSWift is shown in Figure 4.2.

After using some machine learning techniques, they obtained some kind of trapezoidal loci. These loci have dramatically changed the walking performance of the robot. Resulting shapes are shown in Figure 4.3.

4.4. Walking Styles

To produce a walking motion, the legs must not be at the same position on the walk locus at the same time. Essentially, the legs must move out of phase of each other.

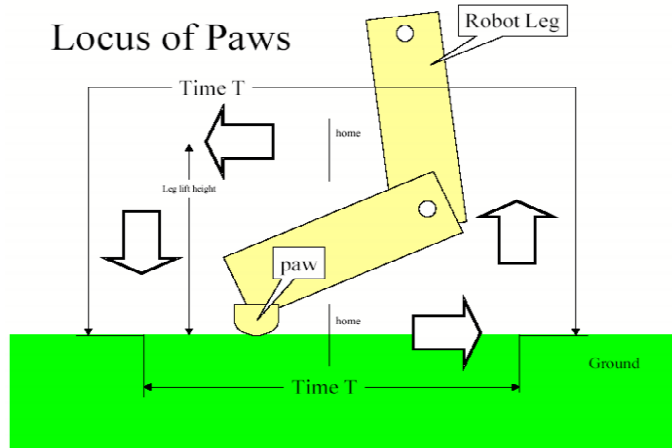


Figure 4.2. Rectangular locus used by rUNSWift [10].

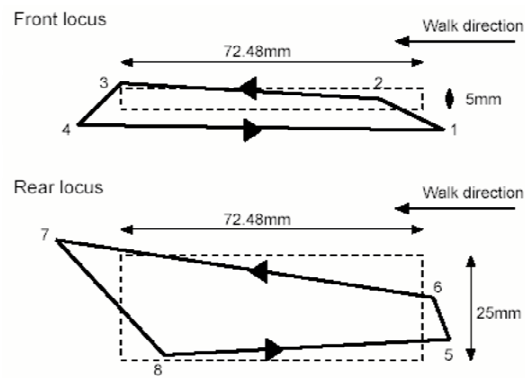


Figure 4.3. The loci produced by minimization technique in the sagittal plane, shown in bold. The dotted lines show the basic rectangular loci.

Human walking actually uses a similar approach. One leg is lifted, moved forward, and then dropped, while the other stays where it was. Once the first step has been taken, the other leg is then lifted and basically mirrors the same action taken by the first leg.

Different gait types can be obtained by shifting the movement phases of each leg in different manners. Timing of each leg and resulting walking type is shown in Figure 4.4

















Quadruped Gaits				
	Crawl	Trot	Pace	Bounding
front right				
front left				
rear right				
rear left				

Figure 4.4. Different timing of each legs motion results in different walking styles [10].

There are various stance types. In Sony's default walking style, high stance is used in which the robot stands on its paws. However, ParaWalk uses low stance in which the robot would have the front forelegs flat on the ground when walking, while the rear legs walked "normally" on paws [10]. Low stance causes a body tilt which promotes the momentum of the robot while walking forwards. Sony's default high stance and ParaWalk's preferred low stance are illustrated in Figure 4.5.

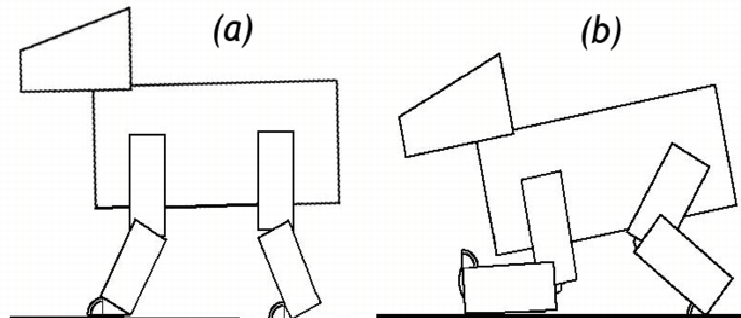


Figure 4.5. (a) High stance and (b) low stance.

4.5. Omnidirectional Motion

Omnidirectional walking can be thought as the motion of a shopping cart, and can be obtained by treating the legs as wheels. This is illustrated in Figure 4.6.

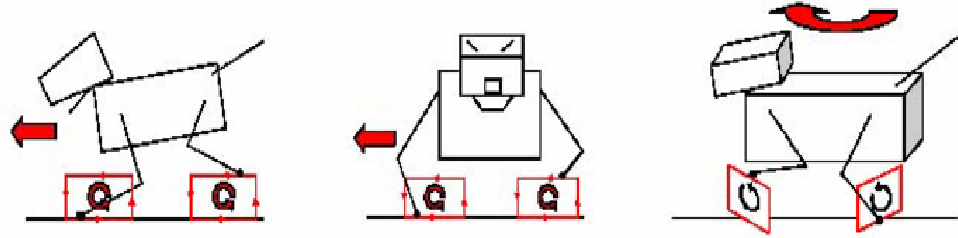
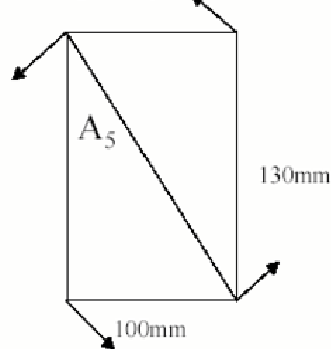


Figure 4.6. Using legs as the wheels of a shopping cart [12].

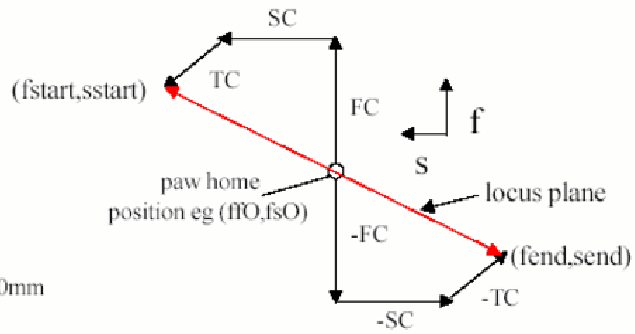
In order to achieve this motion, 3 walk components named *forward*, *sideways*, and *turn* are used. These components are represented as 2-dimensional vectors and they are added vectorally in order to obtain one resulting vector and its symmetric part according to the initial paw location. These two resulting vectors together reproduce the limits of the locus; that is the limits of each leg's area of operation. The operation of obtaining locus limits by using forward, sideways, and turn components is illustrated in Figure 4.7.

Control = forward FCmm, sideways SCmm, turn TCmm

Plan View
(TC component)



$$A_5 = \text{atan} \frac{100 \text{ mm}}{130 \text{ mm}}$$



$$fstart = FC - TC \sin A_5 + ffO$$

$$fend = -FC + TC \sin A_5 + ffO$$

$$sstart = SC + TC \cos A_5 + fsO$$

$$send = -SC - TC \cos A_5 + fsO$$

Figure 4.7. Forward, sideways, and turn components are added vectorally to obtain a resulting vector; which indicates the direction and limit of the paw movement [11].

5. PROPOSED APPROACH

In this chapter, our proposed approach, including kinematic model, parameter determination and description, locus representation, and parameter optimization by using GA is explained in detail.

5.1. Kinematic Model

Kinematics is the study of how things move. In particular, we want to have methods for precisely calculating the position of points on a manipulator (forward kinematics). Further, we need to know which joint values will achieve a desired position (inverse kinematics) in order to produce controlled motions.

Manipulators are constructed from a series of links, each connected by an actuator. A series of links is also called a chain, with one end defined as the base, and the other called the end effector. Actuators are commonly either revolute joints which cause rotation about an axis, or prismatic joints, which cause translation along an axis. All of the AIBO's joints are revolute [13].

5.1.1. Inverse Kinematics

The inverse kinematics problem is much more interesting and its solution is more useful. At the position level, the problem is stated as, "Given the desired position of the robot's paw, what must be the angles at all of the robots joints?".

In our approach, inverse kinematics techniques are used to calculate the desired joint angles while the paw is moving along the path determined by the locus of the leg. The locus is divided into *pStep* (to be explained in Section 5.4) points, and each of these points has (x,y,z) values. When the paw is to move to the location of the next point on the locus, the following formulas are used to calculate the necessary joint angles in order to make the paw move towards this point.

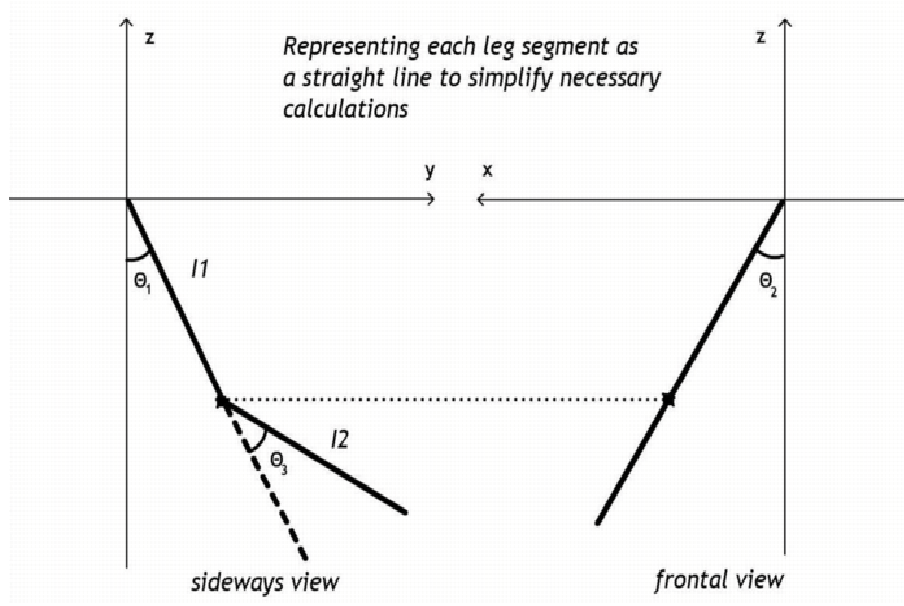


Figure 5.1. Simple kinematic model representation for front right leg.

The law of cosines is used for calculating the knee angle (θ_3).

$$d^2 = x^2 + y^2 + z^2$$

$$I_1^2 + I_2^2 - 2I_1I_2 \cos(\pi - \theta_3) = d^2$$

$$\theta_3 = \pi - \arccos\left(\frac{I_1^2 + I_2^2 - d^2}{2I_1I_2}\right)$$

Then the abductor angle θ_2 is calculated.

$$\theta_2 = \arcsin\left(\frac{x}{I_1 + I_2 \cos(\theta_3)}\right)$$

Finally, rotator angle θ_1 is calculated.

$$\theta_1 = \frac{y \cos(\theta_2)(I_1 + I_2 \cos(\theta_3)) + z I_2 \sin(\theta_3)}{y I_2 \sin(\theta_3) - (z \cos(\theta_2)(I_1 + I_2 \cos(\theta_3)))}$$

5.2. Walking

Trot gait, in which the diagonally opposed legs are synchronized, is used as the primary gait type. Omnidirectional motion is inherited from ParaWalk as it is. The only difference is the meaning of sideways and turn components. In ParaWalk, default sideways direction is leftwards, and default turn angle increases counterclockwise. In our approach, default sideways direction is rightwards, and default turn angle increases clockwise. Resulting motions for different combinations of walk components is shown in Figure 5.2.

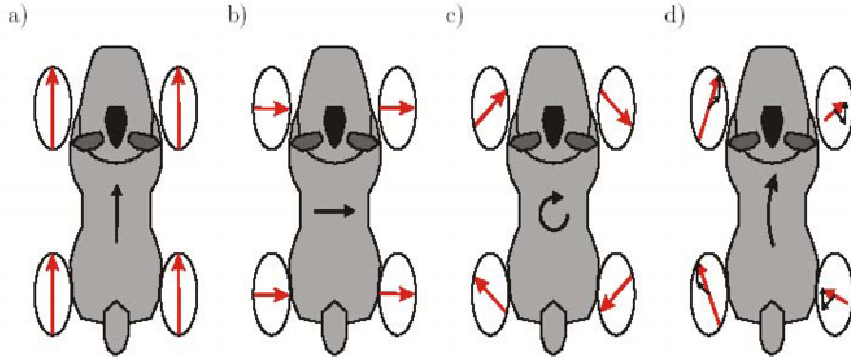


Figure 5.2. Resulting motions with different combinations of forward, sideways, and turnCW parameters: (a) only forward, (b) only sideways, (c) only turnCW, (d) forward and turnCW together [10].

5.2.1. Representing the Locus

According to the research done so far, rectangular, trapezoidal and half elliptic loci are not effective; in fact they have a hindering effect on robot's movement. Especially the movement of rear is the cause of this effect. While performing these kinds of movements the leg touches the ground in the same direction of the movement, which in turn decreases the robot's momentum at that time.

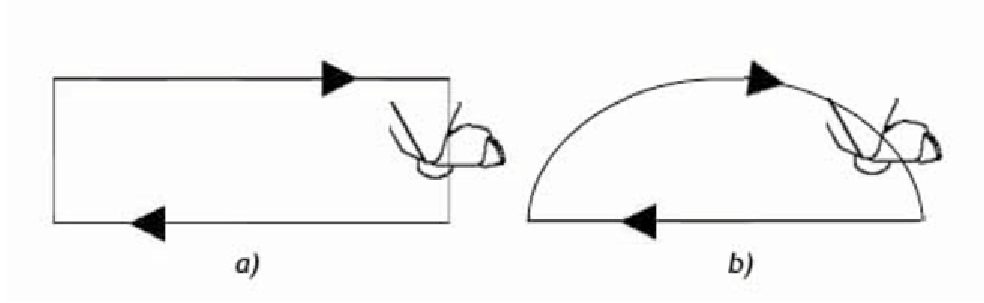


Figure 5.3. Movement of the paw on a (a) rectangular locus and a (b) half elliptic locus.

Proposed locus is in the shape of an ellipse cut from below in some proportion. This shape can be approximated by a hermite curve and it is illustrated in Figure 5.4.

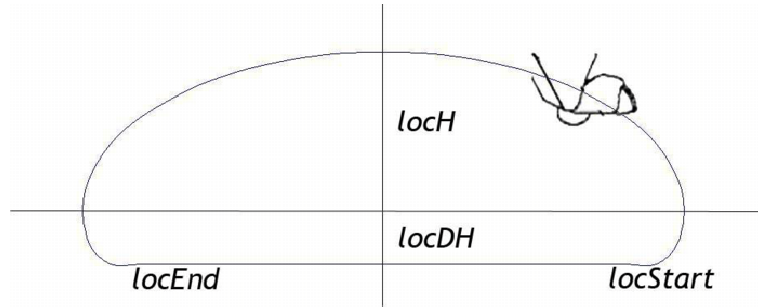


Figure 5.4. Proposed locus in the shape of a hermite curve.

With the introduction of elliptic locus, this effect is avoided since the leg touches the ground after moving in the reverse direction of the movement for a short period of time. This movement type guarantees that the moment of the robot is not hindered but increased. Also, elliptic locus makes the movement of the leg smoother.

5.3. Object-oriented design of the locomotion module

Locomotion module is designed by using an object-oriented approach. First of all, robot is thought as a single object composed of many other objects. Specifically, an AIBO robot physically consists of four legs, a head, and a tail, each of which carries different number of joints. Each Leg has three Joints, which are the rotator, the abductor, and the knee joints. The Head has three Joints, which are pan, tilt, and roll joints. Finally, the Tail has two Joints, which are pan and tilt joints. All these objects are defined as a separate class. The classes used for the locomotion module is shown

in Figure 5.5.

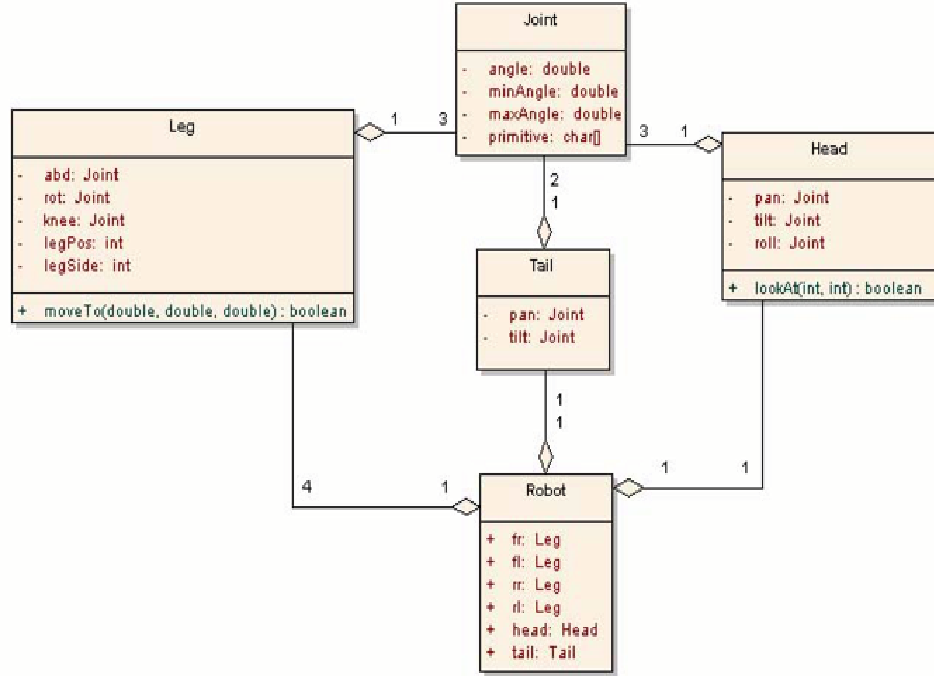


Figure 5.5. Class diagram showing the relations between classes used in the locomotion module.

Leg class has a method named *moveTo* for calculating the required joint angles to be able to reach a specific point in a 3-dimensional space. It performs the aforementioned inverse kinematics calculations and determines the knee, abductor, and rotator angles of the leg, respectively.

Besides these robot related classes, there are two very important classes. One is MotionManager class, which is responsible for the coordination of all movements, and the other is GA class, which is responsible for generating an initial population according to the sample string provided, and then performing the main GA operations (reproduction, crossover, mutation) on each population in order to generate parameter lists to be used during experiment processes.

5.4. Parameter Optimization

There are 11 parameters used by the new walking engine. These parameters can be categorized as step duration related, locus related, and initial paw locations related parameters.

- Step duration related

pStep: Number of steps needed to complete one full step (i.e. the paw comes back to its initial position).

- Locus related

fLocH: Height radius of the ellipse to be used as the locus for front legs.

fLocDH: Perpendicular distance of the center of the ellipse of the front locus from the initial paw location.

bLocH: Height radius of the ellipse to be used as the locus for rear legs.

bLocDH: Perpendicular distance of the center of the ellipse of the rear locus from the initial paw location.

- Initial paw locations related

hF: Height of the chest of the robot from ground.

hB: Height of the back of the robot from ground.

fs0: Sideway distance of the paws of the front legs from shoulder.

ff0: Forward distance of the paws of the front legs from shoulder.

bs0: Sideway distance of the paws of the rear legs from shoulder.

bf0: Forward distance of the paws of the rear legs from shoulder.

GA is used for optimization of these parameters. Initially, a set of hand-tuned parameters are given to the GA engine, and it produces some predefined number of chromosomes by distorting the parameter values on the sample string to obtain a generation. Structure of our parameter set and a sample string are shown in Figure 5.7 and Figure 5.8, respectively.

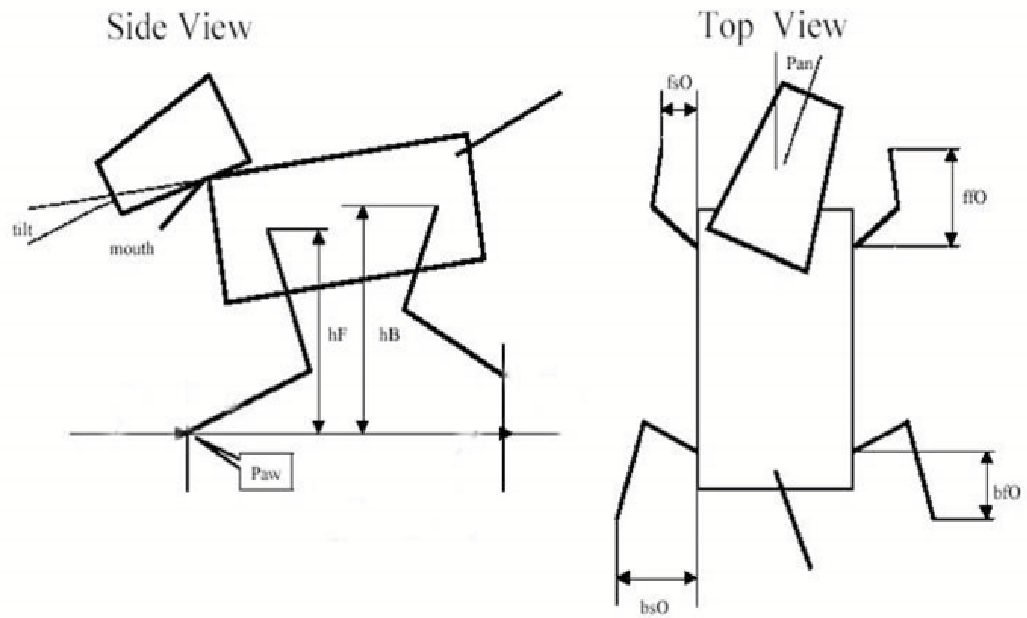


Figure 5.6. Parameters related to initial paw locations [10].

pStep	fLocH	fLocDH	bLocH	bLocDH	hF	hB	fsO	ffO	bsO	bfO
-------	-------	--------	-------	--------	----	----	-----	-----	-----	-----

Figure 5.7. String representation of walking parameters.

80	20	10	25	10	-55	-104	6	52	12	16
----	----	----	----	----	-----	------	---	----	----	----

Figure 5.8. A sample string.

Once a generation is constructed, the process begins. GA engine starts with the first chromosome and replaces the current parameter values on the robot side with these new values taken from this chromosome and the robot starts walking. After the robot completes 8 steps, the fitness of this parameter set (chromosome) is calculated according to our fitness function. Since the main objective of this process is to obtain the optimal parameter set that provides the faster walking without any diversion, our fitness function is constructed in such a way that it would promote moving straight forward, and punish either rotational or sideways diversion. Our fitness function is

$$fitness = 0.9 * fwd - 0.4 * sdwDiv - 0.4 * rotDiv$$

where *fwd* is total forward distance reached, *sdwDiv* is sideways diversion, and *rotDiv* is rotational diversion. When all the chromosomes are tried and associated with a fitness value, the reproduction process begins. During reproduction process, a random number between zero and total fitness values of all chromosomes is generated and a chromosome is selected by using the roulette wheel technique mentioned in Chapter 3. After that second chromosome is selected similarly. Then these chromosomes exchanges some patterns according to a crossover probability rate and generates two new chromosome to be copied into the new generation. Also, during crossover process, some parameter values can be changed according to a mutation probability rate.

However, there are some differences between our approach and simple GA according to the techniques used for performing crossover and mutation operations.

- Crossover

In our approach, there are more than one crossover point determined by the range of parameter category; that is locus related parameters are exchanged in their own range, and initial paw locations related parameters are exchanged in their own range. These clusters are shown in Figure 5.9. Before performing crossover operation, one of the

three clusters is selected, and then the crossover operation is performed within this cluster as shown in Figure 5.10.

- Mutation

In mutation operation, the amount of distortion is determined according to the fitness value of this chromosome. That is, if the fitness values is small, the amount of distortion is greater, and if the fitness value is high, which means that this parameter set is good enough, then the amount of distortion is smaller.

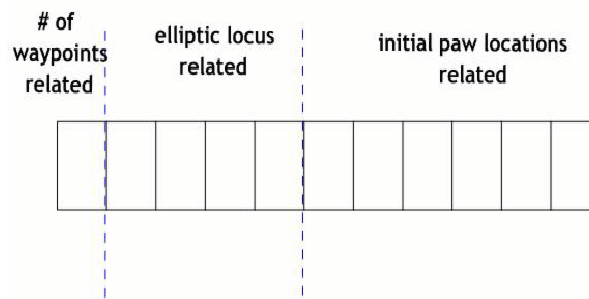


Figure 5.9. A sample chromosome separated into three parts. These parts are related to number of waypoints on the locus, shape of the front and back loci, and initial paw locations, respectively.

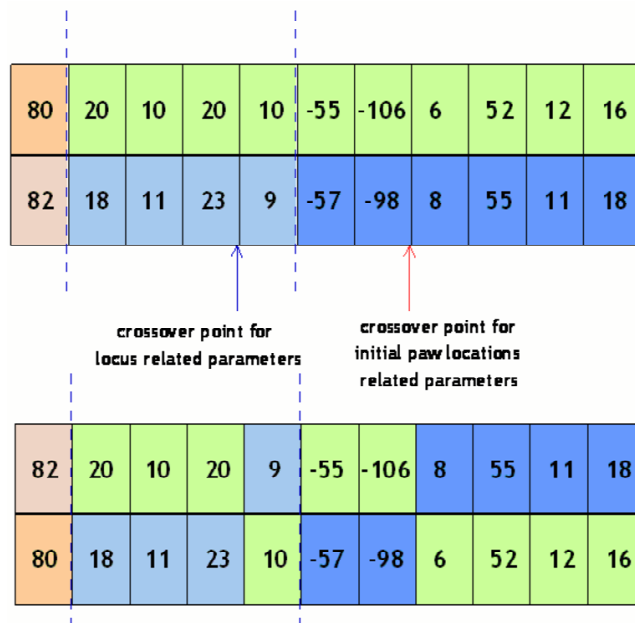


Figure 5.10. Technique used for the crossover operation in our approach.

```

initially construct a sample string
generate a population by distorting the sample string
while currentGenerationIndex < MAX_NUM_OF_GENERATIONS
  for each chromosome in current population
    set the values carried by this chromosome as walking parameters
    totalFitness = 0
    for 1 to AVG_COUNTER
      make the robot move forward 8 steps
      totalFitness += the fitness of this parameter set (chromosome)
    current chromosome's fitness = totalFitness / AVG_COUNTER
currentGenerationIndex ++

```

Figure 5.11. Pseudo-code implementation of the proposed approach.

As a summary, a pseudocode representation of the parameter optimization process is provided in Figure 5.11.

6. EXPERIMENTAL STUDY

In this chapter, stages of experimental study is explained in detail and software and hardware platforms used during this project is mentioned.

6.1. Environments

6.1.1. Webots

Webots is a mobile robotics simulation software that provides a rapid prototyping environment for modelling, programming and simulating mobile robots. The included robot libraries enable transferring control programs to many commercially available real mobile robots including the Sony Aibo ERS-210 robot [14].

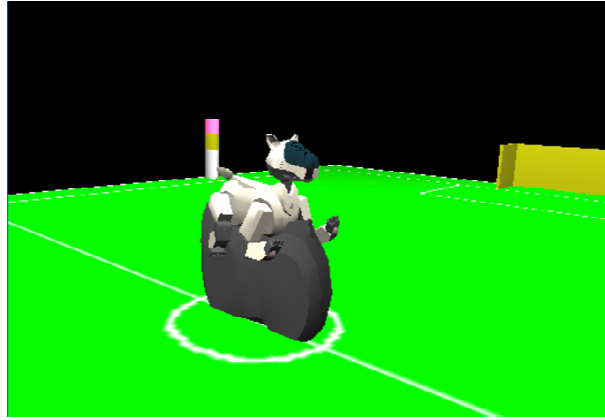


Figure 6.1. A screenshot from Webots simulator environment showing an Aibo ERS-210 on its walking test platform.

Initially a hand-tuned parameter set is created and a small portion (i.e. 25 per cent) of the initial generation is constructed by slightly distorting the parameter values of the sample string. The remaining part of the generation is constructed by using much more distorted individuals. After that the training process begins. Each chromosome is tried for a pre-defined number of times (called AVG_COUNTER in our GA engine) and the average fitness value obtained from these trials is assigned as the fitness value of the current chromosome as shown in Figure 6.2. This process continues until a

maximum number of generations is reached.

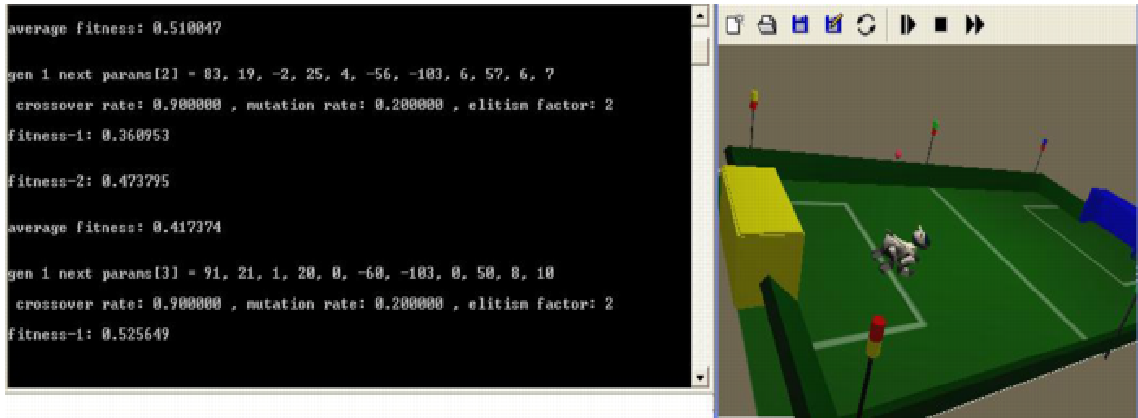


Figure 6.2. A screenshot from the simulator environment during the training process.

6.2. Results

Many experiments were made with different number of generations, different number of individuals in a generation, and different crossover and mutation rates. Figure 6.3 shows the result of an experiment in which 30 generations each having 50 individuals are used with a crossover rate of 80% and a mutation rate of %10.

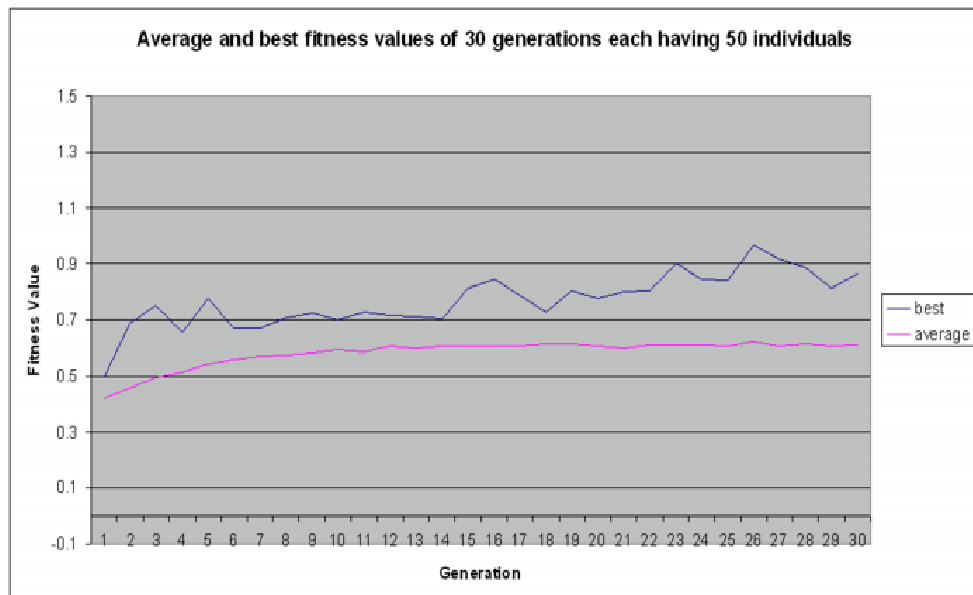


Figure 6.3. Average and best fitness values of 30 generations each having 50 individuals.

In Figure 6.4 the result of an experiment in which 50 generations each having 100 individuals are used with a crossover rate of 90% and a mutation rate of %5 is shown.

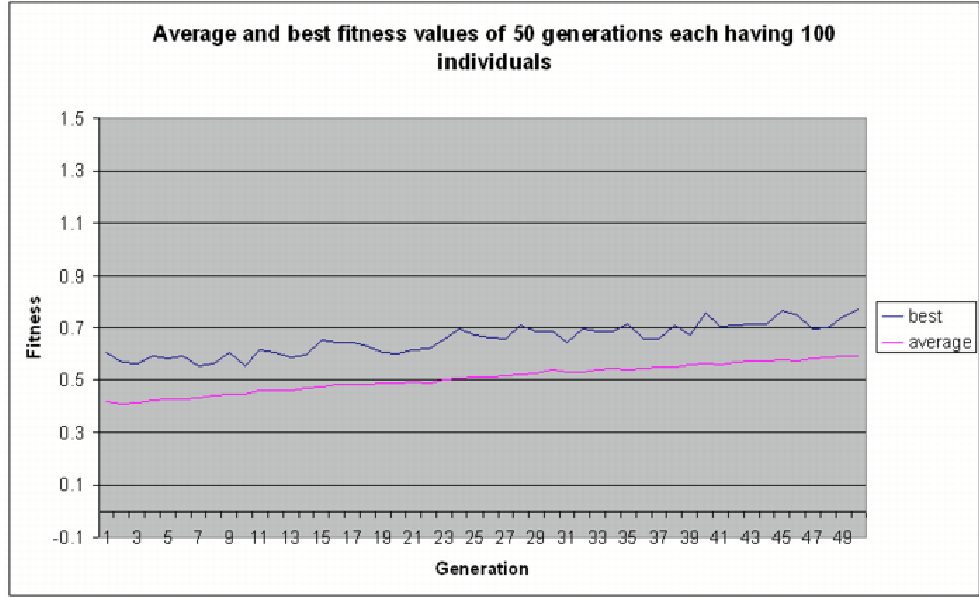


Figure 6.4. Average and best fitness values of 50 generations each having 100 individuals.

Both of these sample results show that GA works quite well since both the best and the average fitness values show increasing tendencies. The rippling effect seen on best fitness values is a result of the randomness of the simulator environment, that is when the same parameter set is tried in the next generation, it may have a different fitness value.

After these experiments, the learned parameter set is shown in Figure 6.5. Fine-tuning this parameter set on a real robot is left as a future work.

84	19	7	18	12	-56	-100	6	51	14	17
-----------	-----------	----------	-----------	-----------	------------	-------------	----------	-----------	-----------	-----------

Figure 6.5. Resulting parameter set that provided best fitness value.

Finally, Table 6.1 shows the performances of different walking engines, and improvement provided by our proposed engine.

Walking Engine	Performance
Default Sony-type	≈ 4 cm/sec
ParaWalk	≈ 21 cm/sec
Our engine (hand crafted)	≈ 25 cm/sec
Our engine (optimized)	≈ 27 cm/sec

Table 6.1. Walking engine vs. Performance.

7. CONCLUSION & FUTURE WORK

Today robots are being used in several important endeavors; industry, medical surgery, agriculture, forestry, and planet exploration. As the robots become smarter, the tasks that they can accomplish become more complex. Recently, some mobile robots are such smart that they can even play soccer which is one of the most complex activities that a humanbeing performs. It requires some capabilities such as vision, locomotion, localization, planning, and communication; however, locomotion is probably the most important one for a mobile robot to achieve its goal as a soccer player.

In this work a new inverse kinematics based, fully object-oriented, and parametric motion module for Aibo robot is presented. GA is used for parameter optimization process. Since the aim of this project is to make Aibo move faster, this problem can be thought as a maximization problem; i.e. our primary aim is to maximize the fitness function value.

Some possible extensions that this project may have can be listed as:

- Some special static actions (such as kicking actions) can also be parametrized and these parameters can be fine-tuned by using GA.
- Omnidirectional motion can be optimized for obtaining better forward, sideways, and turning motion and also the motion resulting from the combination of these three components.
- Fitness function can be modified in such a way to promote a motion type that would keep the head more stable in order to get less distorted camera image.
- Resulting parameter set can be fine-tuned on a real robot walking on a real field.

APPENDIX A: Code Used in Parameter Optimization Process

GA code

```
#pragma once
#include <math.h>
const int STR_LEN = 11;
const int MAX_POP = 100;
const int MAX_GEN = 50;
const int ELITISM_FACTOR = MAX_POP / 20;
const float CROSSOVER_RATE = 0.9;
const float MUTATION_RATE = 0.2;
const float AVG_COUNTER = 2;

typedef struct
{
    int chromo[ STR_LEN ];
    double fitness;
} individual;

class GA
{
public:
    GA( void );
    individual population[ MAX_POP ];
    individual tmpipop[ MAX_POP ];
    int genCount;
    int currentInd;
    void crossover( individual ind1 , individual ind2 );
    void mutate( individual ind );
    individual select( void );
    double totalFitness( void );
    void initPopulation( int sample[] );
    void resetPopulation( void );
    individual nextIndividual( void );
```

```

float fitness( float last[] , float first[] );
void setFitness( int indIdx , float value );
individual highestFitness( void );
void sort( void );

~GA( void );
};

```

Supervisor code

```

if( standStill )
{
    if( error || ga->genCount == MAX_GEN )
    {
        if( supervisor_node_was_found(robot) )
        {
            // Write result to files
            supervisor_simulation_quit();
        }
    }
    else if( loop % (mm->pstep * 8) == 0 )
    {
        if( supervisor_node_was_found(robot) )
        {
            supervisor_field_get(robot,
                SUPERVISOR_FIELD_TRANSLATION_X|
                SUPERVISOR_FIELD_TRANSLATION_Z|
                SUPERVISOR_FIELD_ROTATION_ANGLE,
                &position,SIMULATION_STEP);

            fit = ga->fitness(position,robot_initial_position);
            fitness += fit;
            avgCounter++;
            printf("fitness-%d: %lf\n\n", avgCounter , fit);

            supervisor_field_set(robot,
                SUPERVISOR_FIELD_TRANSLATION_X|

```

```

SUPERVISOR_FIELD_TRANSLATION_Z|
SUPERVISOR_FIELD_ROTATION_ANGLE,
robot_initial_position);

if( fit <= 0.0f )
{
    error = true;
}
else if( ga->currentInd == MAX_POP - 1 && avgCounter == AVG_COUNTER )
{
    for( int i = 0; i < STR_LEN; i++ )
    {
        // append best, worst and average chromosome values
        // to corresponding strings to be copied into files
    }
    if( avgCounter == AVG_COUNTER )
    {
        ga->setFitness( ga->currentInd , (fitness / avgCounter));
        printf("average fitness: %lf\n\n", (fitness / avgCounter));
        avgCounter = 0;
        fit = 0;
        fitness = 0;
        i = ga->nextIndividual();
        mm->setParams(/* current gene values */);
    }
}
loop = 1;
}
mm->calcJointValues();

```

REFERENCES

1. Robocup Organization
”<http://www.robocup.org>”
2. H. L. Akın, H. Köse, Ç Meriçli, K. Kaplan, B. Çelik and T. Meriçli
Cerberus’05 Team Description Paper
3. Sony Aibo
”<http://www.sony.net/Products/aibo/>”
4. OPEN-R SDK
”http://openr.aibo.com/openr/eng/no_perm/faq-openrsdk.php4”
5. Goldberg, David E.
”Genetic Algorithms in Search, Optimization, and Machine Learning” Addison-Wesley 1989
6. Mitchell, Melanie
”An Introduction to Genetic Algorithms” MIT Press 1996
7. Hornby, G.S., Fujita, M., Takamura, S., Yamamoto, T. and Hanagata, O.
Autonomous Evolution of Gaits with the Sony Quadruped Robot. 1999
8. Min Sub Kim, William Uther
Automatic Gait Optimisation for Quadruped Robots
School of Computer Science and Engineering, University of New South Wales
9. Thomas Röfer
Evolutionary Gait-Optimization Using a Fitness Function Based on Proprioception
Center for Computing Technology (TZI), FB 3, Universität Bremen
10. UNSW 2003 team report

"<http://www.cse.unsw.edu.au/robocup/report2003.pdf>"

11. UNSW 2000 team report

"<http://www.cse.unsw.edu.au/robocup/2002site/2000PDF.zip>"

12. Bernhard Hengst, Darren Ibbotson, Son Bao Pham, Claude Sammut
Omnidirectional Locomotion for Quadruped Robots
School of Computer Science and Engineering, University of New South Wales

13. Exploring Tekkotsu Programming on the Sony AIBO

"<http://www-2.cs.cmu.edu/dst/Tekkotsu/Tutorial/forwardkin.shtml>"

14. The Webots mobile robotics simulation software

"<http://www.cyberbotics.com/products/webots/>"