Universidad
Carlos III de Madrid

TESIS DOCTORAL

LOCOMOTION THROUGH MORPHOLOGY, EVOLUTION AND LEARNING FOR LEGGED AND LIMBLESS ROBOTS

Autor:
Avinash Ranganath
Director:
Luis Moreno Lorente

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA
Leganés, October 4, 2016

TESIS DOCTORAL (DOCTORAL THESIS)

LOCOMOTION THROUGH MORPHOLOGY,
EVOLUTION AND LEARNING
FOR LEGGED AND LIMBLESS ROBOTS

Autor (Candidate): Avinash Ranganath

Director (Adviser): Luis Moreno Lorente

Tribunal (Review Committee)

Firma (Signature)

Presidente (Chair): _____

Vocal (Member): _____

Secretario (Secretary): _____

Título (Degree): Doctorado en Ingeniería Eléctrica, Electrónica y Automática
Calificación (Grade): _____

Leganés, de de

To my loved ones

“Nothing grows in the shadow of want without the sunlight of acknowledging your fullness.”

— Bryant McGill

Acknowledgments

Being a PhD student is anything but an easy task, and without the right support system, guidance and encouragement, I would never have reached the finish line by myself.

First and foremost, I would like to thank Prof. Luis Moreno, who has supported my since day one, in every aspect. Thank you for guiding me in the right direction, for being very open and cordial, and for encouraging me to pursue my interests even during difficult times.

I would like to thank the concerned people at the Department of Systems Engineering and Automation, University Carlos III of Madrid, who accepting my application and inviting me to be part of the department.

Angela, Fernando, Eduardo and Sonia; you have all been very cordial and patient with me throughout the years, whenever I have bugged you with hardware and logistics requests. Thank you all a lot.

It has been a wonderful experience working along side Mohamed, Alberto Brunete, Alvaro Castro, Fares and Dolores, among many others over the years. Alvaro, you made we fee very welcome and put a shelter over my head, when I first move to Madrid. Thank you very much for that!

Arnaud and Ainara ("iNara") you both have been my true mates through the years. It would not have been the same without you both. David; you has been very kind to me. My submissions in Spanish would not have been the same without your help. Thank you.

My parents and my sister have been my biggest supports through the years. They have helped my realize my dream, and I cannot thank them enough.

Jolla, there are no words to express how easy you made this journey feel. You are an integral part of it too.

And many thanks to the many others that helped at one or another point of the road.

Thank you to all.

Abstract

Robot locomotion is concerned with providing autonomous locomotion capabilities to mobile robots. Most current day robots feature some form of locomotion for navigating in their environment. Modalities of robot locomotion includes: (i) aerial locomotion, (ii) terrestrial locomotion, and (iii) aquatic locomotion (on or under water). Three main forms of terrestrial locomotion are, legged locomotion, limbless locomotion and wheel-based locomotion. A Modular Robot (MR), on the other hand, is a robotic system composed of several independent unit modules, where, each module is a robot by itself. The objective in this thesis is to develop legged locomotion in a humanoid robot, as well as, limbless locomotion in modular robotic configurations. Taking inspiration from biology, robot locomotion from the perspective of robot's morphology, through evolution, and through learning are investigated in this thesis.

Locomotion is one of the key distinguishing characteristics of a zoological organism. Almost all animal species, and even some plant species, produce some form of locomotion. In the past few years, robots have been "moving out" of the factory floor and research labs, and are becoming increasingly common in everyday life. So, providing stable and agile locomotion capabilities for robots to navigate a wide range of environments becomes pivotal. Developing locomotion in robots through biologically inspired methods, also facilitates furthering our understanding on how biological processes may function.

Connected modules in a configuration, exert force on each other as a result of interaction between each other and their environment. This phenomenon is studied and quantified, and then used as implicit communication between robot modules for producing locomotion coordination in MRs. Through this, a strong link between robot morphology and the gait that emerge in it is established.

A variety of locomotion controller, some periodic-function based and some morphology based, are developed for MR locomotion and bipedal gait generation. A hybrid Evolutionary Algorithm (EA) is implemented for evolving gaits, both in simulation as well as in the real-world on a physical modular robotic configuration. Limbless gaits in MRs are also learnt by learning optimal control policies, through Reinforcement Learning (RL).

Resumen

En robótica, la locomoción trata de proporcionar capacidades de locomoción autónoma a robots móviles. La mayoría de los robots actuales tiene alguna forma de locomoción para navegar en su entorno. Los modos de locomoción robótica se pueden repartir entre: (i) locomoción aérea, (ii) locomoción terrestre, y (iii) locomoción acuática (sobre o bajo el agua). Las tres formas básicas de locomoción terrestre son la locomoción mediante piernas, la locomoción sin miembros, y la locomoción basada en ruedas. Un Robot Modular, por otra parte, es un sistema robótico compuesto por varios módulos independientes, donde cada módulo es un robot en sí mismo. El objetivo de esta tesis es el desarrollo de la locomoción mediante piernas para un robot humanoide, así como el de la locomoción sin miembros para varias configuraciones de robots modulares. Inspirándose en la biología, también se investiga en esta tesis el desarrollo de la locomoción del robot según su morfología, gracias a técnicas de evolución y de aprendizaje.

La locomoción es una de las características distintivas de un organismo zoológico. Casi todas las especies animales, e incluso algunas especies de plantas, poseen algún tipo de locomoción. En los últimos años, los robots han "migrado" desde las fábricas y los laboratorios de investigación, y se están integrando cada vez más en nuestra vida diaria. Por estas razones, es crucial proporcionar capacidades de locomoción estables y ágiles a los robots para que puedan navegar por todo tipo de entornos. El uso de métodos de inspiración biológica para alcanzar esta meta también nos ayuda a entender mejor cómo pueden funcionar los procesos biológicos equivalentes.

En una configuración de módulos conectados, puesto que cada uno interacciona con su entorno, los módulos ejercen fuerza los unos sobre los otros. Este fenómeno se ha estudiado y cuantificado, y luego se ha usado como comunicación implícita entre los módulos para producir la coordinación en la locomoción de este robot. De esta manera, se establece un fuerte vínculo entre la morfología de un robot y el modo de andar que este desarrolla.

Se han desarrollado varios controladores de locomoción para robots modulares y robots bípedos, algunos basados en funciones periódicas, otros en la morfología del robot. Un algoritmo evolutivo híbrido se ha implementado para la evolución de locomociones, tanto en simulación como en el mundo real en una configuración física de robot modular. También se pueden generar locomociones sin miembros para robots modulares, determinando las políticas de control óptimo gracias a técnicas de aprendizaje por refuerzo.

Se presenta en primer lugar en esta tesis el estado del arte de la robótica modular, enfocándose en la locomoción de robots modulares, los controladores, la locomoción bípeda y la computación morfológica. A continuación se describen cinco configuraciones diferentes de robot modular que se utilizan en esta tesis, seguido de cuatro controladores de locomoción. Estos controladores son el controlador heterogéneo, el controlador basado en funciones periódicas, el controlador homogéneo y el controlador basado en la morfología del robot.

Se desarrolla como parte de este trabajo un controlador de locomoción lineal, periódico, basado en features, para la locomoción bípeda de robots humanoides. Los parámetros de control se ajustan primero a mano para reproducir un modelo cart-table, y el controlador se evalúa en un robot humanoide simulado. A continuación, gracias a un algoritmo evolutivo, la optimización de los parámetros de control permite desarrollar una locomoción sin modelo predeterminado.

Se desarrolla como parte de esta tesis un enfoque sobre algoritmos de Embodied Evolución, en otras palabras el uso de robots modulares físicos en la fase de evolución. La implementación material, la configuración experimental, y el Algoritmo Evolutivo implementado para Embodied Evolución, se explican detalladamente.

El trabajo también incluye una visión general de las técnicas de aprendizaje por refuerzo y de los Procesos de Decisión de Markov. A continuación se presenta un algoritmo popular de aprendizaje por refuerzo, llamado Q-Learning, y su adaptación para aprender locomociones de robots modulares. Se proporcionan una implementación del algoritmo de aprendizaje y la evaluación experimental de la locomoción generada.

Contents

List of Tables

List of Figures

List of Algorithms

CHAPTER 1

Introduction, problem definition and goals

Scouring for the definition of the word robot, one can come across several different definitions. Some of which are,

> "A machine controlled by a computer that is used to perform jobs automatically." - Cambridge Dictionary
>
> "A machine capable of carrying out a complex series of actions automatically, especially one programmable by a computer." - Oxford Dictionary
>
> "A real or imaginary machine that is controlled by a computer and is often made to look like a human or animal." - Merriam-Webster
>
> "A robot is a mechanical or virtual artificial agent, usually an electromechanical machine that is guided by a computer program or electronic circuitry, and thus a type of an embedded system." - Wikipedia

In general, a robot can be defined as an electromechanical system that can be programmed to: (i) either be controller remotely by a human operator, (ii) or to perform a set of predefined actions, (iii) or to take actions based on its sensory input, such that it physically effects its surrounding, explicitly. One could then argue that a washing machine or a modern day air-condition can fall into this category, and so, the task of defining a robot is non-trivial.

The field of Mobile Robotics is concerned with automatic machines that have locomotion capabilities. Any machine that can move around in its environment, either controlled remotely

or autonomously, can be categorized as a mobile robot. In contrast, industrial robots, which have multi-articulated arms connected to a fixed base, are stationary robots. **Robot Locomotion** is concerned with proving a robot the ability to move from one place to another, which is accomplished by manipulating its body with respect to the environment. Three basic modalities of locomotion in land based mobile robotics include: (i) wheel-based locomotion, (ii) legged locomotion and (iii) limbless locomotion. Wheeled robots are the most energy efficient for navigating on flat surfaces, but lack the ability to navigate in rough or uneven terrains (e.g.: stairs). Legged robots, on the other hand, have the potential to navigate in unstructured environment, they do not need a special infrastructure to operate in, and they are very well suited to interact with and navigate in human environment (e.g.: home, office, theme-park, mall, etc.). While limbless robots are robots that mimic creeping/crawling gaits of worms, insects, serpents, etc. Limbless robots can be useful in applications involving unforeseen and inaccessible environments, like under collapsed buildings, inside narrow pipes, on sand dunes, etc., where wheeled and legged robots fail to operate.

1.1 Problem definition

The main objective of this thesis is to develop locomotion controllers for limbless robots and a legged robot. Specifically, to develop controllers for creeping/crawling gaits in 2D robots, and bipedal walking gait in a humanoid robot.

Locomotion in general, whether gallop of a horse, flapping wings of a bird, or bipedal walking of a human, can be seen as repetitive and coordinated movement of limbs, through which the desired gait emerges. In limbless organisms like caterpillar and snake species, locomotion comes about as a result of coordinated bilaterally-symmetrical-muscle-contractions and through coordinated undulation of muscles, respectively. In robots, locomotion can be seen as coordinated actuation of joint actuators related to locomotion (e.g.: Limb joints of a humanoid robot).

The problem tackled in this thesis involves bringing about the needed coordination in joint actuations, which produces a stable gait in the respective robot. One way of achieving this task is to explicitly model joint trajectories, based on the kinematic structure of the target robot, which is the traditional approach towards robot locomotion. However, in this thesis, biologically inspired approaches towards robot locomotion are taken.

Two different robot platforms, one for limbless locomotion and the other for bipedal gait, used in this thesis are presented in the following subsection.

1.1.1 Robot

The target platforms used in this thesis are Modular Robot (MR)s and a simulated humanoid robot. Limbless gaits are evaluated on different modular robotic configurations, while bipedal

walking gait is evaluated on a simulated humanoid robot.

1.1.1.i Modular Robots

A MR is a system composed of several unit modules which, provided self-reconfigurable capability, can autonomously change its morphology. Each module, in a modular robotic configuration, is a robot which has its own actuator, computation unit, sensors, power unit, and connectors to physically connect to two or more modules. A single module, due to its simplicity, is limited in its capability. But several modules connected together to form a bigger robotic organism/configuration, is capable of performing more complex tasks. An advantage of a modular robotic system is its ability to change its morphology to suit the task at hand. Also, if one or a few modules in a robotic configuration fail, they can be easily replaced with another module, and the cost of the entire system can be bought down by mass producing unit modules.

In this thesis, *Y1* MR, which is the creation of Juan Gonzalez-Gomez [Gonzalez-Gomez, 2008] is used. As could be seen in Figure 1.1, a *Y1* module has an open-ended cube shape, and it is made up of two three-dimensional (3D) 'U' shaped objects connected together to form an hinge. The module has a dimension of , 1-Degree of Freedom (DOF), and a rotational range of . A *Futaba 3003* servomotor is used as the module's actuator. Locomotion controllers are evaluated both on simulated and real MRs. The simulator used is OpenRAVE [Diankov and Kuffner, 2008], which is an open source, Open Dynamics Engine (ODE) based physics robotic simulator.

The module's hinges are parallel to the ground surface when the actuator is at ( ). At a negative angle, the module's hinge rotates in anti-clock direction, and in clockwise direction, at a positive angle (Figure 1.2).

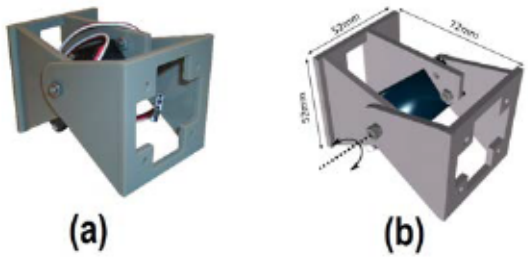


Figure 1.1: *Y1 module (a) Real and (b) Simulated versions.*

1.1.1.ii Humanoid Robot

A humanoid robot is a robot with its body shape built to resemble that of the human body. A humanoid design is mainly for the purpose of integrating the robot with human tools and environments, in which case, navigating human environments such as home, office, hospitals, malls, etc. becomes important.

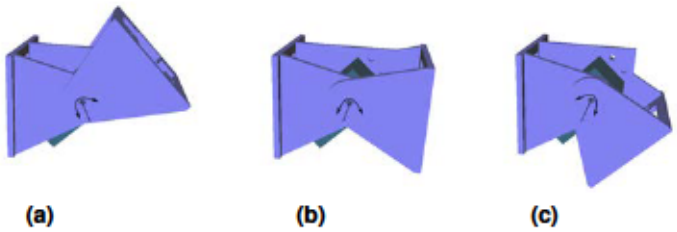


Figure 1.2: *Y1 module while actuated at (a) , (b) and (c) .*

In this thesis, a simulated version of the third edition of the, Humanoid for Open Architecture Platform (HOAP-3) robot (Figure 1.3) is used for bipedal walking gait. The HOAP-3 series of humanoids as a platform for research in bipedal locomotion, human robot collaboration and whole-body control, is developed by Fujitsu Automation, Japan.

Figure 1.3: *HOAP-3 robot. Source:* `http://robots.uc3m.es/index.php/HOAP3`

The HOAP-3 robot weighs , stands tall, and has 28 DOFs. Kinematic structure of the robot is as shown in Figure 1.4.

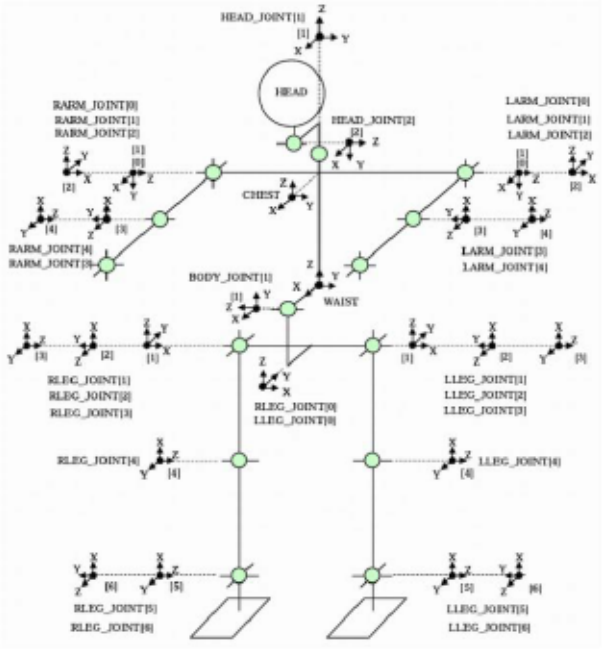


Figure 1.4: *Kinematics structure of HOAP-3. Source:* `http://robots.uc3m.es/index.php/HOAP3`

1.2 Motivation

Motivation for the research conducted during the development of this thesis comes from biology. Both, the locomotion itself, as well as the methods used for developing locomotion in robots are biologically-inspired.

1.2.1 Locomotion

In biology, locomotion is one of the main distinguishing characteristic for differentiating animals from plants. Animals use locomotion for a variety of reasons, including for finding food, a mate, a suitable habitat, or to escape from predators. For many animals, their ability to survive depends strongly on their locomotion capabilities, and so, natural selection has shaped their method and mechanism of locomotion, based on the animal and their environmental constraints. For example, migratory animals that travel vast distances annually, in search of food and breeding grounds, usually have gaits that cost very little energy per unit distance traveled. On the other hand, animal species that move quickly for the purpose of hunting down prey, or for escaping from predators, are more likely to have energetically expensive, but very fast gaits.

Typical robot applications involve jobs that are either dull, dirty or dangerous for humans. Robots used as companions and/or pets is another huge application domain under robotics.

Irrespective of the application, an important capability most robots need to possess to be functional, is locomotion. Many robots designed and built today, directly draw inspiration from humans, as well as other animal morphologies, with the purpose of assisting — if not replacing — humans in day to day life. Given the importance of locomotion in animals of all kinds, locomotion is a key feature that most robots need to possess. Furthermore, robot locomotion can also be used for studying and testing hypothesis concerning human and animal locomotion, from a biomechanics perspective.

1.2.2 Methodology

In biology, terms *Morphology*, *Evolution*, and *Learning* are defined as follows,

> ***Morphology***: Study of the form and structure of organisms and their specific structural features.
>
> ***Evolution***: Change in heritable characteristics of biological populations over successive generations.
>
> ***Learning***: The act of acquiring new, or modifying and reinforcing, existing knowledge, behaviors, skills, values, or preferences, and may involve synthesizing different types of information.

A newborn calf learns to stand up in about 30 minutes, and to walk and run in just a few hours after birth. A toddler learns to crawl, and eventually learns to walk in about 9 to 18 months time from birth. A good question to then ask would be: how much of this behavior (crawling/walking/running) is evolved (genetically-coded) versus learnt, and what influence does the morphology of the organism has on the produced gait? Some literature like [Alexander, 2003] attempt to answer this question.

Author of [Gatesy and Biewener, 1991] study bipedal gait in ground-dwelling bird species, and compare it with human bipedal gait, based on difference in morphologies between species. Relation between small morphological variance and performance in the form of gait speed, distance traveled and time of travel, in newborn garter snake species, is studied in [Arnold and Bennett, 1988].

Evolution of locomotion in anthropoid primates has been studied in [Ryan et al., 2012]. The authors compare fossils of 16 specimen to analyze evolutionary changes in locomotor adaptations in anthropoid primates over the last 35 million years. When land dwelling mammals re-entered the ocean, some 60 million years ago, they had to adapt their morphology and locomotion for the new ecology through evolution, resulting in the current day aquatic mammal species. A study on evolution of cost efficient swimming in marine mammals is presented in [Williams, 1999].

In [Adolph et al., 1997], authors study learning of crawling and walking behavior in infants. Tests of infants going up and down slopes were conducted longitudinally from the first week of

crawling, until several weeks after they begin to walk. This study suggests that there are no transfer of learnt knowledge over the transition from crawling to walking, but instead, infants learn all over again, how to cope with slopes from an upright position. The study also concludes that learning is online, rather than a simple association between a particular locomotor response to a particular slope.

1.3 Goals and Scope

It is evident from studies in biology, that locomotion of an organism is influenced/shaped: (i) by aspects of its morphology, (ii) through evolution, and (iii) by learning. Traditionally, the problem of robot locomotion is tackled in a top-down approach. That is, for a particular gait to emerge, first the kinematic structure of the robot is considered, based on which, joint trajectories are modeled through inverse-kinematics. Objectives of this thesis are as follows,

> Study how morphology of a robot can influence its locomotion, and develop locomotion controllers for two-dimensional (2D) modular robotic configurations that are morphology dependent.
>
> Use algorithms that are based on biological evolution for evolving gaits in both legged and limbless robots in simulation, as well as to evolve a gait on a physical modular robotic configuration in the real-world.
>
> Use learning algorithms inspired by behaviorist psychology, for learning gaits in modular robotic configurations, in an online fashion.

Each of the three approaches towards locomotion followed in this thesis are explained further in the following subsections.

1.3.1 Morphological Computation

The phenomena observed in biological systems that take advantage of their morphology to conduct computations needed for a successful interaction with their environments, is termed **Morphological Computation**. With respect to robots, Morphological Computation refers to outsourcing of computation needed for controlling the robot, to morphology and material property of the robot.

In Modular Robot (MR)s, which are physically connected multi-robot systems that have an embodiment, modules exert forces on one another when actuated. These forces come about as a result of physical interaction between modules, and between modules and their environment, as a function of the morphology of the robotic configuration. So, an objective of this thesis is to, on one hand investigate what effects morphology of a modular robotic configuration has on

its locomotion, and on the other hand to develop locomotion controllers that are morphology dependent and which can adapt to changes in robot morphology.

1.3.2 Evolutionary Robotics

Evolutionary Robotics is a field of research that employs a process analogous to natural evolution for generating robot controllers that adapt to their environment, all without needing direct programming by humans. The generation and optimization of robot controllers are based on evolutionary principles of blind variations, and survival of the fittest. Although typically applied for creating control system for robots, less frequent, Evolutionary Robotics (ER) methods are also applied for generating robot morphologies, as well as for co-evolving control systems and robot morphology simultaneously [Lipson and Pollack, 2000], [Bongard and Pfeifer, 2003].

A common practice in ER is to evolve the controller in simulation and then transfer the evolved controller on to a physical robot. There exists a reality gap in this method because not every single aspect of the physical world and the robot can be modeled accurately in the simulation environment. So, the evolved behavior in simulation may not transfer well onto a real robot. **Embodied Evolution** is a ER method, where evolution is performed on the physical robot, so as to close the reality gap that exist between simulation environment and the real-world.

Objectives of this thesis, from an ER perspective are to: (i) optimize locomotion controllers through evolution, for both limbless gaits in modular robotic configurations, as well as bipedal gait in the humanoid robot, (ii) evolve a gait on a physical modular robotic configuration, through Embodied Evolution (EE).

1.3.3 Reinforcement Learning

Reinforcement Learning is an area under Machine Learning (ML) inspired by behaviorist psychology in humans and animals. Reinforcement Learning (RL) deals with the problem of getting an agent to act in the world, so as to maximize its rewards. For example, consider teaching a dog a new trick, where you cannot tell it what to do or not to do, but you can only reward or punish it, if it does the right or wrong thing respectively. The agent (dog in this case) has to figure out what it did that made it get the reward or the punishment, which is known as the credit assignment problem.

An objective in this thesis, from the learning perspective, is to use RL method to train Modular Robot (MR)s to locomote. To do so only by rewarding them (in a computational sense) whenever they take a step forward in the right direction, or by punishing them whenever they either take a step backward, flip over or stay standstill.

1.4 Structure of the PhD

The following PhD thesis is structured as follows. First, in chapter 2, page 11, current state-of-the-art in the fields of MR, particularly related to MR locomotion and controllers, bipedal locomotion and Morphological Computation are reviewed.

In chapter 3, page 45, five different modular robotic configurations used in this thesis are explained first, followed by four different locomotion controllers for MR locomotion developed in this thesis are presented. The first two controllers are heterogeneous and periodic-function based locomotion controllers, while the next two are homogeneous and morphology based locomotion controllers.

In chapter 4, page 87, a feature based linear periodic locomotion controller for bipedal gait in humanoid robots is presented. Control parameters are first hand-tuned to replicate a cart-table model based gait, and the same is evaluated on the simulated humanoid robot. Then, a walking gait is evolved by optimizing control parameters through Evolutionary Algorithm (EA).

In chapter 5, page 105, an Embodied Evolution (EE) approach for evolving gaits in physical modular robotic configurations is presented. The hardware implementation, experimental setup for EE, and the EA implemented, are all explained in detail in this chapter.

In chapter 6, page 133, first an overview of RL and Markov Decision Process (MDP) are provided. Then, a popular RL algorithm called Q-Learning, and an adaptation of it for the purpose of learning locomotion in MRs are presented. Implementation of the learning algorithm and evaluation results of the learnt gait are provided to close this chapter.

Finally, the thesis is concluded in chapter 7, page 159.

CHAPTER 2

State-of-the-art

Introduction

As presented in the Introduction chapter, the aim of the work presented in this PhD thesis is locomotion in MR, and bipedal gait in a humanoid robot. In this chapter, first an introduction to the field of MR is provided, followed by review of a wide range of MR platforms in the literature. Then, some of the most common gaits developed in MR are reviewed, followed by a review on a wide range of locomotion controller in MR. Some work related to bipedal gait in huamnoid robots, and Morphological Computation in the field of robotics are also reviewed in this chapter.

2.1 Modular robotics

A **Self-Reconfigurable Modular Robot** can be defined as an aggregation of the following definitions,

Module: A piece or a set of pieces that are repeated in a construction of any kind, to make it easier, regular and economical.

Modular Robot: A robotic system formed out of independent unit modules.

Reconfigurable/Multi-Configurable Modular Robot: A MR system wherein modules can be (manually) connected to each other in multiple ways to form robots of varying morphologies.

Self-Reconfigurable Modular Robot: A reconfigurable MR that has the ability to independently change its morphology through self-reconfiguration.

So a Self-Reconfigurable Modular Robot (SRMR) is a system that is composed of several independent unit modules, which are robots themselves, and has the ability to change its morphology through self-reconfiguration. Each module in such a system would have its own computation unit, actuator(s), sensor(s), power supply and two or more connectors to physically connect to other modules. A single module, due to its relative simplicity, is limited in its capability. But several modules connected together to form a bigger robotic organism (or a robotic configuration), is capable of performing more complex tasks. An advantage of a SRMR system is its ability to change its morphology to suit the task at hand. Also, if one or a few modules in the configuration fail, then they can be easily replaced with another module, and the cost of the entire system can be bought down by mass producing unit modules.

MRs based on their configurable capabilities, can be classified into: reconfigurable or multi-configurable, self-reconfigurable, self-assembling, metamorphic and self-replicant systems. Reconfigurability refers to the property of a system to be configured in different ways, without considering the means of reconfiguration. Self-reconfigurable robots are those that are able to change their configuration on their own, while in manually configurable robots the reconfiguration has to be done externally (i.e. by an operator). Metamorphic robots are those that are composed of one kind of repeated module that is able to change its shape. Most reconfigurable robots are also metamorphic. Self-assembling robots have the ability to independently come together to form into a bigger entity. In most cases, self-assembling robots are also self-reconfigurable, with the ability to change their morphology. Self-replicating robots are those that have the ability to make copies of themselves — provided they have the necessary modules — by their own means.

Based on the architecture of the configuration, a MR can be broadly classified into lattice-type, chain-type and hybrid-type architectures. Lattice-type architectures have units that are arranged and connected in some regular space-filling two-dimensional (2D) or three-dimensional (3D) patterns, such as a 2D grid, or cubic or hexagonal grid. Control and motion in lattice-type are executed in parallel, and they usually offer simpler computational representation that can be more easily scaled to complex systems. In lattice-type systems locomotion is achieved through continuous self-reconfiguration, where each module has the ability to move independently in the configuration. Locomotion in lattice-type systems gives the notion of modules flowing on the ground, which is visually similar to locomotion of an amoeba, or to that of a puddle of water flowing on a flat surface.

Chain-type (or tree-type) architectures have units that are connected together in a string or tree topology. This chain or tree can fold up to become space filling, but the underlying architecture is serial. Chain-type architectures can reach any point in 3D space, and are therefore more versatile, but are more computationally difficult to represent and analyze. In a chain-type system locomotion is achieved by controlling the Degree of Freedom (DOF) actuator(s) of individual modules in a fixed configuration.

Hybrid-type architectures have features from both lattice-type and chain-type architectures. Many of the more recent MRs can be classified as hybrid-type systems because they can be configured both as chain-type structures and lattice-type structures.

A modular robotic configuration can either be composed of homogeneous modules, where all the unit modules are identical, or with heterogeneous modules. In an heterogeneous MR system, a robotic configuration could be composed of two or more kinds of modules, where each type of module is designed for a specific task.

2.1.1 The origins

The origin of modular multi-configurable robots is known to have started in 1990 with Cellular Robotic System (CEBOT) [Fukuda and Kawauchi, 1990], from Nagoya University, a dynamically configurable robot (Figure 2.1) that has the capability of self-organizing, self-evolution and functional amplification[1]. It is composed by many robotic units, called *cells*, with a simple function. CEBOT can reconfigure the whole system depending on the given task and environment, and organize collectively, as a system demonstrating Swarm Intelligence. The concept of CEBOT is based on biological multi-cellular organisms. This research project included mutual communication between cells, optimum dynamic knowledge allocation among cells, reconfiguration strategy of the system and Artificial Life concepts such as modeling cooperative behavior in ant swarms. This project addressed many interesting research problems such as dynamic decentralized planning, dynamic distributed and coordinated control systems, as well as hardware systems. Experiments in automated reconfiguration were carried out, but the robot didn't have the ability to self-reconfigure, as a manipulator arm was required to do so.

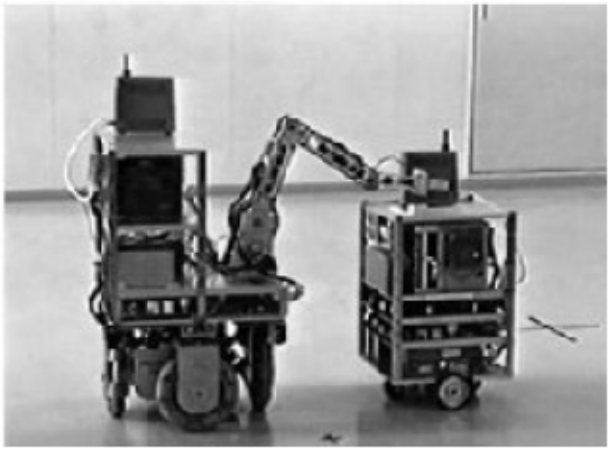


Figure 2.1: *Cellular Robotic System (CEBOT). Source: [Fukuda and Kawauchi, 1990]*

The history of Modular Robot (MR) can be traced back to 1972, when the Active Cord Mechanism (ACM) [Hirose, 1993] was created. It was the first robot using the principles of a

[1] Ability of a system to coordinate together to accomplish tasks that cannot be achieved by individual units

serpentine locomotion, mimicking the gait of an actual snake. It was created by S. Hirose at the Tokyo Institute of Technology, and it could move at a speed of approximately 40 cm/sec. The entire length of the robot was 2m, and it had 20 joints. Each joint consists of servo-mechanisms that can bend to the left and right. To make contact with the ground, casters were installed along the length of the body, and characteristics were added that make it easy to slide in the direction of the torso and difficult to slide in the orthogonal direction of the body length.

Locomotion in this robot was achieved by sending sinusoidal control signals to the head joint servo, followed by a phase-shifted sinusoidal signal to the following joint, and so on to all the joints from the head to the tail of the robot. This resulted, in the body as a whole, propelling forward by sending a wave from the head to the tail of the robot, and the torso of the robot sliding over the floor with the help of the the passive caster-wheels. This principal of locomotion corresponds to that of the swimming motion of an eel. Although in its origin version, ACM cannot be classified as a modular multi-configurable robot, it was a milestone during its development. In the last version (ACM-R5) each joint unit has a CPU, battery and motors, so that they can operate independently. Through communication lines each unit exchanges signals and automatically recognizes the total number of units from the head to tail. In the latest version, operators can remove, add, and exchange units freely and they can operate ACM-R5 flexibly according to situations. It can operate both on the ground and in water undulating its long body. The evolution of ACM is as seen in Figure 2.2.

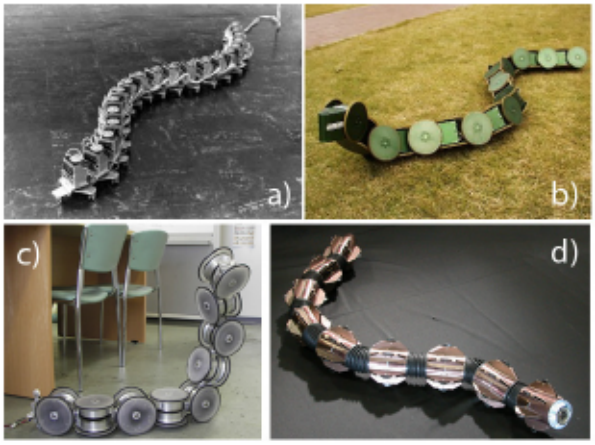


Figure 2.2: *Active Cord Mechanism: (a) Version III, (b) R3, (c) R4 and (d) R5. Source:* `http://www.expo21xx.com/automation21xx/18224_st3_university/default.htm`

2.1.2 Chain-type

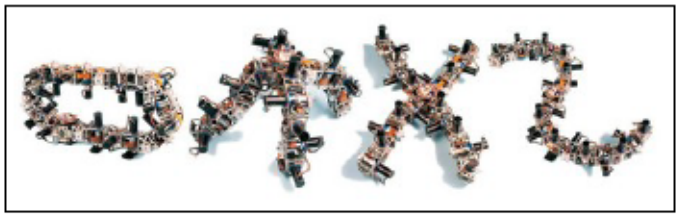
Chain-type architecture is the most common architecture type found among systems in the early years of Modular Robot (MR). Locomotion in a chain-type systems are achieved by controlling actuator(s) of individual modules in a fixed configuration. Some of the most common MRs in

the literature, from the early years to the most recent design, are reviewed in the following subsections.

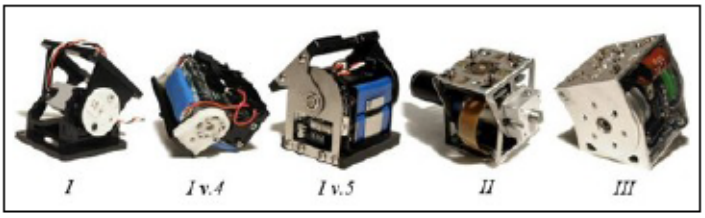
2.1.2.i PolyBot

PolyBot [Yim et al., 2000] [Yim et al., 2001] [Yim et al., 2007] is the evolution of PolyPod from year 1997 (2.3a). It is made up of many repeating modules. Each module is virtually a robot on its own, having a computer, a motor, sensors and the ability to attach to other modules. In some cases, power is supplied off-board and passed from module to module. These modules attach together to form chains, which can be used as an arm, leg or a finger, depending on the task at hand.

PolyBot has gone through many variations with three basic generations. The evolution of the main module is as seen in Figure 2.3b. PolyBot features several different configurations (snake, wheel, quadruped, etc), a multi-master/multi-slave structure in a multi-threaded environment, with three layers of communication protocol (MDCN) and a Attribute/Service Model.



(a) *Configurations*



(b) *Module versions*

Figure 2.3: *PolyBot. Source:* `http://spectrum.ieee.org/robotics/industrial-robots/modular-robots`

2.1.2.ii CONRO

The CONRO Self-Reconfigurable Modular Robot (SRMR)[2] [Castano et al., 2000] [Castano et al., 2002] [Shen et al., 2002] was developed by P. Will et al., at the University of Southern California. It is made of a set of connectable modules (Figure 2.4). Each module is an autonomous unit

[2]

that contains batteries, one STAMP II micro-controller, two motors, four pairs of IR transmitters/receivers and four docking connectors to allow connections with other modules. Of the four connectors, each module has one male connector, used to connect to other modules, and three female connectors, to which other modules can connect to.

A distributed control mechanism inspired by the concept of hormones in biological systems is developed for controlling CONRO modular robotic organisms. Hormones are special messages that can trigger different actions in different modules. They can be used to coordinate locomotion and reconfiguration in the context of limited communication and dynamic network topologies. An automatic docking system through SMA-triggered locking/releasing mechanisms is developed as part of the CONRO project.

Figure 2.4: *Different configurations of CONRO. Source:* `http://www.isi.edu/robots/conro/`

2.1.2.iii Molecubes

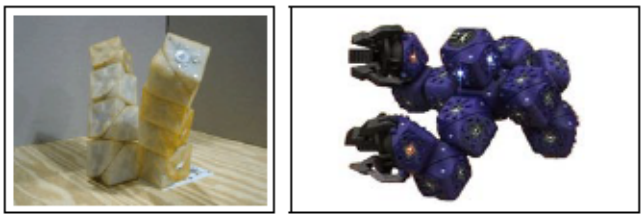
Molecubes[3] [Zykov et al., 2005], developed by V. Zykov and H. Lipson at Cornell University, is a metamorphic robot made up of a series of modular cubes, each containing identical machinery and the complete computer program for replication (Figure 2.5). The cubes have electromagnets on their faces that allow them to selectively attach to eachother, as well as detach from one another, and a complete robot consists of several cubes linked together. Each cube is divided into two halves along the diagonal, which allows a robot composed of many cubes to bend, reconfigure and manipulate other cubes. For example, a tower of cubes can bend itself over at a right angle to pick up another cube (2.5a).

2.1.2.iv Symbricator

Symbricator[4] [Kernbach et al., 2011a] [Kernbach et al., 2011b] [Matthias et al., 2012] [Russo et al., 2012] [Schlachter et al., 2012] is a heterogeneous, SRMR platform developed mainly at the Institute of Parallel and Distributed Systems, University of Stuttgart, and at the Institute for Process Control and Robotics, Karlsruhe Institute of Technology, as part of European funded projects *Symbrion* and *Replicator*. As part of this five year project, three distinct MR platforms,

3

4



(a) *First version* **(b)** *Second version*

(c) *Third version*

Figure 2.5: *Molecubes. Source: [Zykov et al., 2005]*

namely *Backbone*, *Active Wheel* and *Scout*, were developed (Figure 2.6), with each platform designed for a special purpose in the end mission.

The research objectives of the *Symbricator* (*Symbrion* + *Replicator*) project was to investigate reconfigurability, adaptability and learn-ability in modular robotic organisms on one hand, and to investigate evolvability of symbiotic[5] systems, and analogies to biological systems with respect to long-term and short-term evolution, on the other hand.

The *Backbone* platform has an open ended cube shape, and has 1-Degree of Freedom (DOF) for three-dimensional (3D) locomotion. It has a very strong brushless drive capable of lifting several connected modules, and is specialized for 3D locomotion. Each individual module can also perform locomotion in two-dimension (2D), in all four directions, using a pair of specialized screw-driver wheels. Connected modules can also move in two-dimensional (2D) using these wheels. In this platform, each module has four symmetric, genderless docking elements for connecting to other modules, and some basic sensors for perceiving its environment.

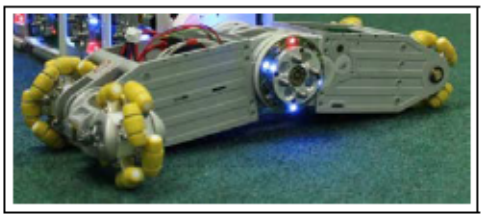
The *Active Wheel* platform is designed with the main functionality of transportation task in mind. It is able to carry and transport a modular robotic organism made up of several *Backbone* and *Scout* modules, in an energy efficient manner. This platform has two symmetric arms connected at the hinge, which can rotate in the range of . Two omni-directional wheels are attached at the end of each arm, which provides 2D locomotion capabilities for this platform. There are two docking elements on the hinge, which can be used to connect other modules. Each docking element can rotate independently, and has two separate motors for this.

The *Scout* platform is similar in shape to the *Backbone* platform, but it is designed for the

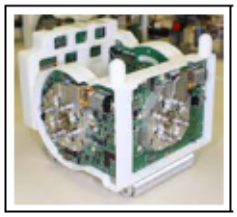
[5] **Symbiosis**: Close and often long-term interaction between two or more different biological species

purpose of fast locomotion for exploration. This platform has tracks for fast 2D locomotion on rough terrain. It has extra sensors compared to the other two platforms, for sensing and exploration task. It has two laser-camera systems on its front and rear, for far and short-range obstacle detection. It also has a motor for 3D locomotion, but the motor is not as powerful as the one present in the *Backbone* platform, as the main purpose of this platform is not macro-locomotion, but exploration. Four docking units are available to connect to other modules, and the docking unit on the left side has a separate motor for rotating between . These modules are best suited to be placed on the outer edge of a modular robotic organism, as it would then not have too many modules to lift for 3D locomotion, and can use the extra sensors for sensing the environment.

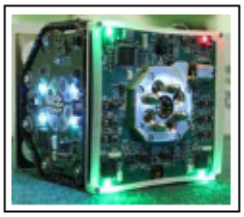
All three heterogeneous modular robotic platforms have some homogeneous elements, one of them being the common docking mechanism, required for connecting any combination of different types of modules to form a larger multi-robotic organism. A docking design called Cone Bolt Locking Device (CoBoLD) is employed for the docking units, which is genderless, symmetric and can handle misalignment. The docking unit also provides wired communication and energy sharing between connected modules. As one of the main research goals of this project is energy sharing mechanism between connected modules, a unified energy sharing system is commonly implemented in all three platforms.



(a) *Active Wheel*



(b) *Backbone*



(c) *Scout*

Figure 2.6: *Symbricator modules. Source: [Kernbach et al., 2011b]*

2.1.2.v Microtub

Microtub [Brunete et al., 2012b] is an heterogeneous modular multi/configurable microrobot developed at the Technical University of Madrid (UPM). It is composed of different types of active modules (ones that are able to move) and passive modules (ones that have to be acted upon).

Different types of modules that have been developed under this project is as shown in Figure 2.7. Diameter of each module is only 27 mm, and thickness of some parts is less than 1 mm.

The camera/contact module is used for environment information acquisition, needed for detecting holes, breakages and cracks in the pipes. It is also used as a contact sensor to detect if the microrobot is facing an obstacle. The rotation module has 2-DOF, that allows the module to rotate in vertical and horizontal planes. A set of these modules put together can perform an undulating (snake-like) locomotion. The support and extension modules are used to perform inchworm like locomotion (move forward and turn right and left). Finally, the helicoidal module was designed to be a fast-drive module that is able to push other modules forward or in reverse directions.

To assemble the modules together, a common interface has been built. This interface allows for mechanical and electrical connections between the modules. Each module includes an electronic control board that performs the following tasks: control of actuators and sensors, communications, auto-protection and adaptable motion, self-orientation detection, and low-level embedded control.

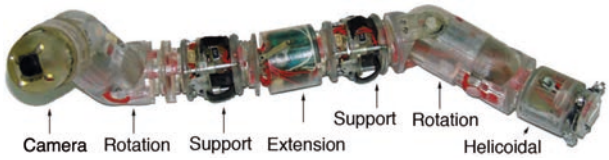


Figure 2.7: *Microtub modules. Source: [Brunete et al., 2012b]*

2.1.3 Lattice-type

In lattice-type architecture, modules connect their docking interfaces at points to form virtual cells of some regular grid-structure. This network of docking points can be compared to atoms in a crystal and the grid to the lattice of that crystal. Usually few units are sufficient to accomplish a reconfiguration step. A much simpled mechanical design and computational representation is allowed in lattice-type architecture. Reconfiguration planning, in lattice-type architecture, can be scaled to complex systems, much easily.

2.1.3.i Cross-Cube and Cross-Ball

Cross-Cube [Meng et al., 2010] and Cross-Ball [Meng et al., 2011] were developed at the Stevens Institute of Technology, mainly for studying self-reconfiguration in Modular Robot (MR). These platforms were used to study how Self-Reconfigurable Modular Robot (SRMR) can adapt in dynamic environments by changing their morphologies.

Cross-Cube is a SRMR developed in a robot simulator using a real-time physics engine PhysX (Figure 2.8). Each module in a Cross-Cube robot is a cubic structure having its own

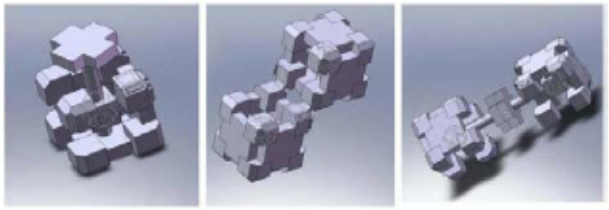
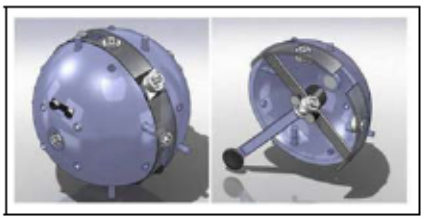
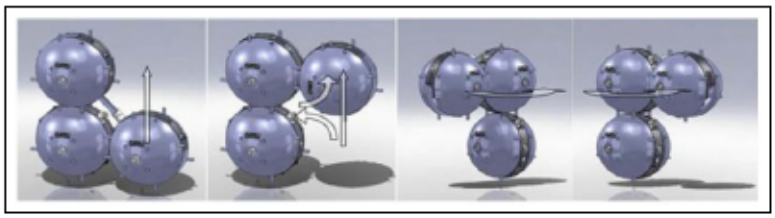


Figure 2.8: *Cross-Cube Module. Source: [Meng et al., 2010]*



(a) *Module*



(b) *Parallell and Diagonal movements*

Figure 2.9: *Cross-Ball Module. Source: [Meng et al., 2011]*

computing and communication resource and actuation capability. Each module can perceive its local environment using on-board sensors, and communicate with neighboring modules. Each Cross-Cube module consists of a core and a shell. The core is a cube with six universal joints. Each joint can be attached to or detached from the joints of its neighboring modules. The axis of each joint can be actively rotated, extended, and returned to its default direction. Basic module movements include rotating, climbing, and parallel movements.

Cross-Ball is a new lattice-based, three-inch diameter sphere SRMR design, that is currently under development at the Stevens Institute of Technology (Figure 2.9). Its major features include: several flexible reconfiguration capabilities such as rotating, parallel and diagonal movements, so that various three-dimensional (3D) configurations can be constructed; a flexible and robust hardware platform for SRMRs using complex self-reconfiguration algorithms, such as the morphogenetic control algorithm developed for MRs introduced in [Meng et al., 2010]; and the mobility of each individual module to simplify the configuration process under certain scenarios and potential applications to swarm robotics. It consists of three main components: an arm system in the middle and two spherical halves on the sides. The two spherical halves can rotate according to the arm system. Three pairs (six) of genderless attachment mechanisms are equipped in three orthogonal directions, with each attachment of a pair placed opposite

to each other. Using the rotary arm and side clasp, a module is capable of three types of self-reconfiguration motions: rotating, parallel and diagonal movements.

2.1.4 Hybrid-type

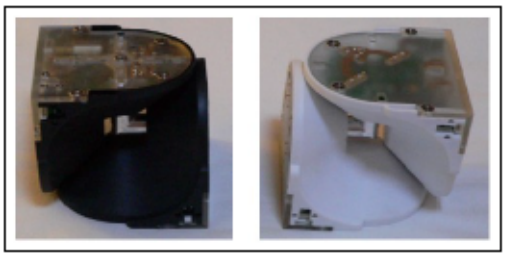
Hybrid-type architecture takes advantages of both chain-type and lattice-type architectures. Control and mechanism are designed for lattice-type reconfiguration, in addition to allowing a module to reach any point in the configuration space. Some of the most recent designs are of Hybrid-type architecture, which are reviewed in the following subsections.

2.1.4.i UBot

UBot [Zhao et al., 2011] [Liu et al., 2012] (Figure 2.10) is a novel Self-Reconfigurable Modular Robot (SRMR) system made of autonomous robotic modules at the State Key Laboratory of Robotics and System of Harbin Institute of Technology. Each robotic module is made up of a simple structure and 2-Degree of Freedom (DOF). A group of connected modules are able to change their configuration by changing their local connections, and has functionality of a robotic system which is capable of generating complicated motions for accomplishing a large variety of tasks, such as: transportation, exploration, inspection, construction and in-situ resource utilization.

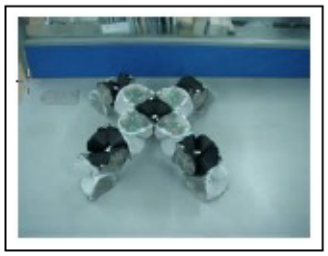
Multimodal locomotion and self-reconfiguration are the basic and essential capabilities for a SRMR system, and UBot achieves these by combining the advantages from chain-type and lattice-type systems. Each UBot module is a cubic structure, based on a universal joint with two rotational DOF and four connecting surfaces that can connect to or disconnect from adjacent modules. The smart structure and the reliable connecting mechanism of the modules make the robot flexible enough to perform both multimodal locomotion and self-reconfiguration. UBot can perform locomotion in the modes of quadruped, chain and loop configurations. Besides, the system can self-reconfigure from one configuration to another (e.g.: Quadruped configuration to chain configuration).

UBot presents a new sensor module for SRMR system, and a novel docking method for precise docking between modules. The sensor module is designed with the same external dimensions as active and passive modules. In order to maintain functional consistency, the connecting surfaces are equipped with hook holes and electrical contacts to connect and communicate with active modules. The sensor module is installed inside with a wireless CCD vision sensor, four linear Hall sensors, infrared distance sensors and acceleration sensor. The automated docking progress has three steps: visual prepositioning process, precise positioning through hall sensors and crawling and self-locking by the hook-type connecting systems.



(a) *Active and passive modules*

(b) *Caterpillar gait*



(c) *Quadruped gait*

Figure 2.10: *UBot. Source: [Zhao et al., 2011]*

2.1.4.ii SMORES

Self-assembling MOdular Robot for Extreme Shape-shifting (SMORES)[6] [Davey et al., 2012] is a SRMR developed at the University of Pennsylvania with the goal of realizing a universal Modular Robot (MR). It uses five identical motors to achieve a module with 4-DOF (Figure 2.11), three active docking connectors, and module movement that can utilize lattice, chain and mobile module movement strategies. SMORES modules can drive upright and upside down, and move by themselves to connect with other modules. SMORES is designed to emulate connection point arrangements and DOF of many existing systems like Polybot, CONRO and Superbot, amongst others.

SMORES has one passive and three active docking ports. Active docking ports control the attachment process, and passive ports only provide a physical space for a neighboring module to connect to. All connectors on the module are genderless. Power is on-board and communication between modules is achieved wirelessly, so docking connections are needed only for mechanical attachment. On-board processing is done by an *mbed* micro-controller that is based on the NXP LPC1768, with a 32-bit ARM Cortex-M3 core running at 96MHz. Wireless data transmission between modules and a central controller is achieved with X-Bee radio transmitter/receivers.

[6]

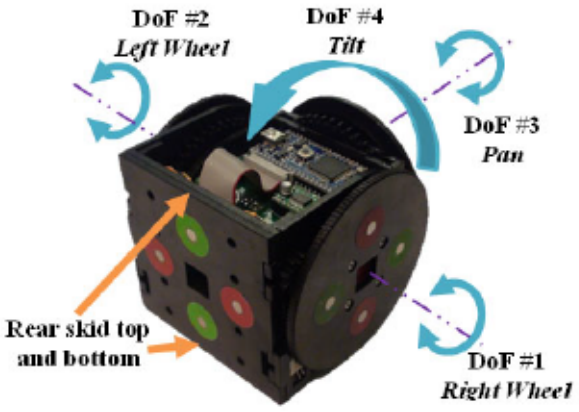


Figure 2.11: *SMORES Module. Source: [Davey et al., 2012]*

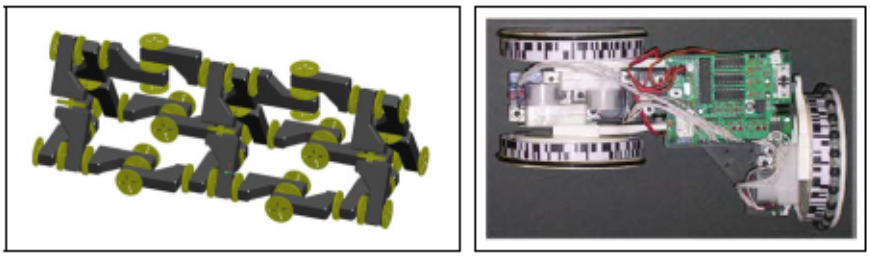
2.1.4.iii

(Modular Mobile Multirobot) [Wolfe et al., 2012] is a design for a low-cost MR based on a previous design [Kutzer et al., 2010] developed at the Johns Hopkins University. The modules are self-mobile, with three independently driven wheels that also serve as connectors (Figure 2.12). The new connectors can be automatically operated, and are based on stationary magnets coupled with mechanically actuated ferromagnetic yoke pieces. Extensive use of plastic castings, laser-cut plastic sheets, and low-cost motors and electronic components are made in developing this platform. Modules interface with a host PC via Bluetooth radio. An off-board camera, along with a set of modules and a control PC form a convenient, low-cost system for rapidly developing and testing control algorithms for self-reconfiguration.

To reduce cost, on board sensing is minimized through the use of external imaging. Multiple robots can operate in an arena observed by an overhead camera. A real-time program running in MATLAB interfaces to the camera over Universal Serial Bus (USB) port and to each robot in the arena via Bluetooth. Special absolute encoder markings on the perimeter of wheels are used to provide high-accuracy measurement of wheel angles. The processor used is an Atmel ATmega328 running Arduino firmware. Serial commands are parsed, and then motion commands are sent to three PIC18F1320 microprocessors with a Step/Direction interface. Each PIC performs closed-loop position and/or speed control on the motors, counts encoder pulses, and outputs a Pulse Width Modulation (PWM) signal for the motor driver. The estimated cost of the system is only 190$.

2.1.4.iv Roombots

Roombots (RB) [Sproewitz et al., 2009] [Spröwitz, 2010] were developed at the Swiss Federal Institute of Technology in Lausanne (EPFL) to form the basis for self-reconfigurable furniture (Figure 2.13). RB were designed to facilitate a single module to autonomously travel, through



(a) *Example of complex lattice and chain structures*

(b) *Detail of the module*

Figure 2.12: [3] . *Source: [Wolfe et al., 2012]*

self-reconfiguration, to any position on a two-dimensional (2D) grid. Here a module can travel on a 2D grid through a sequence of attachments and detachments between the module and the connection mechanism of the grid structure. The design also ensure to overcome concave edges in 3D, by using only 3-DOF per module.

A single RB module is composed of two cube-like elements, each with an edge length of 110 mm. Each module has three continuous rotational DOF, and up to ten four-way symmetric, hermaphrodite Active Connection Mechanisms (ACM) connectors, allowing RB modules to autonomously connect to and disconnect from other RB modules and the grid structures. A RB module weighs about 1.4 kg. Any joint of a RB module can deliver sufficient torque to lift an additional RB module.

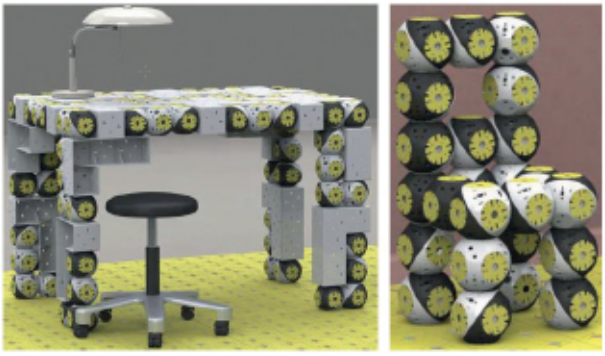


Figure 2.13: *Roombots. Source: [Bonardi et al., 2012]*

2.1.5 Locomotion

Locomotion in Modular Robot (MR)s is achieved through coordinated action of individual modules. In chain-type MRs, locomotion is achieved through continuous actuation of it's Degree of Freedom

(DOF) motors, whereas in lattice-type, locomotion comes about as a result of continuous self-reconfiguration. Gait produced by a modular robotic configuration depends on the morphology of the configuration. Gaits ranging from creeping and crawling gaits, achievable in limbless modular robotic configurations, through walking and rolling gaits achievable in more complicated two-dimensional (2D) and three-dimensional (3D) configurations are reviewed in the following subsections.

2.1.5.i Limbless locomotion

Modular robotic configurations that lack limbs, such as linear configurations, use their body to generate the propulsive force needed for locomotion. Three different limbless gaits that utilize module's DOF motors in linear configurations have been surveyed in the following subsections.

Caterpillar gait

Caterpillar gait is the most common gait in all of chain-type MRs [Kurokawa et al., 2003], [Støy et al., 2003], [Brunete et al., 2012b], [Gonzalez-Gomez and Boemo, 2005], [Gonzalez-Gomez, 2008]. It has been implemented in virtually all the modular robotic platforms discussed in subsection 2.1.2, page 14. It is a one-dimensional (1D) gait inspired by the crawling locomotion of a biological caterpillar. The robot can move either in forward or backward directions on a straight line, and it is commonly implemented in linear modular robotic configurations. The gait is achieved by oscillating only the pitch-axis actuator(s) of connected modules, with consistent phase-shift in oscillation between consecutive modules (Figure 2.14). This produces a traveling wave along the length of the configuration, which results in propelling the robot forward in the direction of the traveling wave. Negating the phase-shift values of the oscillating modules, results in the linear robot moving in the opposite direction.

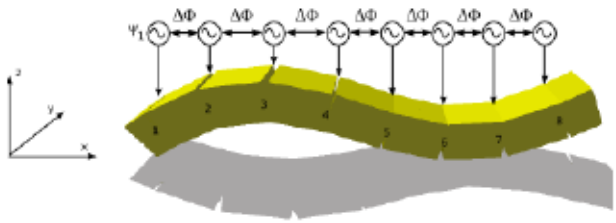


Figure 2.14: *Phase-relation between consecutive modules in a linear configuration to produce Caterpillar gait. Source: [Gonzalez-Gomez, 2008]*

Lateral-shift

This is a 2D gait, and can be implemented in linear configurations with alternating — pitch-axis (vertical oscillation) and yaw-axis (horizontal oscillation) — actuators, along the length of the

configurations. Lateral-shift gait is inspired by and replicates locomotion of *sidewinder* snake species. This gait is achieved by oscillating both pitch-axis actuators and yaw-axis actuators of the configuration, but here phase-shift in oscillating modules are separate for the two sets of pitch-axis and yaw-axis actuators. There exists consistent phase-shift between consecutive pitch-axis actuators, as well as between consecutive yaw-axis actuators (Figure 2.15). Amplitude, offset and phase parameter of the oscillation determines the shape, direction and speed of the gait. Lateral-shift gait is demonstrated in [Støy et al., 2003], [Brunete et al., 2012b], [Gonzalez-Gomez and Boemo, 2005] and [Gonzalez-Gomez, 2008].

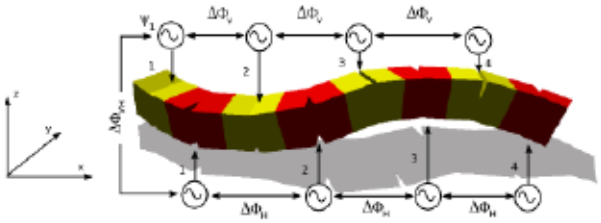
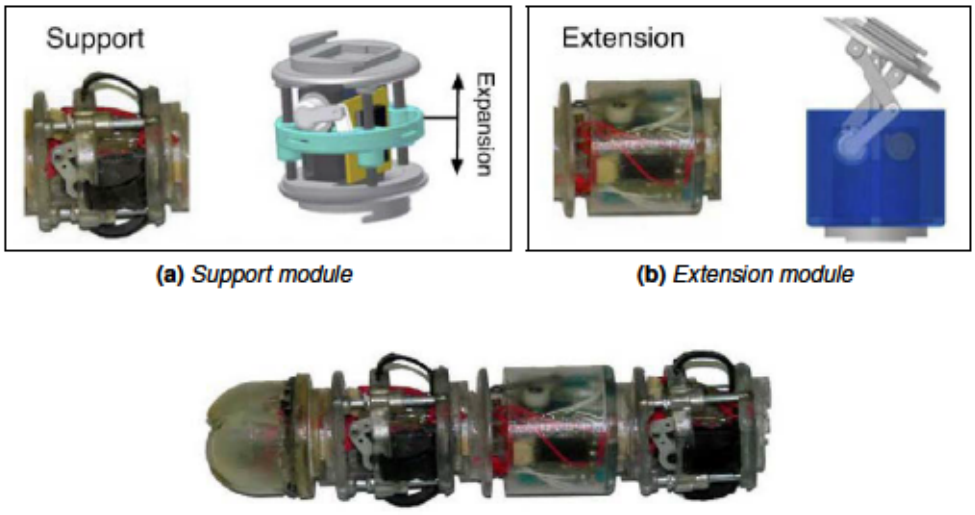


Figure 2.15: *Phase-relation between pitch-axis (vertical oscillation) and yaw-axis (horizontal oscillation) actuators in a linear configuration, for producing lateral-shift gait. Source: [Gonzalez-Gomez, 2008]*

Inchworm



(a) *Support module* **(b)** *Extension module*

(c) *Inchworm configuration*

Figure 2.16: *Microtub inchworm gait modules and configuration. Source: [Brunete et al., 2012b]*

Inchworm gait is achieved in linear configurations by contracting and elongating the length of the configuration. In [Brunete et al., 2012b], inchworm gait inside pipes of varying diameters

has been implemented in the heterogeneous multi-configurable MR *Microtub*, using two kinds of modules: support module (Figure 2.16a), and extension module (Figure 2.16b). Support module, the outer surface of which is partially rubber, when actuated expands outward, firmly holding the module to the inner-surface of the pipe. It is a passive module, as in, it does not by itself propel the robot, but aids in doing so. The extension module, which has a linear actuator, when actuated can both extend and rotate in the yaw-axis. The gait is achieved by connecting at least one extension module in between two support modules (Figure 2.16c). Sequence of action in such a configuration to achieve inchworm gait is as follows,

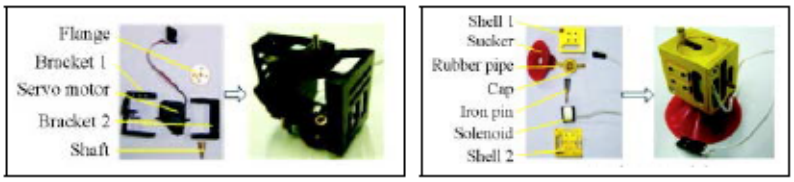
1. Rear support module is actuated, expanding and holding the surface of the pipe, while the front support module is released.
2. The center extension module is extended, pushing the robot forward.
3. Front support module is actuated, while the rear support module is released.
4. The center extension module is contracted, pulling the robot forward.

In [Wang et al., 2009] the authors have studied kinematics of the gait of a biological inchworm, and replicated the same on a linear modular robotic configuration with suction cups. Two kinds of modules are used: joint modules (Figure 2.17a) with 1-DOF actuator in pitch-axis, and attachment modules (Figure 2.17b) with an active suction cup for attaching the module to vertical surface. A linear configuration is formed by connecting three joint modules serially, along with two attachment modules at either end (Figure 2.17c). Here, locomotion is achieved by first activating the upper attachment module, holding the configuration on the vertical surface, while releasing the lower attachment module, and at the same time contracting the length of the configuration by actuating all three joint modules. Then by activating the lower attachment module, while releasing the upper attachment module, and actuating the joint modules to extend the length of the configuration, pushing the robot upward.

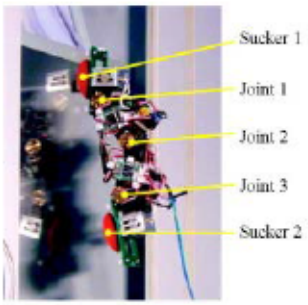
In [Russo et al., 2012] inchworm gait has also been achieved in a linear configuration containing three *Scout* modules (Figure 2.18). The gait is achieved by repeating cycles of contraction and elongation of the body length of a linear configuration, through coordinated actuation of module's DOF actuators. Here, unlike in the previous two cases, the gait has been demonstrated on horizontal surface, in open-air.

2.1.5.ii Walking locomotion

In biological terms, walking is a gait in which at least one foot is kept in contact with the ground at all times during locomotion. Configurations in which modules span 2D surface of the ground plane, during locomotion, can be classified as 2D configurations, while configurations with modules spanning 3D space can be classified as 3D configurations. Common walking gaits in 2D and 3D configurations found in the literature are as reviewed in the following subsections.



(a) *Joint module* **(b)** *Attachment module*



(c) *Inchworm configuration*

Figure 2.17: *Inchworm modules and configuration. Source: [Wang et al., 2009]*

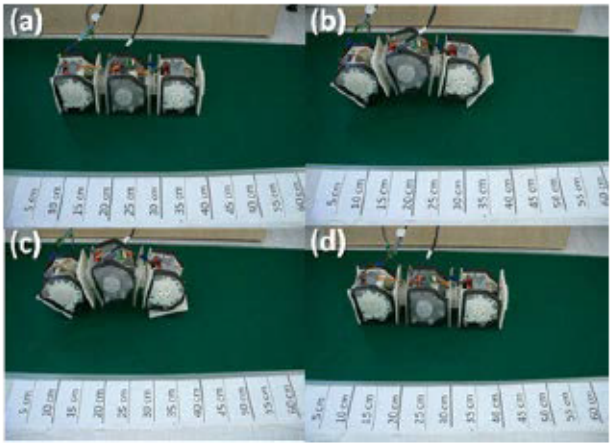


Figure 2.18: *Inchworm configuration using three Scout modules, performing inchworm gait. (a) start position, (b) and (c) contraction, (d) elongation. Source: [Russo et al., 2012]*

2D walker

In [Russo et al., 2012] five *Scout* modules are used in a 2D quadruped configuration (Figure 2.19), where four modules form limbs of the quadruped, while the fifth module is the torso module, holding the rest of the modules together. Here, each limb has one rotational DOF in the pitch-axis, and locomotion in any direction on a 2D plain is achievable in this configuration. The quadruped can move in forward and backward directions at a speed of , and the authors

have experimentally proven its ability to climb over obstacles high and at a slope of .

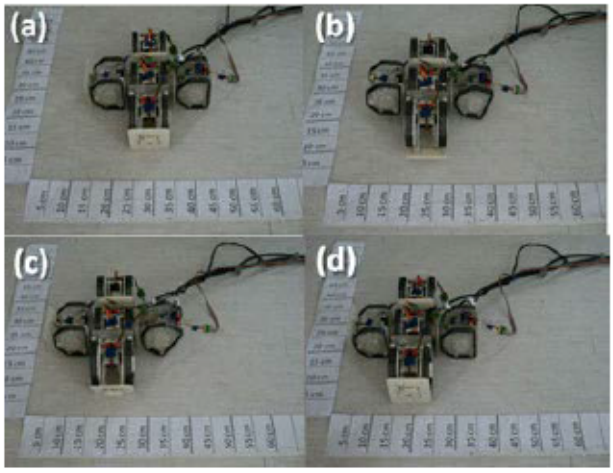
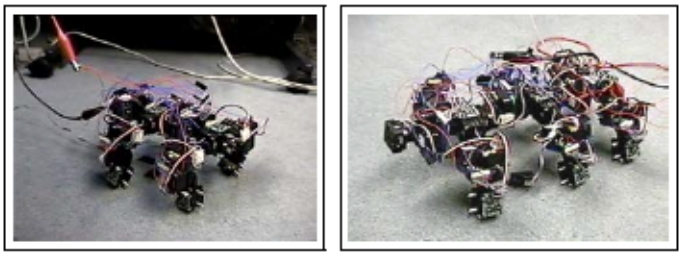


Figure 2.19: *Scout 2D quadruped configuration performin four-legged locomotion. Source: [Russo et al., 2012]*

3D walker

3D quadruped configurations have been presented in [Stoy et al., 2002] and [Pouya et al., 2010], where each limb has two rotational DOF. Each limb has one hip-joint, that oscillate in the pitch-axis, and one knee-joint, that oscillate in the roll-axis. Here, each limb needs to coordinate with other limbs in the configuration — Inter-limb coordination — as well as joints within a limb (hip joint and knee joint) needs to coordinate with each other — Intra-limb coordination — to produce walking gait. Spine modules in these configuration oscillate in the yaw-axis, enhancing the produced gait. In [Stoy et al., 2002], an hexapod configuration is created by appending the quadruped configuration with an extra spine module and two limb modules (Figure 2.20). Similar to the 2D walker, walking gait is achieved in these configurations when diagonal set of limbs oscillate in phase, while maintaining a phase-difference between the two sets of limbs.



(a) *Quadruped* **(b)** *Hexapod*

Figure 2.20: *CONRO modules configured as quadruped and hexapod walker. Source: [Stoy et al., 2002]*

An *H-Walker* is a 3D quadruped with multiple DOF limbs presented in [Shen et al., 2006] and [Kurokawa et al., 2003]. Here the spine module is static, used only for connecting the limb modules together (Figure 2.21).

Figure 2.21: *M-Tran modules in H-Walker configuration. Source: [Kurokawa et al., 2003]*

2.1.5.iii Rolling/Wheel-based locomotion

Rolling-track

Rolling-track gait can be achieved in a loop configuration, which is formed by connecting the two end modules of a linear configuration to each other, closing the loop. An example of a loop configuration is as seen in Figure 2.22. Experiments on this gait has been done in [Støy et al., 2003], [Shen et al., 2006], [Chiu et al., 2007], with loop configurations of varying sizes. Loop configurations, depending on the number of modules, can take the shape ranging from an hexagon to a circle. Locomotion in this configuration is achieved by continuously changing the shape of the configuration, by squeezing and realizing the shape of the loop. In [Chiu et al., 2007] steep slope-climbing with a loop configuration has been demonstrated, where the loop is collapsed to maintain a low center of gravity.

Helicoidal

In [Brunete et al., 2012b] authors have described experimenting with helicoidal gait on the heterogeneous chained robot MictoTub. The gait is achieved in linear configuration with a special module composed of two parts: a body and a rotating head (2.23a). When the head is continuously rotated, it pushed the body of the robot forward in a helicoidal motion. This gait has been tested inside vertical and horizontal narrow pipes, and in open air. This gait is very fast, but only locomotion in 1D is possible.

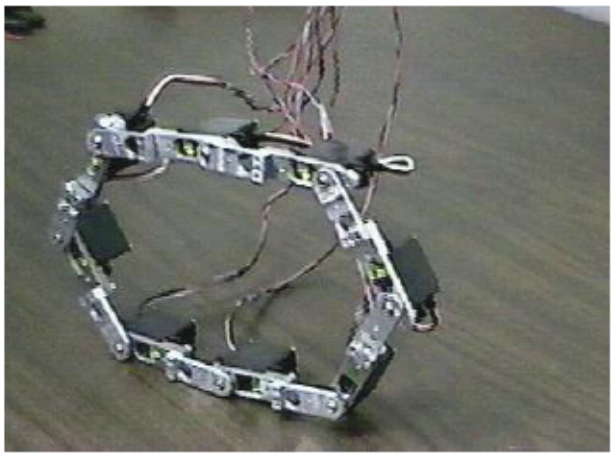
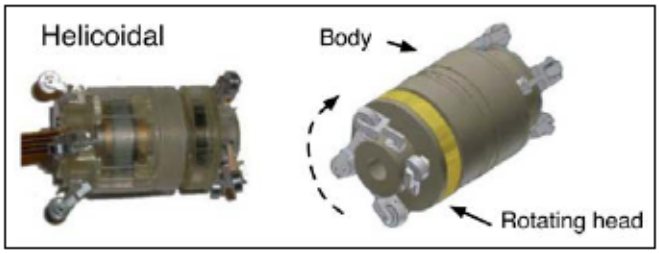


Figure 2.22: *CONRO modules in closed-loop configuration. Source:* `http://www.isi.edu/robots/conro/proto.html`



(a) *Helicoidal module*

(b) *Helicoidal gait in a horizontal pipe*

(c) *Helicoidal gait in a vertical pipe*

Figure 2.23: *Helicoidal module and gait. Source: [Brunete et al., 2012b]*

Differential drive

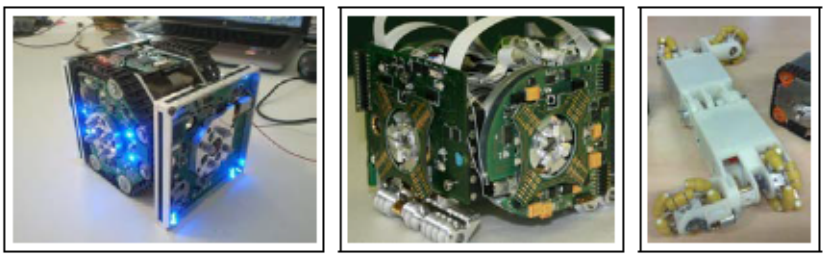
Differential drive is a two-wheeled drive system, where each wheel is actuated independently. Direction and rotation of locomotion comes about as a result of the speed and direction of rotation of the two independently controlled wheels. In [Kernbach et al., 2011c], track based, screw drive based and omnidirection-wheel based differential drive locomotion has been implemented in *Scout*, *Backbone* and *Active Wheel* MRs respectively (Figure 2.24), which gives independent

locomotion capability for individual modules, as well as macro-locomotion as part of a larger robotic organism.

A *Scout* module has two independently driven tracks on either sides (left and right) of its quasi-cubic shaped chassis, which gives it the ability to move forward, backward, turn right, turn left and to rotate on its own axis. The tracks are differentially driven, and provides *Scout* module the ability to move around independently on rough terrain, climb over small obstacles, pass over small holes and scale slopes of up to incline. In a multi-robot configuration mode, several *Scout* robots connected in a linear configuration, can coordinate together to produce linear-track-gait; resembling a multiple bogie train.

Two cylindrical screw drives placed in the front-bottom and rear-bottom side of the quasi-cubic chassis are used for independent locomotion in the *Backbone* MR. Two motors control the screw drives independently, and through differential drive, locomotion in forward, reverse, left and right directions, along with turning left and right are possible. Screw drives are mainly used for independent locomotion of a *Backbone* module in its environment, and for alignment during docking procedure between two modules, but it can also be used for macro-locomotion in multi-robot configuration mode (e.g.: In a *Backbone* modules based linear *Caterpillar* configuration, lateral movement for avoiding an obstacle can be achieved through coordinated actuation of screw drives of all the modules in the configuration).

Omnidirection-wheels have small disks around the circumference of the wheel, which can passively roll in the direction perpendicular to the rotational direction of the wheel. An *Active Wheel* module has two pairs of omnidirection-wheels, placed perpendicular to each other, and four motors independently controlling the four wheel. This gives an *Active Wheel* module the ability to independently move forward, reverse, right, left, turn right and left, rotate on its own axis, move on an arc, and also move in complex trajectories on a 2D plane through accurately controlling the four independent motors. Through coordinated actuation of omnidirection-wheels on multiple *Active Wheel* modules in a multi-robot configuration, the robot organism can move on 2D surface for transporting connected *Scout* and *Backbone* modules from one place to another.



(a) *Scout* **(b)** *Backbone* **(c)** *Active Wheel*

Figure 2.24: *Scout, Backbone and Active Wheel modules with tracks, screw driver and omnidirection-wheels respectively. Source: [Kernbach et al., 2011b]*

In [Davey et al., 2012] authors have implemented a wheel based differential drive system

in Self-assembling MOdular Robot for Extreme Shape-shifting (SMORES) MR for modules to move independently on relatively flat surfaces. The wheels, which also holds docking ports on its surface, are on the right and left sides of the cubic shape chassis of the module. Each module can move forward, reverse, turn left and right, and rotate on its own axis. This locomotion is necessary for modules to aggregate and align for self-assembly.

2.1.5.iv Self-reconfiguration based locomotion

In lattice-type MRs, locomotion is achieved through continuous self-reconfiguration. In a lattice structure, when modules at one end of the grid, by continuously connecting to and disconnecting with other modules on the outer edge of the structure, move to the other end of the grid, this results in the displacement of the entire structure. This kind of locomotion gives the notion of modules flowing on the ground, which is visually similar to locomotion of an amoeba, or to that of a puddle of water flowing on a flat surface.

In [Meng and Jin, 2011] lattice-type MR Cross-cube modules are continuously reconfigured to produce flowing locomotion. An advantage of this locomotion is that a configuration can morph its shape to avoid obstacles, or to move through narrow passage. In [Bonardi et al., 2012], Roombots (RB) modules move on a 2D grid, from an initial position to the goal position on the grid, through self-reconfiguration. Modules connect to the 2D grid surface, where each grid cell has a connector similar to those on the module. A module connected to a grid cell, through its DOF motor actuation, can reach its neighboring cell, then connect to that cell and disconnect from the previous cell. In this way, a module can move from cell to cell on a grid, moving from an initial to the goal location.

2.1.6 Controller

As per the literature, controllers in modular robotics are generally used for controlling locomotion and self-reconfiguration. These controllers can be broadly classified into Central Control (CC) type and Distributed Control (DC) type. In a CC type controller, individual modules in a configuration, receive high-level control signals from either a master module or from an external source. On the contrary, in a DC type controller, each module computes its own control signal, based on its sensor readings and inter-module communication. Homogeneity is another aspect of controllers in Modular Robot (MR), where if all the modules in a configuration has the exact same controller, then it is called as homogeneous controller, and non-homogeneous or heterogeneous otherwise. Scalability of a controller determines its ability to continue to function without any modifications, as the number of modules in the configuration grows.

2.1.6.i Locomotion Controllers

A wide variety of locomotion controllers for MR can be found in the literature. Some controllers are based on control theory, while many others are biologically influenced. Several of the locomotion controllers from the early days on MR, to more recent controllers are reviewed in the following subsections.

Gait-table controller

In [Yim et al., 2000] Mark Yim and colleagues propose a gait control table for caterpillar and rolling-track locomotion in Polybot MR, where each column of the table represents the action of a module in the configuration. So each row would then be a set of actions for the entire robotic configuration, at a certain point in time. Each module would have the entire gait control table, making the controller homogeneous. A master module would then communicate with the rest of the modules to synchronize the transition from one row to the next, making it a central controller. If the number of modules in the configuration changes, then gait table has to be modified accordingly, which means that the controller is not scalable.

Cellular Automata

Lal et al. in [Lal et al., 2006] have implemented a Cellular Automata model for controlling locomotion of a five limbed star-shaped MR (Figure 2.25), where rules are evolved for controlling the actuator of each module, distributedly, based on the state of the module's actuator, and that of its immediate neighboring modules' actuators, in the previous time step.

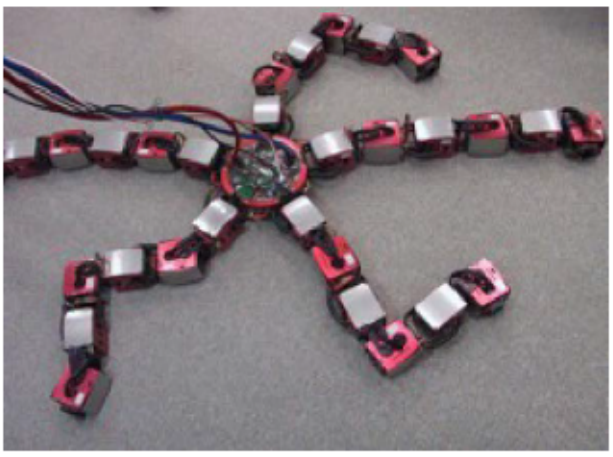


Figure 2.25: *The Brittle Star MR, with five limbs and six modules per limb. Source: [Lal et al., 2006]*

Hormone inspired controller

Shen et al. have used a biologically inspired method called Digital Hormone Method [Shen et al., 2000], [Shen et al., 2002] for adaptive communication of state information between modules, based on which a module can decide an action from a predefined gait-table, which results in the emergence of locomotion. A particularly interesting aspect of this work is that if the configuration of the robotic organism changes, or if one or some modules fail, with adaptive communication, the gait is adapted to suit the change in configuration. Digital Hormones have been successfully implemented on two different modular robotic platforms, CONRO [Castano et al., 2000], [Castano et al., 2002] and Superbot [Salemi et al., 2006]. In this control model, there is no central master module, and so it is a DC. Also, the controller is homogeneous and scalable.

Sinusoidal controller

Gonzalez-Gomez et al. demonstrate in [Gonzalez-Gomez and Boemo, 2005] how simple sinusoidal oscillators can be used on minimal configuration MRs, with two and three modules, for generating locomotion in one-dimension (1D) and two-dimension (2D) respectively, and in [Zhang et al., 2009] they study locomotion of two different kinds of caterpillar gaits, from a kinematic perspective, and replicate the same on linear configuration MRs, again using simple sinusoidal oscillators. The controller here is distributed, but without any communication between modules for coordination. So time synchronization between modules is achieved by the internal clock of the module's computing unit, and so the user needs to ensure booting all the modules at the same time. The controller is scalable, but not homogeneous since control parameters are predetermined based on the position of the module in a given configuration.

Role-based controller

In [Støy et al., 2003] Stoy et al. propose a role-based controller for locomotion in CONRO [Castano et al., 2000] [Castano et al., 2002] MR. A role-based controller consists of an oscillatory function, with a period T, which calculates the joint-angles of the module. Every time a module has completed a specific fraction d of period T, a message is sent to its children modules. When a module receives a message from its parent module, it resets its T to 0, making its action sequence delay by d compared to its parent. This delay introduces a phase-shift of $-$ radians to the module's joint angle(s) sequence, and thus the needed coordination between modules to produce the intended gait. This mechanism works in chain-type configurations for producing caterpillar, sidewinder and rolling-track gaits, where all the modules have to perform the same sequence of actions, albeit starting at different points in time.

For producing more complex quadruped and hexapod walking gaits in legged modular robotic configurations, modules have to perform different sequence of actions based on their location in the configurations (e.g.: Spine-module and leg-module in a quadruped configuration would have to perform different sequence of actions). So, to facilitate this, role-based controller is further

enhanced in [Stoy et al., 2002] by adding a role-selection mechanism, a set of different oscillatory functions and a set of different delays . In the enhanced controller, a module chooses its role based on the role of its parent controller, the connector of the parent module it is connected to, a set of its connectors connecting it to children modules, and the role of its children modules. The authors demonstrate the scalability of the controller by applying the same controller on both a six-module quadruped configuration, as well as on a nine-module hexapod configuration — which is conceived by adding an extra spine-module and two additional leg-modules to the quadruped configuration — and successfully producing walking gaits in both the configurations. The controller is distributed as there isn't a single central module used for synchronization, as well as homogeneous since all the modules have the exact same controller.

Central Pattern Generator

In [Spröwitz et al., 2008] Ijspreet et al. at the Biorobotics Laboratory, Swiss Federal Institute of Technology in Lausanne (EPFL), have used Central Pattern Generator (CPG) [Ijspeert, 2008] for producing locomotory oscillations on their modular robotic platform *YaMoR* [Moeckel et al., 2005] (Figure 2.26), among other robotic platforms. In [Pouya et al., 2010] they have tried similar CPGs for producing both oscillation and rotation in Degree of Freedom (DOF) actuators of their second generation modular robotic platform Roombots (RB) [Sproewitz et al., 2009] [Spröwitz, 2010]. CPGs are specialized neurons found in the spinal cord of vertebrates, that have the ability to produce rhythmic output without rhythmic sensory or central input. Mathematical model of CPGs used for controlling locomotion in MRs are usually one or two CPG neurons per module, which are coupled in different ways, based on the configuration, with similar neurons of other modules in a given configuration. CPGs were first successfully used on a modular robotic platform by Kamimura et al. in [Kamimura et al., 2003], where they use it for producing oscillations for adaptive locomotion on their M-TRAN MR.

(a) *Yamor module* **(b)** *Linear configuration* **(c)** *Quadruped configuration*

Figure 2.26: *Yamor MR. Source:* `http://biorob.epfl.ch/cms/page-36377.html`

The CPG based controller is of distributed control type, as there isn't a single module responsible for coordination among modules, but modules synchronize their action based on actions of other connected modules in a distributed manner. An important aspect of CPG

based controller is how CPG of one module is coupled with CPGs of other modules in a given configuration. This coupling is based on the morphology of the configuration, so it is fixed and predetermined. Control parameters of a CPG differs from module to module, based on the module's position in the configuration. So this controller can neither be classified as homogeneous nor as scalable.

Heterogeneous layered controller

Brunete et al. in [Brunete et al., 2012a] have developed a multi-layered control architecture for the heterogeneous multi-configurable MR *Microtub*. The control architecture consists of three layers: high-level central controller (CC), low-level behavior-based embedded controller and a middle-level heterogeneous interpreter connecting the high-level and low-level controllers. The CC is behavior-based as well, and controls the modular robotic configuration as a single entity, collecting sensory information from modules, processing it and sending situation and action commands to the modules. It also acts as a planner and helps modules synchronize their actions. The CC can either be off-board on an external PC, as it was in [Brunete et al., 2012a], or on-board as a part of one of the modules.

The low-level controller is a set of individual behaviors, which allows the module to react in real-time, independent of the CC. Examples of low-level behavior control include sensing and acting on external and internal stimuli, such as detecting an obstacle and adapting the shape accordingly to avoid the obstacle, and turning off an actuator on sensing overheating, respectively. Other behaviors include controlling the module's actuator for producing locomotion, communicating with connected modules, etc.

The heterogeneous middle-level layer acts as an interpreter between the CC and the low-level controller. Control commands sent by the CC are identical to all the modules, which themselves are heterogeneous. So when a high-level command is received by a module, it is processed by its middle-level interpreter, and translated to module specific instructions. Similarly, when a module needs to send a message to the CC or other modules, this too is handled by the middle-level interpreter.

The layered controller is semi-distributed, as some low-level actions can be taken by the module in real-time, independent of the CC, but high-level control commands and synchronization are still controlled by the CC. It is heterogeneous because middle-level and low-level controllers are module specific, but scalable because of the semi-distributed and layered architecture of the controller.

Artificial Neural Network based controller

Lal et al. in [Lal et al., 2008] have implemented an Artificial Neural Network (ANN) model as a locomotion controller for their six legged brittle star MR, where each module is modeled as a neuron in a fully connected ANN. Neurons sum their weighted input stimulus, which is

the actuator's phase angle that they share locally or globally based on their location in the configuration, and use a sinusoidal activation function to determine the next step. The authors have used Genetic Algorithm (GA) for evolving optimal synaptic weight vector of the ANN.

This controller is distributed and homogeneous, as all the modules decide their own action, and each module start with the same initial parameters. The parameters of the ANN (synaptic weights) are learned for the specific morphology of the robot, through evolution. By extending the size of the robot, either by increasing the number of legs or by increasing the number of modules in each leg, through a similar evolutionary process, new gaits for the modified configuration can be learned, making the controller scalable as well.

2.1.6.ii Self-reconfiguration Controllers

Self-reconfiguration is an important aspect of a Self-Reconfigurable Modular Robot (SRMR), which gives it the ability to change its morphology to suit its environment and/or current task. In lattice-type and hybrid-type SRMR, locomotion is achieved through the process of self-reconfiguration.

Morphogenesis inspired controller

In [Meng and Jin, 2011] Meng et al. have proposed a hybrid-hierarchical-layered controller for self-reconfiguration, inspired by biological multi-cellular morphogenesis. Morphogenesis is a biological process through which the shape of an organism is determined. During the stage of embryonic development of an organism, genes contained in individual cells are expressed, resulting in various cellular functions. This expression of genes is regulated by proteins produced by the same gene or by other genes in the cell, or by other genes in neighboring cells, through intra-cellular or inter-cellular diffusion, forming a complex web of Gene Regulatory Network (GRN). In this work, the authors consider a module equivalent to a biological cell, connected to other modules to form a multi-cellular organism.

The controller consists of three layers: pattern generation layer, pattern formation layer and low-level hardware dependent layer. The first layer responsible for generating the pattern (modular robotic configuration) is a rule-based controller, where a pattern is represented as a three-dimensional (3D) grid occupancy graph, and encoded as a lookup-table. Based on the environmental constraints and task at hand, by following a set of rules, modules can modify this table to bring about a change in the pattern. For example, when an individual module senses an obstacle in the environment through its local sensor, it can diffuse this information in the network through communication, to bring about a global change in the pattern.

Once a target pattern is set or adapted in the lookup-table, modules can then act independently to converge to the global pattern. By setting any of the modules as the origin in the occupancy graph, modules can then localize themselves in the configuration through local communication. Based on the relative position and the desired target pattern, modules can then

produce and diffuse different kinds of proteins through local communication. This is the pattern formation, second layer of the hierarchical controller. A module can produce and diffuse positive proteins to attract modules to occupy one of its empty neighboring space in the grid, or negative proteins to empty one of its neighboring space. The third layer is hardware specific, and controls the module's actuator and connectors for self-reconfiguration.

The proposed controller has successfully been tested on Cross-cube [Meng et al., 2010] MRs, in a physics based simulator. The controller is distributed, as modules decide their action independently based on their sensory information, and through local communication with neighboring modules. It is homogeneous, as all the modules have the exact same controller. In the article, the authors have also experimentally proven the scalability of the controller by successfully testing it on varying-size configurations.

Motor primitives based controller

Bonardi et al. in [Bonardi et al., 2012] have proposed a locomotion controller for the RB [Sproewitz et al., 2009] [Spröwitz, 2010] MR, achieved through self-reconfiguration. Locomotion in this context refers to the process of a RB module moving from an initial position to a goal position on a two-dimensional (2D) grid. Each grid position contains an Active Connection Mechanisms (ACM) connector, similar to the ones on the RB modules (Figure 2.27).

Each module has three continues rotational DOF, through which a module can translate by a distance of one unit on the grid. Each such action, leading a module to translate by one grid unit, is termed as an Atomic Motor Primitive (AMP). Due to the kinematic constraints of the module, a module can translate to only two out of eight possible neighboring positions on the grid. So, to move to any of the eight possible neighboring grid positions, a module needs to perform a set of AMPs, and a concatenation of one or more such AMPs is termed Composed Motor Primitive (CMP).

*D** algorithm is used as a high-level planner for planning a path for the module to travel between the initial and the goal position on the grid. A low-level planner is proposed which, based on the path found by the *D** algorithm, decomposes the path into a set of CMPs to traverse the path. Both the high and the low-level controllers take static obstacles (e.g.: Other modules) into consideration while planning. The proposed planner, due to the kinematic constrains of the module, does not have a solution for every possible grid-world configuration containing obstacles. If the planner finds a set of CMPs, it is not necessarily optimal either.

The authors have successfully tested the controller, both in simulation and on the real robot. In simulation, fixed size grid-world with fixed and random size and number of obstacles, along with random initial and goal positions were generated. Out of 300 total experiments, the authors report a success rate in the planner finding a path from the initial to the goal position. All the experiments in the article have been performed with a single RB module, so it cannot be determined if the controller is distributed or central, homogeneous or heterogeneous, and if it is scalable.

Figure 2.27: *A single RB module in a 2D grid world, where each grid contains an ACM connector. Source: [Bonardi et al., 2012]*

2.2 Humanoids locomotion

Humanoid locomotion is an ongoing topic under robot locomotion, and a wide range of approaches towards bipedal locomotion can be found in the literature. Some approaches such as [Kajita et al., 2003], [Wieber, 2006], [González-Fierro et al., 2014] and [Monje et al., 2013] make use of reduced models for obtaining the dynamics, while several other approached such as [Nagasaka et al., 1999], [Yamane and Nakamura, 2003] and [Khatib et al., 2004] make use of a distributed mass formulation. While many of these approaches are based on Zero Moment Point (ZMP) controllers, an interesting alternative is based on bipedal locomotion through CPGs.

CPGs, as reviewed in section 2.1.6.i, page 36, are specialized neurons found in the spinal cord of vertebrates, that have the capability of producing rhythmic output without rhythmic sensory or central input. It has been hypothesized that during animal locomotion, there is a feed-forward mechanism that activates muscles using signals generated by the CPGs. Locomotion in humanoid robots based on CPGs have been successfully implemented in [Shan and Nagashima, 2002] and [Endo et al., 2008]. In [Or, 2010], a CPG controller is combined with a ZMP controller, to obtain a stable bipedal gait. Authors of [Shan et al., 2000] proposed a CPG controller, where weights of the CPG are optimized optimized through GA. Here fitness function is formulated as a combination of ZMP, altitude of the robot and walking speed, with the ZMP component implicitly ensures stability. Authors of [Nakanishi et al., 2004] present another CPG based approach, where stable bipedal gait trajectories are generated based on human demonstrations.

Another parallel and interesting approach towards bipedal locomotion in humanoid robots involves obtain prerecorded data from human bipedal walking and then transfer the locomotion behavior on to a robot. In [Mombaur et al., 2010] Mombaur et.el, have used inverse optimal control on motion capture data of human bipedal walking, and have successfully implemented it on the Humanoid Robotics Platform (HRP)-2 platform [Kaneko et al., 2002]. In [Morimoto and Atkeson, 2007], the authors have implemented a Reinforcement Learning (RL) algorithm for bipedal walking, in which the robot learns how to appropriately modulate observed gait pattern

for producing a stable gait. In [Miura et al., 2009], bipedal walking and turning locomotion are implemented on the HRP-4C humanoid robot. It is achieved by adapting gait parameters such as step length, gait speed, rotational angle during turn, etc., of the human motion capture data.

2.3 Morphological Computation

The term **Morphological Computation** was coined by Chandana Paul [Paul, 2006], and it refers to outsourcing of computation to morphology and material property of an agent. Traditionally, intelligence and cognition have been associated solely with the brain of an agent, without taking into consideration its body and morphology. But some articles like [McGeer, 1990], [Iida and Pfeifer, 2004] have shown that intelligence can arise as a result of interaction between the brain, body and the environment the agent is embedded in.

2.3.1 Passive dynamic walking

One on the earliest and a classical example of Morphological Computation was demonstrated by Tad McGeer [McGeer, 1990] with the passive dynamic walker, which is a mechanical system that can walk down an inclined ramp with a slope of a certain degree, without any actuator, power-supply, sensing or computation. In a sense, this agent is completely brain less, and the behavior of walking is produces solely based on its morphological properties, which is specifically tailored to produce the walking behavior. Energy requirement for walking in this design is minimal, as walking is produced solely by gravitation, and the walking gait produced seems very human like.

The drawback of the passive dynamic walker is that the conditions in which it operates, also called as its ecological niche, is very narrow. It means that the passive dynamic walker would cease to operate if anything in its environment or its morphology, like the slope on the ramp or the material property of its feet, respectively, is changed. *Denise*, an augmented passive dynamic walker with actuators, power supply and controller, was created by Martijn Wisse [Wisse, 2004] at Techinical University of Delft. Morphology of *Denise* was adapted to walk on level surface, and the emerged walking gait was very human like and in-turn very energy efficient.

2.3.2 Puppy

Puppy [Iida and Pfeifer, 2004], a quadruped robot, built at the Artificial Intelligence lab, University of Zurich, mimics the morphology of a canine. Puppy has four limbs, and twelve joint (four each at the shoulder/hip, elbow/knees and ankle) in total. There are eight standard digital servomotors at the shoulder/hip and elbow/knee joints, and the ankle joints are connected via passive springs. A simple sinusoidal position controller was applied to each of the four shoulder/hip joints, wherein the motor commands for the two shoulder joints are symmetrical, and the motor commands for

(a) *The classic passive dynamic walker* **(b)** *Denise*

Figure 2.28: *Passive dynamic walkers. Source:* `http://ruina.tam.cornell.edu/research/topics/robots/index.php`

the two hip joints are symmetrical as well. The elbow/knee joints are fixed to a constant position, and the ankle joints are passive. When evaluated by placing the robot on the ground, the robot displays a running gait, which is a result of the morphology of the robot (its shape and the passive spring joints), controller (parameters of the sinusoidal controller) and the environment (friction of the ground surface and gravity) the robot interacts with.

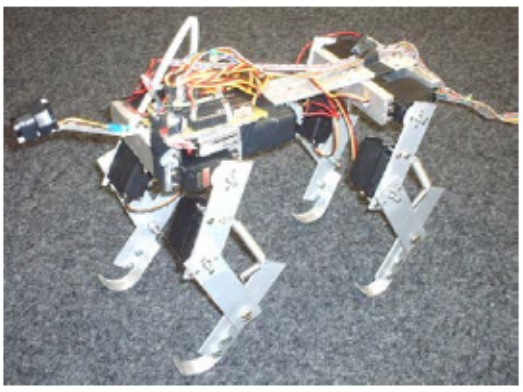


Figure 2.29: *The quadruped robot Puppy. Source: [Iida and Pfeifer, 2004]*

2.3.3 WalkNet

Studies of insect locomotion [Cruse, 1990] [Cruse et al., 2002] has shown that the coordination between legs of an insect during walking, comes about as a result of coupled local neural circuits, and that there is no central controller that coordinates the legs during walking. When an insect, standing on the ground, tries to move forward by pushing one of its legs backward, as a

consequence of its embodiment, there is force applied on the rest of the stationary legs, and this information, in the form of joint angle measurement, can be used by the insect as a form of global communication between legs for producing locomotion, even without there existing a central controller that coordinates leg movements. Inspired by this, a distributed neural network architecture for controlling a six legged robot was developed [Dürr et al., 2003]. This is a very strong example of Morphological Computation, as the communication, and in turn the needed coordination between legs for locomotion, comes about as a result of the interaction between the insect and its environment.

CHAPTER 3

Locomotion Controllers for Modular Robots

Introduction

One of the goals of the work presented in this thesis is to develop locomotion controller for Modular Robot (MR) through morphology, evolution and learning. In this chapter, locomotion in MR through morphology is presented. First, five different modular robotic configurations used for testing locomotion controller in the rest of the thesis are presented. Kinematics and gaits achievable in each of the five configurations are explained. Two classes of controllers are presented in this chapter: periodic function based controllers and morphology based controllers.

Two kinds of periodic function based locomotion controllers are presented: Sinusoidal controller and Fourier controller. Sinusoidal oscillators as locomotion controllers for MRs have been previously implemented in [Gonzalez-Gomez and Boemo, 2005] and [Zhang et al., 2009]. To set as a benchmark for comparing with other controllers developed in this thesis, sinusoidal controllers are implemented and evaluated on all of the five modular robotic configurations. The second controller under this category is the Fourier controller, which is developed as part of this thesis, and is based on Fourier series.

Morphology refers to the form and structure of a biological organism, and in the context of MRs, it refers to the topology of the modular robotic configuration. Under morphology based locomotion controllers category, two kinds of controllers are developed: an Artificial Neural Network (ANN) based controller and an inverse sine function based controller. Although the parameters of all four controllers presented in this chapter are optimized through Genetic Algorithm (GA), locomotion through evolution is presented in a later chapter (chapter 5, page

105).

3.1 Robot configurations

Y1 MR is the platform used in this thesis for all the experiments concerning MR locomotion. Two *Y1* modules can be connected together in several different ways, and each module can be connected to at most four other modules. Connection mechanism supported by *Y1* modules is very basic, and only static connections between modules are possible, making self-reconfiguration among modules not possible. Modules can be connected to each other either using nuts and bolts, or using zap-straps.

During the course of this thesis, simulated versions of five different modular robotic configurations have been experimented with. Gaits achievable by each robotic configuration has been studied by applying simple phase-differed sinusoidal oscillators to modules in a given configuration. In the following subsections, each configuration has been described in detail.

3.1.1 *Minibot*

This is a two-module, one-dimensional (1D) configuration (Figure 3.1), wherein modules are connected to each other in series, and according to [Gonzalez-Gomez and Boemo, 2005] this is the smallest possible configuration that can produce locomotion. Only locomotion in 1D can be achieved in this configuration, wherein the robot can either move in forward or backward/reverse directions. Applying simple sinusoidal oscillators to modules, with predefined phase-difference ( ), produces a caterpillar gait that resembles a traveling sine-wave. The phase-difference determines the direction of locomotion, with the robot moving in the direction of the module that has a negative phase-difference with respect to the other module. Locomotion cannot be achieved in this configuration if the phase-difference between modules is either around or around .

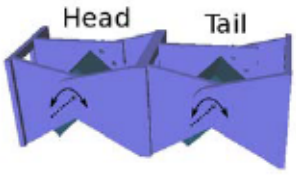


Figure 3.1: *Two module Minibot configuration.*

3.1.2 *Tripod*

This is a three-module symmetric configuration, wherein modules are connect to each other at an angle of , as shown in Figure 3.2. It is a two-dimensional (2D) configuration and it can move

on a 2D surface in three possible directions, as well as rotate on its own axis. When modules are applied with sinusoidal oscillators, with two modules oscillating in phase and the third module oscillating with a phase-difference of , the robot moves in the direction of the module oscillating out of phase, and in the opposite direction if . When no two modules oscillate in phase, while phase-difference between pairs of adjacent modules is (E.g. , ,), the robot rotates on its own axis in clockwise direction.

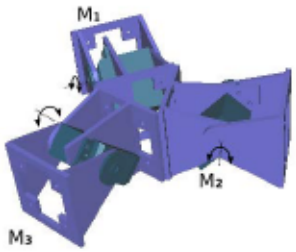


Figure 3.2: *Three-module Tripod configuration.*

3.1.3 *Quadropod*

The *Quadropod* configuration is an extension of the *Tripod* configuration, which has an additional module, and the angle between modules is (Figure 3.3). It is a symmetric two-dimensional (2D) configuration, which can move in eight possible directions on a 2D surface, depending on the phase-difference between oscillating modules. If two opposite modules oscillate in phase, while the other two modules oscillate with a phase-difference of , then the robot moves in the direction perpendicular to the modules oscillating in phase. If two pairs of adjacent modules oscillate in phase, with a phase-difference between these pairs (E.g. and), then the robot moves in the direction diagonal to itself. When no two modules oscillate in phase, while phase-difference between pairs of adjacent modules is (E.g. , , ,), the robot rotates on its own axis in clockwise direction.

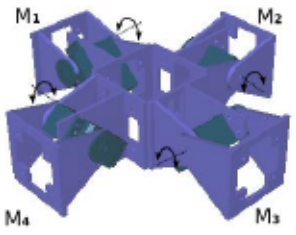


Figure 3.3: *Four-module Quadropod configuration.*

3.1.4 *Y-bot*

The *Y-bot* configuration, as shown in Figure 3.4, is an extension of the *Tripod* configuration, which is conceived by adding an additional *Y1* module (*Tail*) to one of the three modules of the *Tripod* configuration, which then becomes the *Spine* module. Locomotion in two-dimension (2D) is possible with this configuration, although only locomotion in one-dimension (1D) is being focused on in this work. When modules are applied with phase-differed sinusoidal oscillators such that there is increasing phase-difference between modules, starting from that *Head* modules to the *Tail* module, while the two *Head* modules oscillate in phase (E.g. , ,), the robot moves in the direction of the *Tail* module. The robot moves in the opposite direction, if this case is reversed (i.e. , ,).

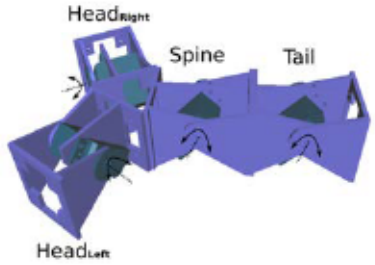


Figure 3.4: *Y-bot configuration.*

3.1.5 *Lizard*

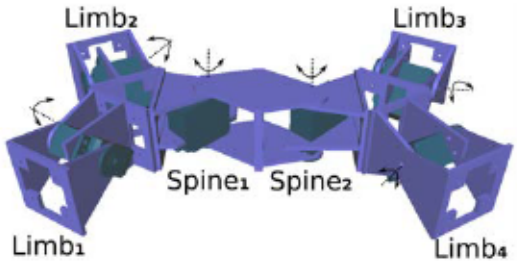


Figure 3.5: *Lizard configuration with four Limb modules and two Spine modules.*

Lizard, as shown in Figure 3.5, is a six module configuration that has four *Limb* modules and two *Spine* modules. This configuration is formed by connecting two *Tripod* configurations together, and then rotating the *Spine* modules by and respectively, along the roll-axis of the configuration. This makes the two halves of the robot — considering modules , , and , as one half, and modules , and as the other half — mirror images of each other. When modules in this configuration are actuated with phase-differed sinusoidal oscillators, as presented in Table 3.1 (which are derived empirically), it results in a

quadruped walking gait, resembling that of a lizard. The two spine modules wiggle side to side with a phase-difference between them [1], while the limb modules move up and down, resulting in a walking gait.

Module	Phase Angle

Table 3.1: *Phase-relation between module pairs in a Lizard configuration with respect to the module Limb-1.*

3.2 Periodic function locomotion controllers

Locomotion in general, whether gallop of a horse, flapping wings of a bird, or bipedal walking of a human, can be seen as repetitive and coordinated movement of limbs, through which the desired gait emerges. Looking at locomotion as a collection of oscillators, phase-relation between these oscillators determines the generated gait. So, in this section, two different periodic functions — Sinusoidal function and Fourier series — are implemented as locomotion controllers for Modular Robot (MR)s.

3.2.1 Sinusoidal controller

In this thesis, a class of locomotion controller for Modular Robot (MR)s have been developed. To analyze locomotion of each of the five modular robotic configurations, and to set a benchmark for comparing other locomotion controllers, that are present in the rest of the thesis, we first implemented a sinusoidal controller. Sinusoidal controller for MR locomotion has been previously investigated in [Gonzalez-Gomez, 2008], but with a focus on linear modular robotic configurations, and by hand-tuning oscillation parameters. In this work, we applied a sinusoidal oscillator per module Equation 3.1, and oscillation parameters are optimized through Evolutionary Algorithm (EA).

(3.1)

[1] Although the control signal to the two spine modules are identical, they oscillate with a phase-difference of between them, since the two modules are connected as mirror images of each other.

where is the total number of modules in the configuration, is the amplitude, is the frequency, is the phase and is the offset of the oscillator for the module in the configuration.

Each module is controlled independently by Equation 3.1, with amplitude and offset of its oscillation determined by parameters and respectively. Oscillation frequency is common among all modules in a given configuration. Phase-shift of a module's oscillation is determined by the parameter , and relative difference in phase-shift value among modules determines the emerged gait in the robotic configuration. To test this control model, each of the five configurations are setup in the simulation environment individually, starting with a set of random control parameters, and optimal control parameters are evolved through EA. For each configuration, a random set of fifty candidate solutions are initialized, where each candidate solution is a vector of all the oscillation parameters for all the modules in the given configuration. So, for the case of *Minibot* configuration with two modules, each candidate solution consisted of two amplitude parameters , two offset parameters , , two phase-shift parameters , , and one frequency parameter . The range for each parameter to be optimized is set to values as shown in Table 3.2. Each candidate solution is evaluated for a period of , with the objective function for parameter optimization being the absolute distance between the start and the finish position of the robot, at the end of the evaluation.

Parameter	Min.	Max.

Table 3.2: *Range of sinusoidal controller parameters for optimization.*

For parameter optimization through EA, a combination of Genetic Algorithm (GA) and Evolutionary Strategy (ES) has been implemented, which is further explained in subsection 5.3.1, page 125.

3.2.1.i Evaluation

Post evolution, best performing individuals of the final generation for each of the five configurations are evaluated for a period of seconds ( ). Evaluation results are as presented in the following subsections.

Minibot

In the *Minibot* configuration, the best evolved controller, parameters of which are as presented in Table 3.3, produced a caterpillar gait. The *Head* module, with an amplitude of

and a significantly larger negative offset of , oscillates between , while the *Tail* module, with an amplitude of and a positive offset of , oscillates between . The *Head* and the *Tail* modules oscillated with a phase-difference of between them. The *Head* module pulls forward, while the *Tail* module pushes off the ground-surface, propelling the robot forward and moving in the direction of the *Head* module, at an average speed of . One gait-cycle of this locomotion is as shown in Figure 3.6.

Parameter	Module	
	Head	*Tail*

Table 3.3: *Sinusoidal controller parameters of the best performing individual for Minibot configuration, optimized through EA.*

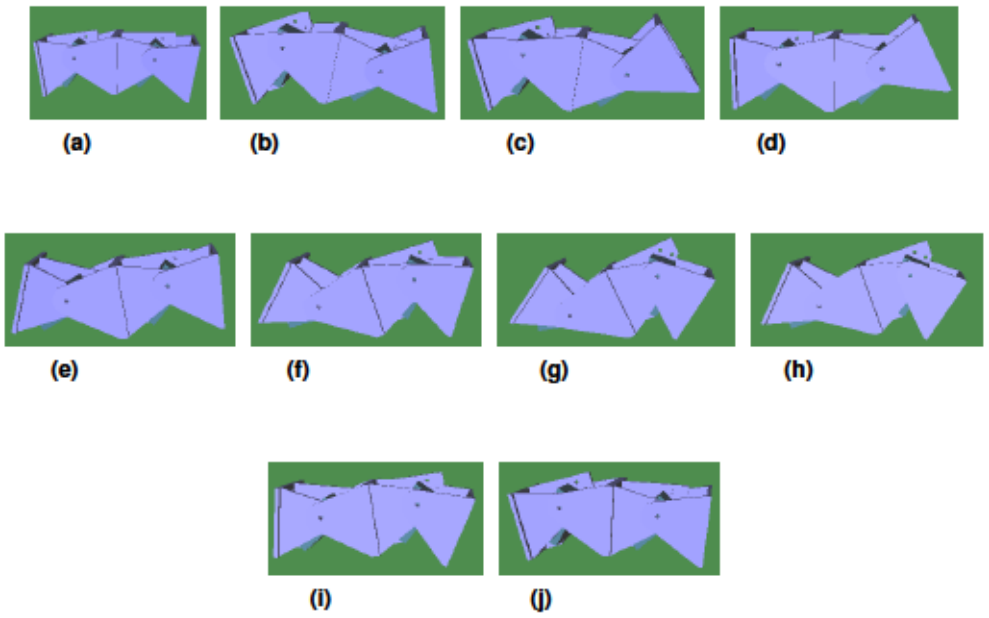


Figure 3.6: *Scree capture of one gait-cycle of the Minibot configuration, evaluated with the best evolved Sinusoidal controller.*

Tripod

In *Tripod* configuration, with the best evolved controller (Table 3.4), modules and oscillate in phase[23], while there exists a phase-difference of between modules and the other two modules (and). Module , with an amplitude of and a negative offset of , oscillates between . Modules and , with amplitude of and respectively, and offset and respectively, oscillate between and respectively. Modules and push forward simultaneously, while module pulls forward, making the *Tripod* configuration move in the direction of module at an average speed of . One gait-cycle of this locomotion is as seen in Figure 3.7.

Parameter	Module		

Table 3.4: *Sinusoidal controller parameters of the best performing individual for Tripod configuration, optimized through EA.*

Quadropod

Modules through in the *Quadropod* configuration, with the best evolved controller (Table 3.5), oscillate in the following range,

:

:

:

:

Modules and oscillate in phase[footnote 3], while modules and oscillate with a phase-difference of , and modules and oscillate with a phase-difference of between them. This phase-relation among modules result in a gait where modules and push, while modules and pull in south and west directions respectively, propelling

[2]Modules and oscillate with phase-shift of and respectively, so with a phase-difference of between them.

[3]A phase-difference of is small enough to be considered insignificant in the context of this thesis.

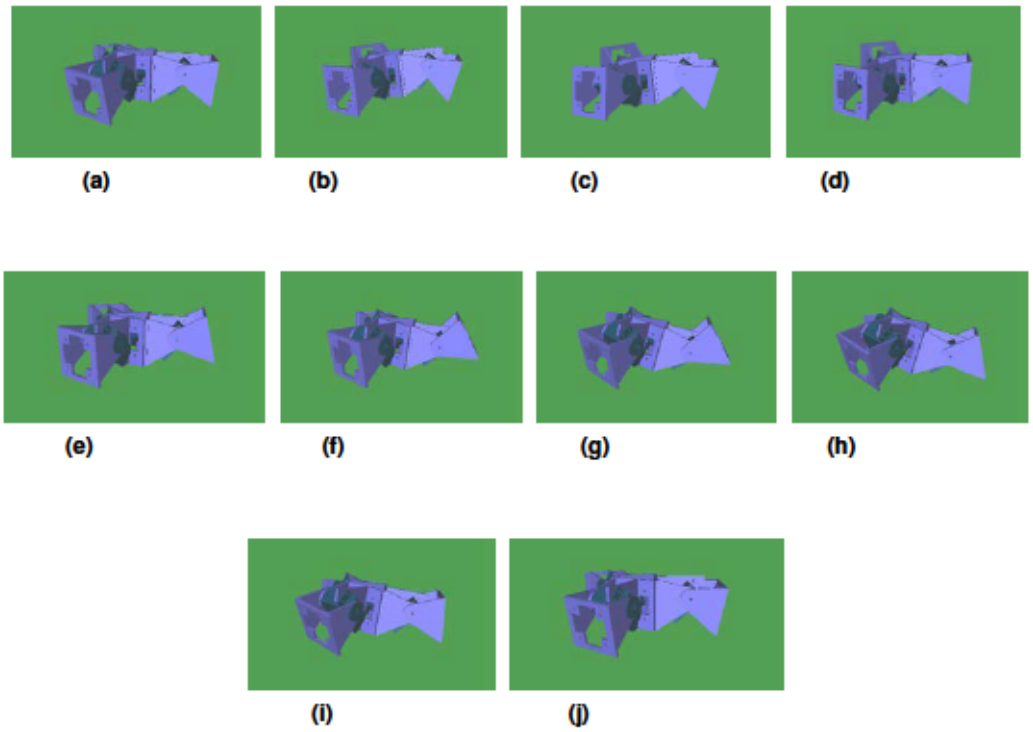


Figure 3.7: *Scree capture of one gait-cycle of the Tripod configuration, evaluated with the best evolved Sinusoidal controller.*

the robot in the south-west direction, relative to the robot's initial orientation (Figure 3.8). Modules and coordinate to move the robot one step in the east direction, while modules and coordinate to move th robot one step in the south direction. There exists a phase-difference of between the two pairs, so the robot moves one step east followed by one step south, in a zig-zag pattern, at an average speed of .

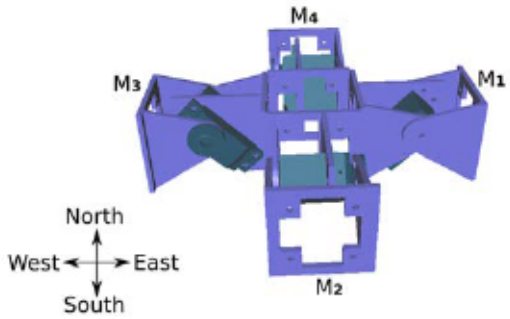


Figure 3.8: *Initial orientation of the Quadpod configuration.*

Parameter	Module			

Table 3.5: *Sinusoidal controller parameters of the best performing individual for Quadropod configuration, optimized through EA.*

Y-bot

Modules in the *Y-bot* configuration, with the best evolved controller (Table 3.6), oscillate in the following range,

:

:

:

:

Modules and oscillated in phase[footnote 3], while there exists a phase-difference of between the modules and the module, and a phase-difference of between the module and the module. This phase-relation among modules result in a traveling sine-wave, starting from the modules and moving in the direction of the module, resulting in a caterpillar gait, propelling the robot in the direction of the module. One gait-cycle of this locomotion is as seen in Figure 3.9, and the average speed of locomotion achieved in this gait is .

Parameter	Module			

Table 3.6: *Sinusoidal controller parameters of the best performing individual for Y-bot configuration, optimized through EA.*

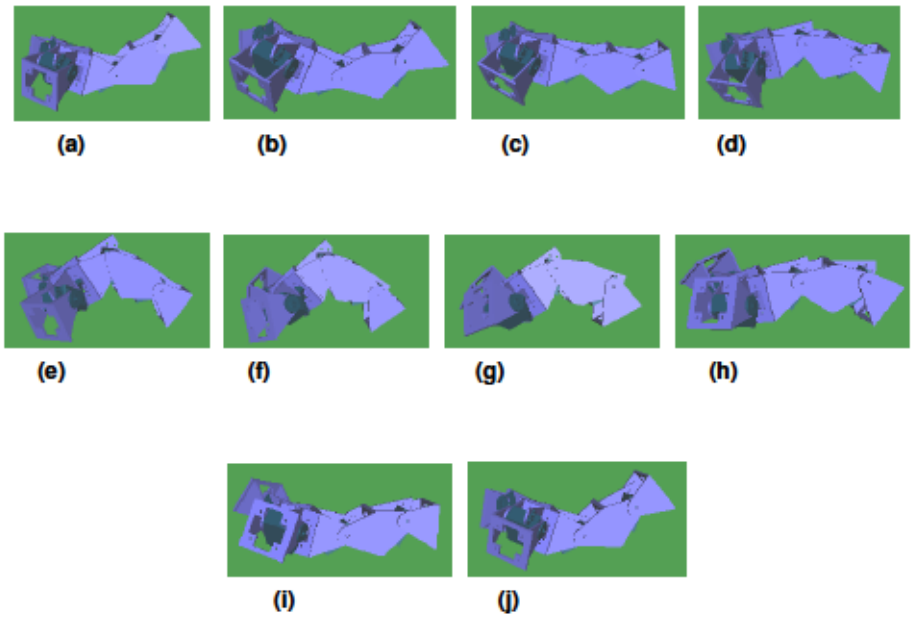


Figure 3.9: *Scree capture of one gait-cycle of the Y-bot configuration, evaluated with the best evolved Sinusoidal controller.*

Lizard

With the *Lizard* configuration, two separate gaits emerged through two independent but identical EA epochs[4]. The first gait resembles a reptilian like forward-walking gait, while the second gait resembles a crab like lateral-walking gait. Modules in the *Lizard* configuration, when evaluated with the best evolved forward-walking Sinusoidal controller (Table 3.7), oscillate in the following range,

:

:

:

:

:

:

[4]EA parameters like population size, selection/crossover type/size/rate, mutation rate/range, etc. are all exactly same between the two epochs

Parameter	Module					

Table 3.7: *Sinusoidal controller parameters of the best performing individual for Lizard configuration that produced forward-walking gait.*

Modules and oscillate with a phase-difference of between them, while and oscillate in phase[footnote 3]. There exists a phase-difference of between modules and . Module oscillates in the range of , without ever making contact with the ground surface, and so does not directly contribute to the gait. The gait emerges as module pulls, while modules and push off the ground surface. The *Spine* modules, oscillating in the yaw-axis, amplify the actions of the *Limb* modules. Phase-difference between all module pairs in this gait are as provided in Table 3.8. The average speed of locomotion achieved with this gait is .

Module						

Table 3.8: *Phase-difference between all modules pairs in the Lizard configuration, with the best performing forward-walking Sinusoidal controller.*

When evaluated with the best evolved lateral-walking Sinusoidal controller (Table 3.9), modules in the *Lizard* configuration oscillate in the following range,

:

:

:

:

:

:

Parameter	Module					

Table 3.9: *Sinusoidal controller parameters of the best performing individual for Lizard configuration that produced lateral-walking gait.*

In this gait, modules and oscillate with a phase-difference of , while the two *Spine* modules oscillate in phase[5]. Module oscillate in the range of , without ever making contact with the ground surface, and so does not directly contribute to the gait. The lateral-walking gait emerges as module pulls, while modules and push off the surface, aided by the rowing motion of the *Spine* modules in the yaw-axis. The main difference between this gait and the forward-walking gait is the phase-relation between the two *Spine* modules. In the forward-walking gait, there exists a phase-difference between the two *Spine* modules, while in the lateral-walking gait, the two *Spine* modules oscillate in phase. Phase-difference between all module pairs in this gait are as provided in Table 3.10. Average speed of locomotion achieved with this gait is .

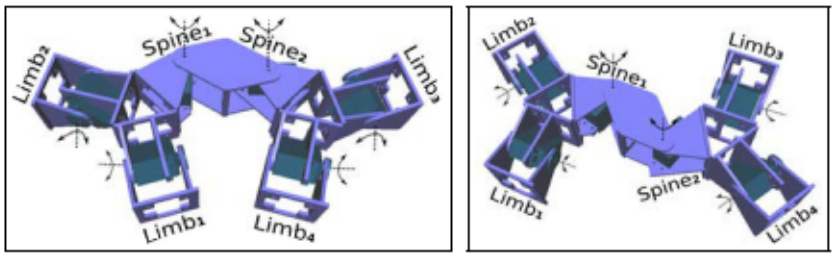
Module						

Table 3.10: *Phase-difference between all modules pairs in the Lizard configuration, with the best performing lateral-walking Sinusoidal controller.*

3.2.1.ii Discussion

Sinusoidal oscillator as controller, produces stable and fast gait in each of the five two-dimensional (2D) modular robotic configurations. Resulting fastest sinusoidal controller for each configuration are evaluated 10 times, with each evaluation lasting for a period of . Mean and standard deviation (SD) of these evaluations are as presented in Table 3.11. This controller is distributed

[5] In theory there exists a phase-difference of between modules and . But since these two modules are connected to each other as a mirror image of each other, when $_1$ $_2$, the two *Spine* modules swing to the same side, forming an arc (Figure 3.10a). On the contrary, when $_1$ $_2$, the two *Spine* modules swing to opposite side, forming an 'S' shape (Figure 3.10b).



(a) $\theta_{Spine_1} = 45\,°, \theta_{Spine_2} = -45\,°$ **(b)** $\theta_{Spine_1} = \theta_{Spine_2} = 45\,°$

Figure 3.10: *Spine modules in the Lizard configuration.*

but not homogeneous, as control parameters differ between modules, and is dependent on the position of the module in the configuration. The controller is scalable, since adding modules to a configuration would not require a change in controller of the preexisting modules. This control-model is simple and intuitive, and good for studying different possible gaits in a given configuration by hand-tuning control parameters. Through parameter optimization it is possible to evaluate the fastest possible gait in a given configuration.

As the number of modules in the configuration grow, the total number of parameters that needs to be tuned for a robot to function, increases linearly as well. During parameter-optimization process, it takes significantly longer to find any meaningful gait in the six-module *Lizard* configuration, compared to the two-module *Minibot* configurations. In both the gaits that emerged in the *Lizard* configuration, the robot does not utilize all four limb-modules. Intuitively, an optimal gait would utilize all available resources, but in the case of the gaits that emerge in *Lizard* configuration, it seems that the EA fails to find the global-optimal solution, due to the shear depth of the parameter space (19 dimensions deep) it is searching for a solution in. Another drawback of this controller is its inability to adapt to change in environment or configuration.

Robot		Speed (cm/s)	
		Mean	*SD*
Minibot		3.41	0.11
Tripod		3.10	0.41
Quadropod		4.22	0.08
Y-bot		6.23	0.28
Lizard	Forward gait	4.57	0.08
	Lateral gait	5.28	0.06

Table 3.11: *Mean and SD of locomotion speed of the best evolved Sinusoidal controller for all five configurations.*

3.2.2 Fourier controller

Sinusoidal oscillators as controller for Modular Robot (MR)s work well for creeping and crawling gaits in one-dimensional (1D) and two-dimensional (2D) modular robotic configurations, but are simple in terms of the control signal generated. For locomotion in three-dimensional (3D) modular robotic configurations with multiple Degree of Freedom (DOF) per limb, or for bipedal gaits, control signal needed for generating locomotion is more complex in nature. So, we look at Fourier series as a way of generating complex control signals for locomotion.

Fourier Series is a way of representing any periodic function with a fundamental period (i.e, with a fundamental frequency of —), as an infinite sum of sine and cosine functions, each with a frequency that is an integer multiple of the fundamental frequency —. The general form for representing any function as a Fourier Series is provided in Equation 3.2.

——— ———

——— ——— (3.2)

where is the fundamental period of the function that is being synthesized. and are the coefficients of the Fourier series, which determine the relative weights of the cosine and the sine components respectively. The first term represents the offset (non-zero-center amplitude) of the periodic function.

Using just the first three terms (, and), i.e, with the first frequency component, a sine function with any offset and phase-shift can be generated. So, to validate this control model as a locomotion controller, *Minibot* configuration is evaluated where each module is controlled by a Fourier controller Equation 3.3, and with parameters as provided in Table 3.12.

— —

——————————— (3.3)

where is the amplitude of the module, is the normalization term, which bounds the control signal to the range of , before amplifying the signal. is determined based on the crest and trough of the generated Fourier signal, as follows:

if

otherwise

With the above stated Fourier controller, modules in the *Minibot* configuration oscillate with a phase-difference of between them, and in turn produce a caterpillar gait, at a speed of . Similarly, evaluating the Fourier controller on modules in the *Y-bot* configuration,

Parameter	Module	
	Head	*Tail*

Table 3.12: *Hand tuned Fourier controller parameters for Minibot configuration.*

with parameters provided in table Table 3.13, resulted in a caterpillar like gait with a locomotion speed of . Modules and oscillate in phase, while there exists a phase-difference of between module and , as well as a phase-difference of between and modules.

Parameter	Module			

Table 3.13: *Hand tuned Fourier controller parameters for Y-bot configuration.*

The Fourier controller in Equation 3.3 is based only on the first frequency component, and so essentially a Sinusoidal controller. A general Fourier controller with frequency components is as shown in Equation 3.4.

—— —— (3.4)

where is the frequency domain size.

Unlike in Sinusoidal controller and Fourier controller with one frequency component, hand tuning parameters (Fourier coefficients) for Fourier controllers beyond the first frequency component is not trivial. Moreover, for a complex configuration like *Quadropod* and *Lizard*, conceptualizing a set of complex trajectories (non-sinusoidal) that would result in stable gait, is not straightforward either. So, we evolved gaits for different modular robotic configurations, using Evolutionary Algorithm (EA), with two variants of Fourier controller: first frequency component Fourier controller (*ffc*-Fourier controller) — with one frequency component — and two frequency component Fourier controller (*tfc*-Fourier controller) — with first two frequency components. Evaluation results of the evolved gaits are provided in the following section.

3.2.2.i Evaluation

Minibot

When evaluated with the best evolved *ffc*-Fourier controller, parameters of which are as presented in Table 3.14, modules in the *Minibot* configuration oscillate with a phase-difference of between them. Reference trajectories for the two modules, generated by this controller, are as shown in Figure 3.11. *Head* and *Tail* modules oscillate in the range of and respectively. Average locomotion speed in this gait is . Compared to the gait emerged with the Sinusoidal controller, in this gait, modules oscillate at almost higher frequency. With the Sinusoidal controller, *Tail* module oscillates between a larger range, and the phase-difference between the two modules is higher as well.

Parameter	Module	
	Head	*Tail*

Table 3.14: *ffc-Fourier controller parameters of the best performing individual for Minibot configuration, optimized through EA.*

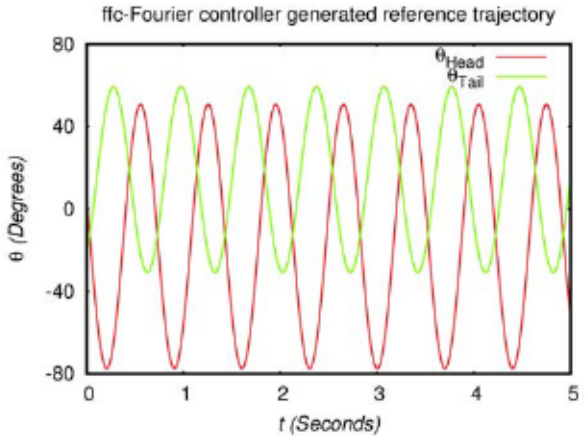


Figure 3.11: *Reference trajectories generated by the best evolved ffc-Fourier controller for Minibot configuration.*

Similarly *tfc*-Fourier controller parameters are optimized for the *Minibot* configuration through EA, and the parameters of the best evolved control is as presented in Table 3.15. Reference

trajectories for the two modules generated by this controller is as shown in Figure 3.12. In this gait, modules *Head* and *Tail* oscillate in the range of and respectively. The emerged gait seems visually similar to a Caterpillar gait, but here both the modules rise and dip twice per cycle of oscillation, which has a period of . Average locomotion speed in this gait is , which is not significantly different from gaits with best evolved Sinusoidal controller and *ffc*-Fourier controller.

Parameter	Module	
	Head	*Tail*

Table 3.15: *tfc-Fourier controller parameters of the best performing individual for Minibot configuration, optimized through EA.*

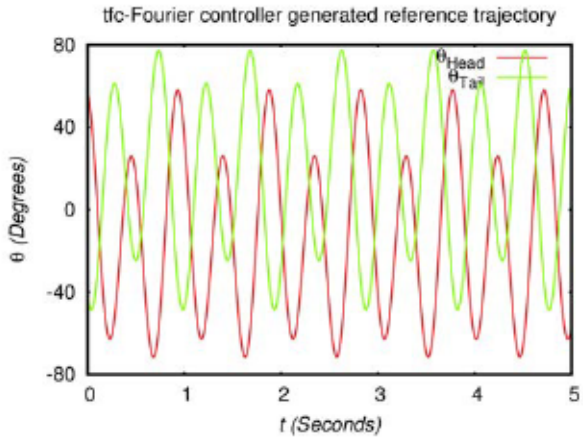


Figure 3.12: *Reference trajectories generated by the best evolved tfc-Fourier controller for Minibot modules.*

Y-bot

Modules in the *Y-bot* configuration, with the best evolved *ffc*-Fourier controller (Table 3.16), oscillate in the following range,

:

:

:

:

Parameter	Module			

Table 3.16: *ffc-Fourier controller parameters of the best performing individual for Y-bot configuration, optimized through EA.*

Modules and oscillated with a phase-difference of , while there exists a phase-difference of between modules and the , and a phase-difference of between the module and the module. But module barely oscillates, contributing minimally for the emerged gait. Average locomotion speed of this gait is , which is sub-optimal. Total number of parameters in this controller, that needs to be optimized for the *Y-bot* configuration is 17, in contrast to 13 parameters in the case of the Sinusoidal controller. Since all the EA parameters between the two epochs (one for Sinusoidal controller and the other for the *ffc*-Fourier controller) are virtually identical, due to the increased number in search dimension, EA fails to find the optimal — or close to optimal — gait in this case.

So, for optimizing *tfc*-Fourier controller parameters for the *Y-bot* configuration (25 parameters), EA is altered wherein, instead of randomly initializing population of the first generation, only those candidates whose fitness is above a set threshold [6] are selected to be part of the first generation. Rest of the EA parameters are kept the same. When evaluated with the best evolved controller (Table 3.17), modules in this configuration oscillated in the following range, and reference trajectories generated by this controller is as shown in Figure 3.13

:

:

:

:

[6] The fitness threshold is set to locomotion speed of .

Parameter	Module			

Table 3.17: *tfc-Fourier controller parameters of the best performing individual for Y-bot configuration, optimized through EA.*

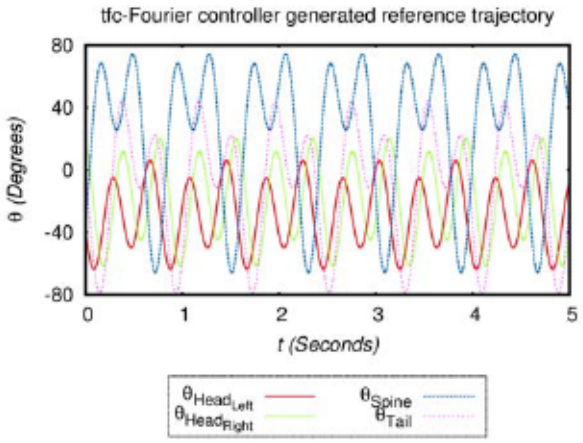


Figure 3.13: *Reference trajectories generated by the best evolved tfc-Fourier controller for Y-bot modules.*

In this gait, modules have a smaller oscillation range compared to the and the modules. Modules and oscillate with a small phase-difference between them, while there exists a bigger phase-difference between the modules and the module, and between the module and the module. All the modules rise and dip twice per cycle of oscillation, with actions of the and the modules contributing the most to the emerged gait. Average speed of locomotion in this gait is slightly better than the speed attained with the best evolved *ffc*-Fourier controller, at , but the gait is still suboptimal compared to the Sinusoidal controller gait for *Y-bot* configuration.

Lizard

For the *Lizard* configuration, parameters of the *tfc*-Fourier controller are optimized through EA, wherein the first generation is populated with randomly selected individuals whose fitness is

above a set threshold of . This is because of the sheer depth of the parameter space, which is 37 dimensions deep. When evaluated with the best evolved *tfc*-Fourier controller (Table 3.18), modules in the *Lizard* configuration oscillate in the following range, and reference trajectories generated by this controller is as shown in Figure 3.14.

:

:

:

:

:

:

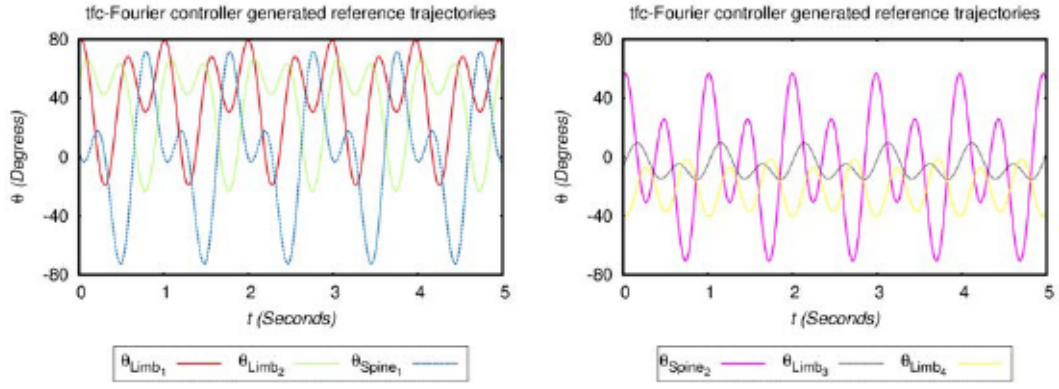
Parameter	Module					

Table 3.18: *tfc-Fourier controller parameters of the best performing individual for Lizard configuration, optimized through EA.*

In this gait, modules and oscillate with a phase-difference of between them, and as a result of these two modules, the robot is propelled in the forward direction. Modules and oscillate with a large phase-difference as well, complementing the forward propulsion, while modules and oscillate with a very small range, barely making contact with the ground surface. Average locomotion speed of the emerged gait is , and so it is suboptimal compared to the Sinusoidal controller gait that emerged in this configuration.

3.2.2.ii Discussion

Fourier series based locomotion controller for MR, provides an alternative to Sinusoidal controller. Similar to Sinusoidal controller, Fourier controllers are distributed, scalable and heterogeneous. It is possible to generate complex trajectories with Fourier controllers beyond the first frequency



(a) *Modules , and* **(b)** *Modules , and*

Figure 3.14: *Reference trajectories generated by the best evolved tfc-Fourier controller for Lizard modules.*

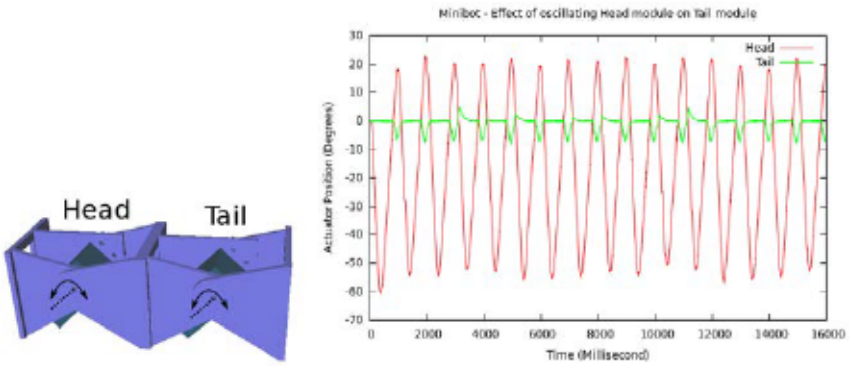
component. One main disadvantage of Fourier controllers is the increasing parameter space, as the frequency domain size grows. For creeping/crawling locomotion in 2D modular robotic configurations, Fourier controllers do not seem to provide any advantage over Sinusoidal controller.

3.3 Morphology dependent locomotion controller

3.3.1 Influence of morphology

Since Modular Robot (MR)s are physically connected multi-robot systems, modules exert force on one another when actuated. In a simulated *Minibot* configuration, when one module (*Head*) is actuated with a sinusoidal oscillator, with an amplitude of , and the other module (*Tail*) is actuated as well, but made to remain at a constant reference position of , the effects of the oscillating *Head* module is observable on the fixed-position *Tail* module. As could be seen in Figure 3.15b, the *Tail* module oscillates as well, but with a low amplitude and an offset, due to the force exerted on it by the oscillating *Head* module. We have termed this phenomenon Intra-Configuration Force (ICF), and it can be quantified by measuring the mean and standard deviation (SD) of the actuator value of the affected (*Tail*) module, which is as shown in Table 3.19. Similarly, when roles of the *Head* and *Tail* modules are interchanged, effects of the oscillating *Tail* module can be observed on the fixed-position *Head* module, but with almost twice the mean and SD, which suggests that effect of the oscillating *Tail* module on the *Head* module is twice compared to that of the oscillating *Head* module over the *Tail* module. This is because of the asymmetric mass distribution of the configuration, which is based on the way in which the two

modules are connected to each other [7].



(a) *Minibot configuration*

(b) *Plot of actuator values in the Minibot configuration, demonstrating effects of the oscillating Head module, over the fixed-position Tail module.*

Figure 3.15: *Minibot configuration and a plot of its actuator values.*

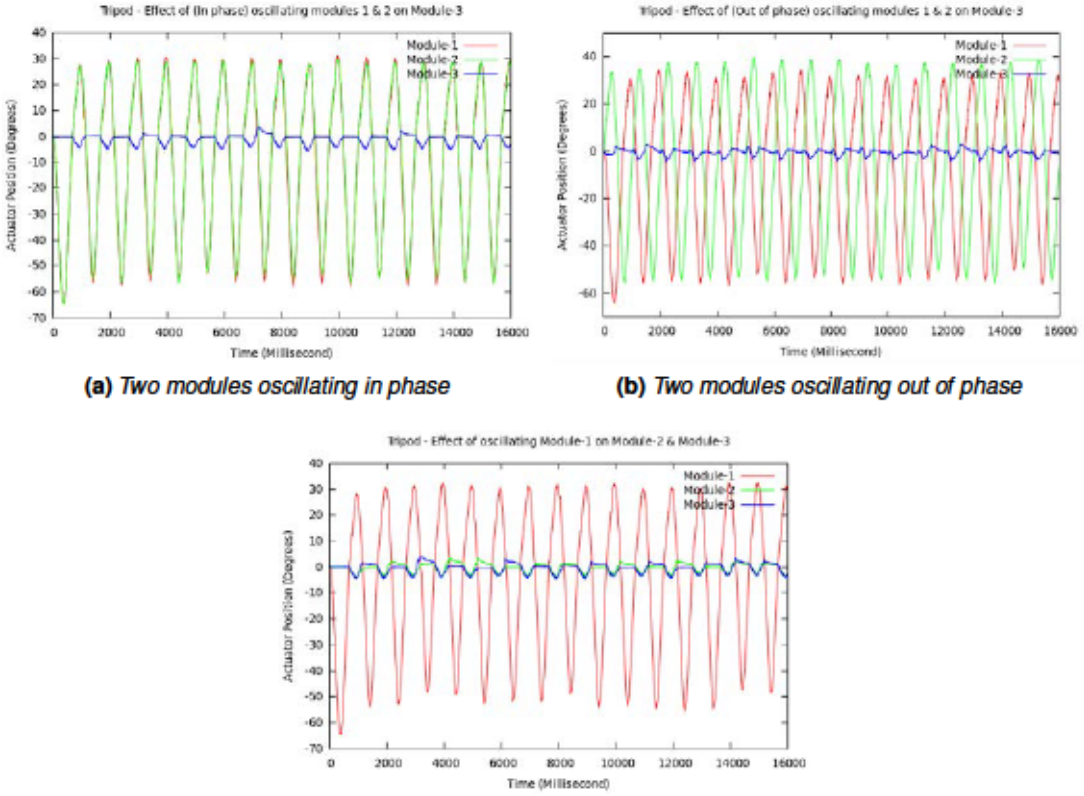
		Fixed			
		Head		Tail	
		Mean	SD	Mean	SD
Oscillating	Head	-	-		
	Tail			-	-

Table 3.19: *Quantifying ICF by calculating mean and SD of actuator position of the fixed-position module, connected to an oscillating module in the Minibot configuration.*

Similar experiments are conducted on a simulated *Tripod* configuration, by actuating two modules with sinusoidal oscillators, which oscillate in phase in the first experiment, and with a phase-difference of in the second experiment, while the third module is fixed to a reference position of , in both the cases. In the third experiment, a single module is oscillated, while the other two modules are fixed to a reference position of . In all three cases, although the oscillating module(s) has/have an effect on the fixed-position module(s), the effects are different on the fixed-position modules(s), as could be seen in Figure 3.16. Mean and SD of actuator position of the respective fixed-position modules(s) are as provided in Table 3.20.

As could be observed in Table 3.20, there is a noticeable difference in the force exerted on the fixed-position module , based on the phase-difference between the two oscillating modules in the *Tripod* configuration. To further examine this relationship between varying phase-difference among oscillating modules, and the force exerted on the fixed-position module, modules and are actuated with sinusoidal oscillators, and with phase-difference between the modules ranging between and , at an interval of . Result of this experiment is as shown in

[7] The side of the *Tail* module holding the servo motor is connected to the side of the *Head* module that is free.



(a) *Two modules oscillating in phase*

(b) *Two modules oscillating out of phase*

(c) *One module oscillating*

Figure 3.16: *Plot of actuator values in the Tripod configuration, demonstrating effects of oscillating module(s) over fixed-position module(s).*

Table 3.21, which is also plotted as a bar-graph in Figure 3.17. Force exerted on the fixed-position module is at the highest when the two oscillating modules oscillate in phase, and at the lowest when modules oscillate out of phase. The SD at a phase-difference of is twice compared to SD at a phase-difference of . This is because both the modules, while oscillating in phase, exert force on the fixed-position module at the same time, whereas when oscillating with a phase-difference of , each module exert force on the fixed-position module at slightly different points in time.

Force on the fixed-position module is exerted when an oscillating module pushes down on the ground-surface. So, to study how ground-surface friction determines the force exerted, another experiment, similar to the one explained in the previous paragraph with the *Tripod* configuration, is conducted. In this experiment, modules and oscillate in phase, and the mean and SD of the fixed-position module are sampled over varying Coefficient of Friction (COF) of the ground-surface. Results of this experiment is as shown in Table 3.22, and the same is plotted as a bar-graph in Figure 3.18. The results indicate that there exists a positive correlation between COF of the ground-surface and the force exerted on a module.

		Fixed			
		Mean	SD	Mean	SD
Oscillating	and	-	-		
	and	-	-		

Table 3.20: *Mean and SD of actuator position of fixed-position modules in the Tripod configuration.*

		Fixed-position module	
		Mean	SD
Oscillating modules and			

Table 3.21: *Mean and SD of actuator values of the fixed-position module in the Tripod configuration, sampled over different phase value of oscillating modules.*

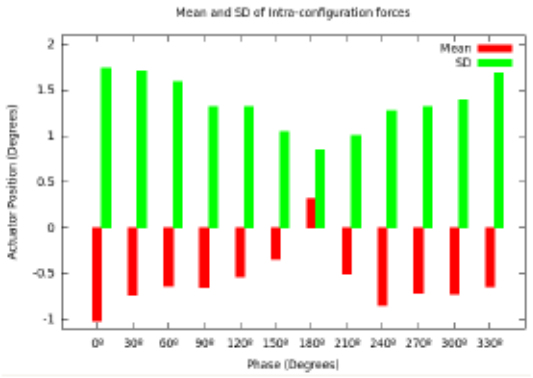


Figure 3.17: *Plot of mean and SD of actuator values of the fixed-position module in the Tripod configuration, sampled over different phase values of oscillating modules.*

Similar experiments are conducted on a real *Y-bot* configuration, wherein all the modules in the configuration, except the module, are actuated with sinusoidal oscillators with an amplitude of and in phase, while the module is actuated as well, but made to

		Fixed-position module	
	COF	Mean	SD
Oscillating modules and			

Table 3.22: *Mean and SD of actuator values of the fixed-position module in the Tripod configuration, sampled over varying COF of the ground-surface.*

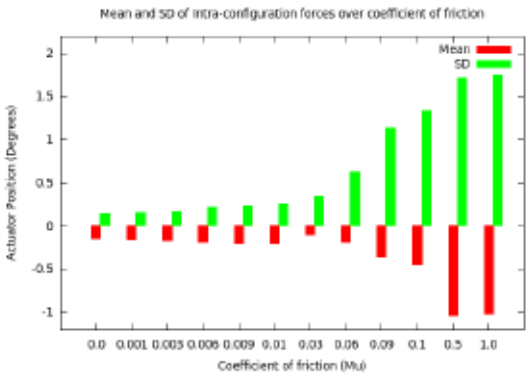
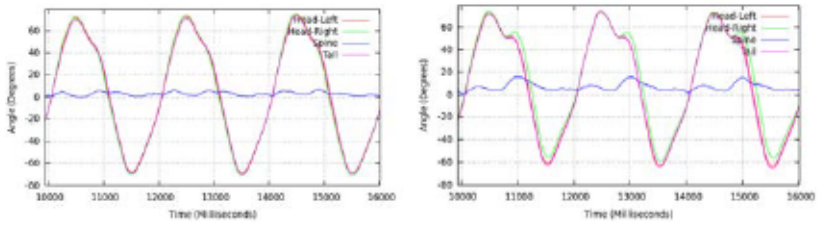


Figure 3.18: *Plot of mean and SD of actuator values of the fixed-position module in the Tripod configuration, sampled over varying COF of the ground-surface.*

remain at a constant reference position of . Experiments are conducted on both low-friction and high-friction surfaces — on polished stone floor tiles, and on grip tape of a skateboard, respectively. In the low-friction surface experiment, the module oscillate with a mean and SD of and respectively (Figure 3.19a), while in the high-friction experiment, the module oscillate with a mean and SD of and respectively (Figure 3.19b). It is to be noted that in experiments with the real *Y-bot* configuration, the fixed-position module peaks in the positive-axis, unlike the fixed-position modules in the previous experiments. This is due to the morphology of the robot. When all the oscillating modules push down on the ground surface (i.e,) simultaneously, this results in the hinges of the fixed-position module pushed down towards the ground surface as well, and hence the spike in the positive direction.

Based on several factors such as morphology and mass-distribution of a configuration,



(a) *Low-friction surface experiment* **(b)** *High-friction surface experiment*

Figure 3.19: *Plot of actuator values in the Y-bot configuration, demonstrating effects of oscillating modules over the fixed-position Spine module.*

phase-relation between oscillating modules and COF of the ground-surface, connected modules in a given modular robotic configuration, exert force of each other. This could be seen as implicit-analog-communication between modules, as a result of the embodiment of the robot, and could be used for controlling modules distributively, such that difference in local behavior of individual modules result in the emerged global behavior of the robotic configuration.

In the rest of this sections, two different morphology based locomotion controllers for MRs, that is developed as part of this thesis, which rely on such implicit-analog-communication between modules in a configuration, is presented. Controllers parameters of both the controllers are optimized through Evolutionary Algorithm (EA), and evaluation results of different modular robotic configurations are presented.

3.3.2 Neural-oscillator controller

Based on the experimental results from section subsection 3.3.1, page 66, a neural-oscillator based distributed locomotion controller is developed. In this control scheme, each module would have its own controller, the controllers are uncoupled and have the exact same parameter values across the configuration. Control flow of the proposed controller is as shown in Figure 3.20.

The Artificial Neural Network (ANN) of the controller has one input neuron, one hidden layer with a single hidden neuron, and one output neuron. The input to the ANN is the position of the module's actuator, whereas the output of the ANN is the control signal to the same. The lone hidden neuron and the output neuron have one bias node each, and hyperbolic tangent activation function is used in all the layers.

Rate of rotation of an oscillating module's actuator is dynamically influenced as a result of Intra-Configuration Force (ICF) that exist among connected oscillating modules, and this phenomenon is captured by Equation 3.5. The controller sources the position of the actuator at every time step, and if the rotation speed of the module's actuator is below a certain threshold, then the current actuator position of the module is fed into the ANN to produce the next control signal.

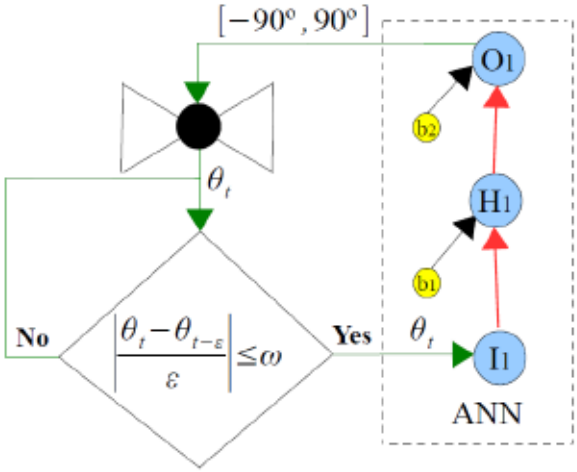


Figure 3.20: *Control flow of the Neural-oscillator controller.*

(3.5)

where is the position of the module's actuator at time . Parameters and are constants, with corresponding to the actuator's rotation speed threshold.

Unlike in Sinusoidal controller and Fourier controller, where a continuous reference trajectory is generated, in this controller, only two reference signals are generated per cycle of oscillation. Imagine, if control signal to the module's actuator (i.e, output of the ANN) at instance is , then the internal Proportional–Integral–Derivative (PID) controller of the actuator drives the actuator to reach the goal position . Either on or before reaching the goal position , when Equation 3.5 is satisfied, the next control signal (i.e,) is generated by the ANN part of the Neural-oscillator controller, and fed to the module's actuator, followed by control signal at the next instance. Synaptic weights of the ANN are optimized through Evolutionary Algorithm (EA), so as to produce such oscillatory output.

This controller is distributed, as each module has its own controller, and do not rely on a central controller for synchronization. All the controllers in all the modules of a configuration would have identical parameter values, making the controller homogeneous. Each controller is completely independent, as controllers in a configuration are not coupled with each other, and so the controller is scalable as well. Modules do not communicate with each other explicitly either. So, difference in action of modules in a configuration, comes about as a result of interaction between modules and their environment, in the form of ICF, which is captured by Equation 3.5.

3.3.2.i Evaluation

All the parameters of the Neural-oscillator controller are optimized through EA. For evolving a gait, the target modular robotic configuration is setup in the simulation environment, with randomly initialized controller parameter values, and with all the modules having their own copy of the cloned controller. Each candidate controller is evaluated in simulation for a period of , and based on the fitness value of the controllers, they are selected, crossed-over, mutated and carried forward to the next generation. This is done independently for all five configurations. Post evolution, best performing candidate controller of the last generation is evaluated for a period of . Evaluation results of all five configurations are presented in the following subsections.

Minibot

Applying the best evolved Neural-oscillator controller for *Minibot* configuration results in the typical caterpillar like gait. Since both the modules have identical controllers, start at the same time, and with the same initial conditions, modules start to oscillate in phase, but quickly develop and maintain a steady phase-difference. Average phase-difference between the two modules is , with a standard deviation (SD) of . Frequency of oscillation is not predefined in this control scheme, but is intrinsic to the system. Average frequency of oscillation of the *Head* and *Tail* modules are and respectively, with a SD of and respectively. Similarly, phase-difference between modules is not predefined either, but is a result of the morphology of the robot, and due to the interaction of the robot with its environment. Plots of actuator values, frequency and phase-difference between modules in the emerged gait is as shown in Figure 3.21, Figure 3.22 and Figure 3.23 respectively. Average locomotion speed in this gait is .

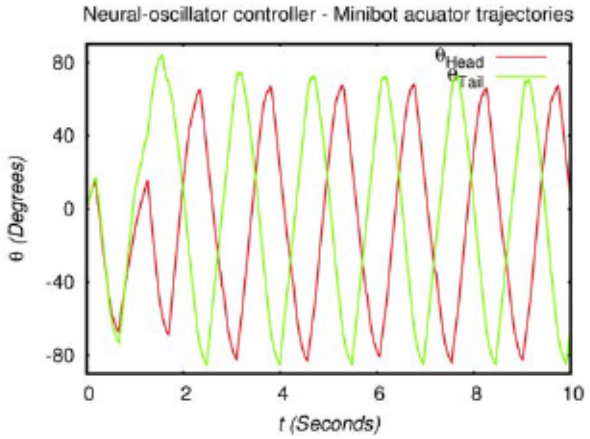


Figure 3.21: *Actuator trajectories of Minibot modules, when evaluated with the best evolved Neural-oscillator controller.*

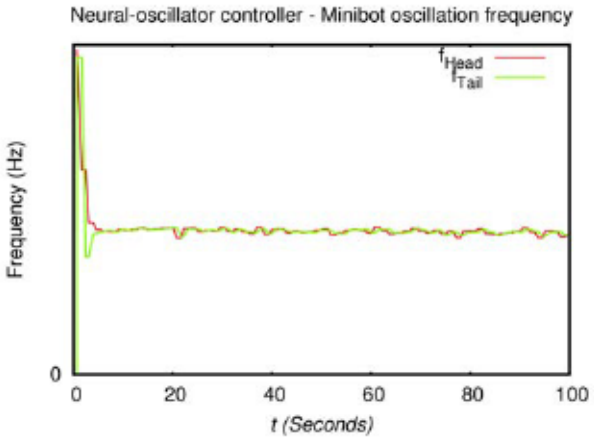


Figure 3.22: *Oscillation frequency of Minibot modules, when evaluated with the best evolved Neural-oscillator controller.*

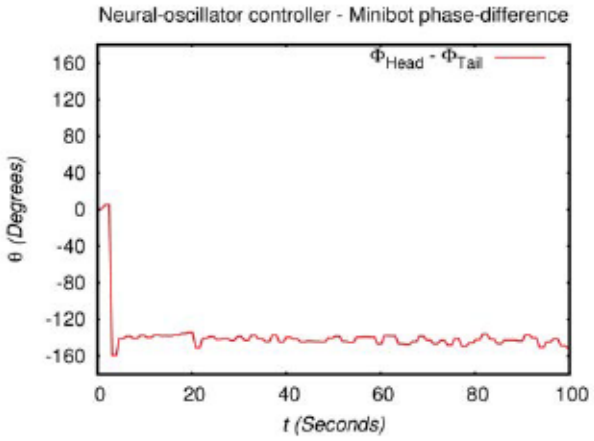


Figure 3.23: *Phase-difference between Minibot modules, when evaluated with the best evolved Neural-oscillator controller.*

Tripod

Applying the best evolved Neural-oscillator controller for the *Tripod* configuration results in a stable gait, where the robot travels on an arced trajectory. Here, stability of a gait is measured in terms of consistency of phase-difference between modules. In the *Minibot* configuration, modules *Head* and *Tail* oscillate with a phase-difference of between them, but with a low SD of in the emerged phase-difference. If, on the other hand, the phase-difference is erratic or oscillated over time, this would result in the robot move back and forth on a straight line, rather than travel consistently in one direction. In the *Tripod* configuration, when evaluated

with the best evolved Neural-oscillator controller, modules oscillate with fairly consistent phase-difference between each other, with SD in phase-difference ranging between .
A plot of the phase-difference between modules, during one of the evaluations with the best evolved Neural-oscillator controller, is as shown in Figure 3.24]. Table 3.23 contains mean and SD of phase-difference between all the module pairs. Average locomotion speed in this gait is .

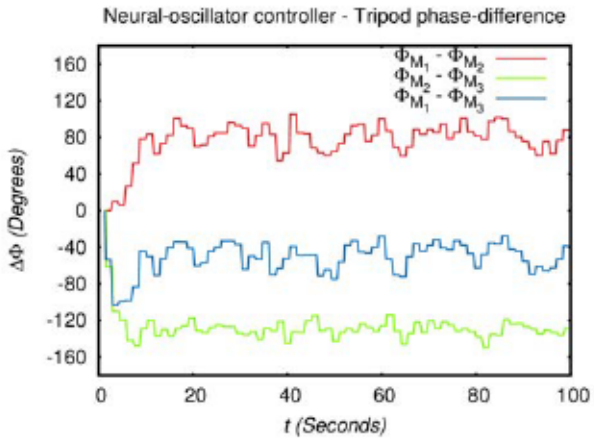


Figure 3.24: *Phase-difference between Tripod modules, when evaluated with the best evolved Neural-oscillator controller.*

		Tripod Modules		
	Mean			
	SD			
	Mean			
	SD			
	Mean			
	SD			

Table 3.23: *Mean and SD of phase-difference between module pairs in the Tripod configuration, when evaluated with the best evolved Neural-oscillator controller.*

Quadropod

In the *Quadropod* configuration, best evolved Neural-oscillator controller results in a diagonal crawling gait. The controller is evaluated ten times, and in each evaluation, a pair of adjutant modules oscillate in phase[footnote 3], while there existing phase-difference between the rest of the module pairs. This results in the diagonal crawling gait, similar to the gait that emerges in this configuration with the best evolved Sinusoidal controller (section 3.2.1.i, page 52). Unlike

the periodic function controllers, output of this controller is not deterministic, but stochastic as it is based on the module's interaction with other connected modules and the environment. Due to the symmetry of the configuration, phase-relation that emerges between modules is not consistent between evaluations, although stable (low SD of phase-difference) within an evaluation. This results in the robot crawling in a different direction (north-east, north-west, south-west, south-east) each evaluation. Out of the ten evaluations, the robot crawled in north-east and north-west directions thrice each, and twice each in south-east and south-west directions. A plot of phase-difference values between modules during one of the evaluations is as shown in Figure 3.25, and Table 3.24 contains mean and SD of phase-difference between all module pairs from the same evaluation. Average locomotion speed in this controller is .

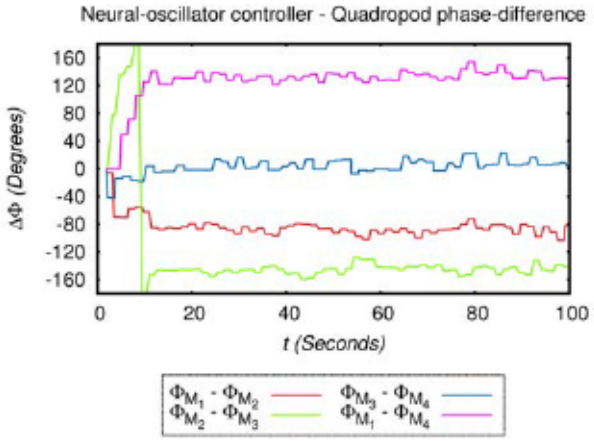


Figure 3.25: *Phase-difference between Quadropod modules, when evaluated with the best evolved Neural-oscillator controller.*

		Quadropod Modules			
	Mean				
	SD				
	Mean				
	SD				
	Mean				
	SD				
	Mean				
	SD				

Table 3.24: *Mean and SD of phase-difference between module pairs in the Quadropod configuration, when evaluated with the best evolved Neural-oscillator controller.*

Y-bot

In the *Y-bot* configuration, when applied with its best evolved Neural-oscillator controller, the emerged gait is very similar to the gait that emerge in this configuration with the best evolved Sinusoidal controller (section 3.2.1.i, page 54). Based on the consistency of the phase-relation among modules, the emerged gait is again very stable. The two *Head* modules oscillate in phase[footnote 3], and there is a steady phase-difference between the rest of the modules in the configuration, which produces a propagating sine-wave starting from the *Head* modules and moving in the direction of the *Tail* module, resulting in the forward propulsion of the robot, in the direction of the *Tail* module. phase-relation table and a plot of phase-difference between modules, when evaluated with the best evolved controller, can be seen in Figure 3.26 and Table 3.25 respectively. Average speed of locomotion achieved in this controller is .

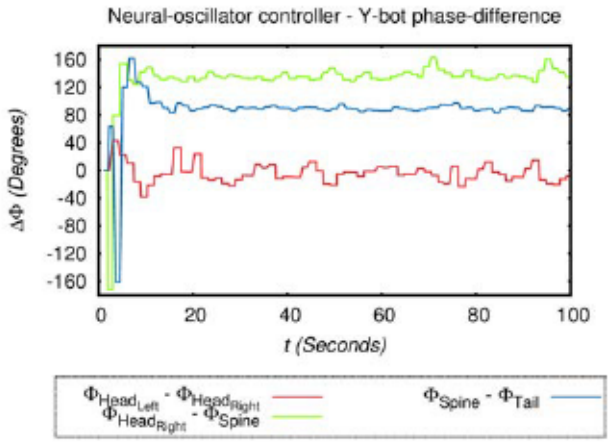


Figure 3.26: *Phase-difference between Y-bot modules, when evaluated with the best evolved Neural-oscillator controller.*

		Y-bot Modules			
	Mean				
	SD				
	Mean				
	SD				
	Mean				
	SD				
	Mean				
	SD				

Table 3.25: *Mean and SD of phase-difference between module pairs in the Y-bot configuration, when evaluated with the best evolved Neural-oscillator controller.*

Lizard

The *Lizard* configuration produces a reptilian like quadruped forward-walking gait, when evaluated with the best evolved Neural-oscillator controller for this configuration. Modules start to oscillate in phase, but very quickly converge to and maintain a steady phase-difference. In the emerged gait, pairs of leg modules on the same side (and) oscillate in phase[footnote 3], while there exists a phase-difference of between the two pairs. The two modules oscillate in the yaw-axis with a phase-difference of between them. Considering module as the reference module, a phase-difference of exists between and modules, and phase-difference of between and modules. The robot would move in the opposite direction if these two phase-difference values are interchanged. Average speed of locomotion achieved in this controller is .

3.3.2.ii Cross evaluation

Neural-oscillator controller is distributer, homogeneous and scalable. Therefore, controller optimized for one configuration can be applied to any other configuration, of any size. So, to test how adaptable Neural-oscillator controller is, best evolved controller of each of the five configurations are cross-evaluated on rest of the four configurations. Cross-evaluation results, as speed of locomotion ( ) in each evaluation, are provided in Table 3.26.

			Controller					Average
			Minibot	*Tripod*	*Quadropod*	*Y-bot*	*Lizard*	
Robot	*Minibot*	Mean	3.12	2.60	2.84	N/A[footnote 8]	2.73	2.26
		SD	0.04	0.18	0.04		0.06	
	Tripod	Mean	U/P[footnote 10]	1.48	1.31	1.15	1.26	1.04
		SD		0.17	0.28	0.56	0.31	
	Quadropod	Mean	2.10	1.45	3.66	1.61	2.68	2.30
		SD	0.77	0.22	0.42	0.48	0.46	
	Y-bot	Mean	3.0	1.76	3.44	5.31	5.08	3.72
		SD	0.57	0.42	0.47	0.89	0.37	
	Lizard	Mean	U/P[footnote 10]	4.52	U/P[footnote 10]	U/P[footnote 10]	5.92	2.1
		SD		0.57			0.32	
Average			1.64	2.36	2.25	1.61	3.53	

Table 3.26: *Cross-evaluation results of best evolved Neural-oscillator controllers evaluated on all five configuration.*

Each controller is evaluated on all five configurations, and the values shown in Table 3.26 are the mean speed of locomotion ( ), and SD, after 100 seconds of evaluation, averaged over 10 trials. Each row consists results of one configuration evaluated with all five controllers. If the robot tipped over at any point during evaluation, on 4 or more out of 10 evaluations, then the result

is not considered and marked as N/A[8]. Stability of a gait is based on consistency of the phase-difference among modules in a given configuration. Inconsistent or periodic phase-difference results in the robot moving erratically or back-and-forth respectively. So, if phase-difference in the emerged gait during cross-evaluation is erratic or unstable[9] in 4 or more out of 10 trails, then that evaluation is not considered either, and marked as U/P[10].

Best performance in each configuration is obtained with the controller optimized for that configuration, which are the diagonal elements of Table 3.26 (cells colored gray). The last row contains average locomotion speed of each of the five controllers, evaluated on all five configurations, while the last column contains average locomotion speed of each of the five configuration, evaluated with all five controllers. Controllers optimized for *Tripod* and *Lizard* configurations are the most adaptive, as they produce some locomotion in the rest of the four configurations, albeit suboptimal, with the *Lizard* controller having the highest average locomotion speed. Among the robots, *Quadropod* and *Y-bot* are the most adaptive configurations, as they results in stable gaits when evaluated with the rest of the four controllers. *Y-bot* configuration averaged the highest mean locomotion speed at .

3.3.2.iii Dicussion

The Neural-oscillator controller is distributer and homogeneous. Although all the modules in a configuration have the exact same controller and parameters, start with the same initial conditions, and do not even communicate with each other explicitly, they result in producing different local actions — in the form of phase-different between modules — which results in the emerged global action in the form of stable gait. The reason for this behavior is the ICF that exists between modules, which is due to the embodiment of the robot.

When cross-evaluated, 15 out of 20 cross-evaluations results in stable, albeit suboptimal, gaits. So, Neural-oscillator controller is not only scalable, but can also adapt to the change in configuration. For example, when a controller optimized for *Lizard* configuration is cross-evaluated on the *Y-bot* configuration, a caterpillar-like crawling gait, which is typical for the *Y-bot* configuration, emerges. The same controller, on the *Lizard* configuration, results in a reptilian-like quadruped forward-walking gait. Two fundamentally different gaits emerge from a single controller on two different configurations. This is due to the morphology of the configuration in which a gait emerges, rather than due to just the controller alone.

[8]Not Available

[9]Consistency of phase-difference is measured by calculating SD of phase-difference between module pairs in the configuration. In the context of this thesis, phase-difference is considered consistent if

.

[10]Unstable Phase-difference

3.3.3 Inverse sinusoidal controller

The morphology dependent Neural-oscillator controller is homogeneous, uncoupled and can adapt to the morphology of the robot. The Artificial Neural Network (ANN) part of the Neural-oscillator controller, takes current actuator position of a module as input, and outputs the next actuator position that the module needs to oscillate to. On the contrary, periodic class of controllers (Sinusoidal controller and Fourier controller) generate precise trajectory for the module to follow. But these controllers, unlike the morphology dependent Neural-oscillator controller, are module specific (heterogeneous) and so are not adaptive. To combine features of both periodic and morphology dependent controllers, Inverse sinusoidal controller is developed.

Control flow of Inverse sinusoidal controller is as shown in Figure 3.27. Similar to the Neural-oscillator controller, this controller is distributer, homogeneous, uncoupled and morphology dependent as well. At its core, a simple sinusoidal function generates trajectory for the module's actuator to follow. One major distinction between this controller and the Sinusoidal controller (subsection 3.2.1, page 49) is, here for all the modules in the configuration. So, change in a module's behavior comes about as a result of the morphology of the robotic configuration. The controller sources actuator position at ever time step, and compares it against the last control position generated by Equation 3.6. When this difference is greater than , variable gets readjusted (Equation 3.7), such that , at the next time step. So, difference in a module's action comes about as a result of its interaction with other modules in the configuration, and with the environment. At the beginning, all the modules in a configuration start off by oscillating in phase, but difference in their oscillation phase emerges as a result of readjusting variable .

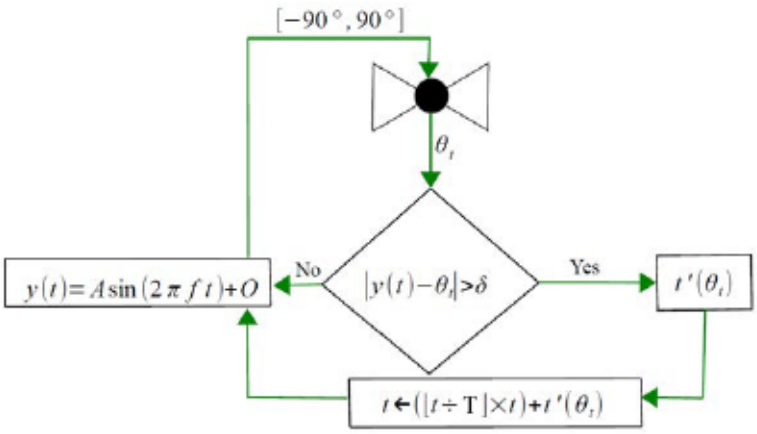


Figure 3.27: *Control flow of the Inverse sinusoidal controller.*

(3.6)

(3.7)

(3.8)

(3.9)

Equation 3.9 is the inverse sinusoidal function, which is developed as past of this thesis. For a sinusoidal function, which is a function of time , with amplitude , offset and frequency , given the current angle and velocity and velocity respectively, inverse sinusoidal function can calculate time (where is the period of the oscillation). That is, it can calculate the point within the current oscillation cycle at which . This function calculates based on the current angle and the sign of , i.e, the direction of the current velocity.

Given by Equation 3.9, in Equation 3.8 , as well as the direction of velocity is switched on the following time step . Based on , is updated accordingly in Equation 3.7.

The intuition behind this control model is: due to homogeneity of the controller, all the module in a configuration start by oscillating in phase. But as a result of Intra-Configuration Force (ICF) between connected modules, varying difference between control signal and actuator position emerge in modules, based on the position of the module in a configuration. When , gets adjusted such that in the following time-step, while also switching the direction of oscillation. This results in implicitly breaking phase symmetry among oscillating modules. Although, a meaningful gait for a given configuration only emerges when parameter is optimized, and an optimal gait by further optimizing parameters , and .

3.3.3.i Evaluation

A gait in each of the five modular robotic configurations is evolved by following an approach similar to that of the Neural-oscillator controller (subsubsection 3.3.2.i, page 73). In this case, only four parameters (, , and) are optimized per gait/configuration. Post evolution, best performing candidate controller of the last generation is evaluated for a period of . Evaluation results of all five configurations are presented in the following subsections.

Minibot

Reference and actuator trajectories of the *Minibot* modules, when evaluated with the best evolved Inverse sinusoidal controller, are as shown in Figure 3.28. Control signals for the first cycle are identical for the two *Minibot* modules. At , difference between the control signal and actuator position of the *Head* module grows to a value greater than parameter (in this case), which results in updated by Equation 3.8, breaking oscillation

symmetry between the two modules. Similarly, in the third ($t = 1.88s$), fourth ($t = 2.50s$) and fifth ($t = 3.11s$) oscillation cycles of the *Head* module (marked in red on the x-axis of the plot in Figure 3.28), t_{Head} is further readjusted, resulting in the two modules to oscillate with a phase-difference of $117.17°$ for the rest of the evaluation period ($3.11s < t < 100s$). With this controller, *Minibot* produces the caterpillar gait, which is typical and the only meaning gait for this configuration, traveling at an average speed of $2.72cm/s$.

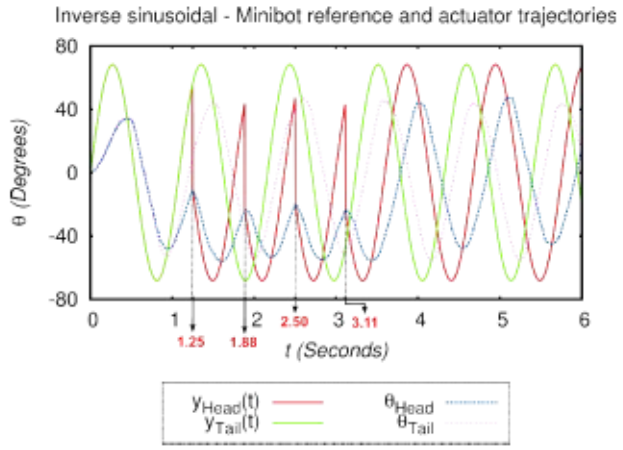


Figure 3.28: *Reference and actuator trajectories of Minibot modules, when evaluated with the best evolved Inverse sinusoidal controller. Marked in red on the x-axis are points in time when control signal of the Head module is adjusted, resulting in the emerged gait.*

Tripod

When evaluated with the best evolved controller, all the modules in the *Tripod* configuration initiate by oscillating in phase, but after several iterations of adjustments of the respective t values of all the three modules, all of which occur during the first 10 seconds of the evaluation (Figure 3.29), a forward crawling gait emerges. After t adjustments, modules M_1 and M_3 settle into oscillating in phase, while module M_2 settles into an oscillation with a phase-difference of $\approx 148.67°$ with respect to the other two modules, resulting in the forward-crawling gait at an average speed of $1.26cm/s$.

Quadropod

When evaluated with the best evolved controller, modules M_1 and M_2 settle into oscillating in phase, while modules M_3 and M_4 settle into oscillations with a phase-difference compared to the other two pairs. All the module in this configuration settle into a steady gait within the first 5 seconds of the evaluation, and the emerged gait is very similar to the gait that emerge in this configuration, when evaluated with the best evolved sinusoidal controller, and at an average

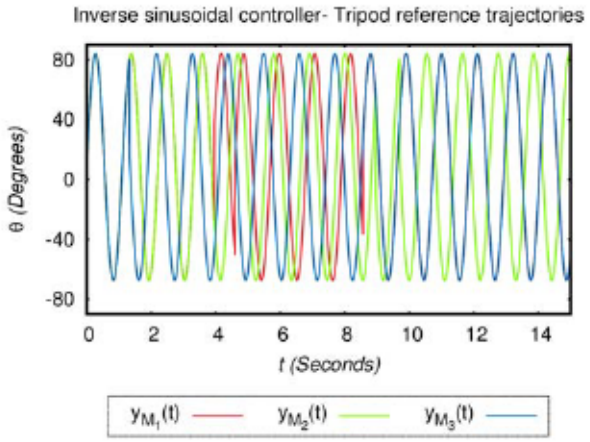


Figure 3.29: *Reference trajectories of Tripod modules, when evaluated with the best evolved Inverse sinusoidal controller.*

locomotion speed of . Phase-relation between modules that emerged in this gait, during one of the evaluations, is as shown in Table 3.27.

	Quadropod Modules			

Table 3.27: *Phase-difference between module pairs in the Quadropod configuration, that emerge when evaluated with the best evolved Inverse sinusoidal controller.*

Y-bot

Best evolved controller for the *Y-bot* configuration produces the caterpillar gait, which is typical for this configuration. Modules and oscillate is phase, while there exists a consistent phase-difference between the modules and the module, as well as between module and the module. Although, the emerged gait is suboptimal at an average locomotion speed of .

Lizard

The *Lizard* configuration produced a lateral-walking gait, when evaluated with the best evolved Inverse sinusoidal controller. Limb module pairs on the same side (modules and

as one pair, and modules and as the second pair) oscillate in phase, while a phase-difference of exist between the two pairs. The two modules oscillate in phase as well, resulting in a lateral-walking gait similar to the gait that emerges in this configuration with the Sinusoidal controller. Average speed of locomotion in this gait is .

3.3.3.ii Dicussion

Similar to the Neural-oscillator controller, Inverse sinusoidal controller is distributed, homogeneous, and scalable. But unlike a Neural-oscillator controller, it provides a continues trajectory for the module's actuator to follow. This controller combines features of Sinusoidal controller and Neural-oscillator controller, and only four parameters needs to be tuned to achieve a gait with this controller. Change in action of oscillating modules come about as a result of ICF that exist among modules in a configuration.

Best evolved controller in each configuration is evaluated 10 times, with each evaluation lasting for a period of 100 seconds. Mean and standard deviation (SD) of these evaluations are as presented in Table 3.28. Gaits that emerge in each configuration with the resulting fastest controller is typical for the respective configuration. In the *Lizard* configuration, a lateral-walking gait emerges, propelling the robot in the north direction [11]. Speed of locomotion in all five gaits are lower compared to both Sinusoidal controller and Neural-oscillator controller. The main reason for this is the common amplitude and offset parameters for all the modules in the configuration.

Robot	Speed ( )	
	Mean	*SD*
Minibot	2.72	0.13
Tripod	1.26	0.23
Quadropod	2.63	0.27
Y-bot	3.75	0.12
Lizard	2.32	0.08

Table 3.28: *Mean and SD of locomotion speed of all the configurations, evaluated with the respective best evolved Inverse sinusoidal controller.*

[11] A similar gait emerges in this configuration, with some of the Sinusoidal controllers evolved (section 3.2.1.i, page 55). But a distinction between the two gaits is the direction of locomotion, which is opposite to each other.

Summary of the chapter

In this chapter five different modular robotic configurations, experimented with during this course of this thesis, are described from a kinematic and gait perspective. Of the five configurations, *Minibot*, *Tripod* and *Quadropod* configurations can be found in the literature, but *Y-bot* and *Lizard* configurations are original to this thesis, to the best of the author's knowledge.

Two class of locomotion controllers for Modular Robot (MR)s are developed and evaluated on several modular robotic configurations. The first class of controllers is periodic function based, which includes: Sinusoidal controller, based on sine wave function, and Fourier controller, based on Fourier series. Sinusoidal oscillators as locomotion controllers for MRs can be found in the literature, and so implemented in this thesis so as to set a benchmark for comparing rest of the controllers developed as part the thesis. Evolutionary Algorithm (EA) is used for optimizing Sinusoidal controller parameters, and then the resulting fastest controller is evaluated for each of the five modular robotic configurations. Two different gaits emerge in the *Lizard* configuration, and one each in the rest of the configurations. Of the total six gaits, four of them are the fastest compared to the rest of the controllers. But a drawback of this controller is that it is heterogeneous, and so cannot adapt to change in configuration. Also, as the number of modules in the configuration grows, dimension of the controller parameter space grows linearly as well, making it hard for an optimization technique to find good set of parameters.

The second under the periodic function based controllers class, is the Fourier controller. This is the first instance of use of Fourier series as locomotion controller in MR, to the best of the author's knowledge. This controller is heterogeneous as well, and so not adaptive to change in configuration. A distinction between this controller and Sinusoidal controller is the increased complexity in the generated trajectory, when anything over the first frequency component is used. Two variants of Fourier controller are evaluated: one with just the first frequency component, and the other with two frequency components. Gaits in some configurations are evolved with these controllers, and evaluated. Best evolved first frequency component Fourier controller (*ffc*-Fourier controller) on the *Minibot* configuration produces the fasted caterpillar gait, while the rest of the evaluations are suboptimal. A major disadvantage of this controller is increased dimension of the parameter space, which is unnecessary for two-dimensional (2D) modular robotic configurations.

The second class of controllers developed in this thesis are morphology based. Two types of morphology based locomotion controllers for MRs are developed, which are: ANN based Neural-oscillator controller, and inverse sinusoidal function based Inverse sinusoidal controller. Implicit forces that exist between connected modules in a configuration, due to embodiment of the robot, are studied and quantified. Based on this, a distributed and homogeneous locomotion controller, consisting an ANN, is developed. Control parameters are optimized through EA and evaluated of all five modular robotic configurations. Although all the modules carry the exact same controller, start with the same initial conditions, and are uncoupled, modules act differently, through which a stable gait emerges as a result of the morphology of the robotic configuration. Best evolved controllers of each configuration are cross-evaluated on each other. Results of the

cross-evaluation prove scalability and adaptability of the controller. Two distinct but stable gaits emerge on two different configurations, when evaluated with the same controller, suggesting a strong link between the morphology of the robot and the controller.

Inverse sinusoidal controller is the second type of morphology based locomotion controller developed in this thesis. A general inverse sinusoidal function is developed as part of this controller, which, given the current position and velocity of a sinusoidal oscillation, gives time (value on the -axis) within the current cycle. This function is used to modulate control signal to module's actuator, through which diversity in module's action comes about, and when the parameters are optimized, a gait emerges. EA is once again used for optimizing control parameter for each configuration independently, and the best evolved controller is evaluated. Stable gaits emerge in all five configurations, although not optimal.

CHAPTER 4

Bipedal Locomotion Controller for Humanoids

Introduction

In the previous chapter, four different locomotion controllers for MRs are developed and evaluated. Trajectories generated in three of these controllers are very simple — with two of them being sinusoidal based — which suffice for creeping/crawling gaits in 2D modular robotic configurations. More complex trajectories are needed for bipedal locomotion in a humanoid robots. Fourier controller presented in the previous chapter could generate such trajectories for bipedal locomotion, but this controller is parameter heavy. A Fourier series based controller for bipedal locomotion has been implemented in [Yang et al., 2006] and [Shafii et al., 2009].

One of the objectives of this thesis is to develop locomotion controller for producing bipedal gait in a humanoid robot, and to do so through a model-free approach. Looking at locomotion as a set of coordinated oscillations, the focus in this chapter is to: (i) develop a parameterized periodic function, that can generate a wide variety of complex trajectories, and be used as locomotion controller (ii) learn a stable gait in a simulated humanoid robot by optimizing controller parameters through EA.

In this chapter, a feature-based, general periodic function is developed, through which a wide variety of linear trajectories can be generated. Shape of the trajectory generated through this function, can be defined as a set of features, such as symmetry, skewness, signal-width, duality and squareness, along with the regular amplitude, offset, phase and frequency parameters. First, joint trajectories of a previously evaluated nonlinear bipedal gait are taken as reference, and a set of linear approximates are modeled. These approximates are then tested on a simulated

humanoid robot, to prove the validity of the control model. After which, linear joint trajectories are optimized afresh through EA, and evaluated. The main object of the controller model presented in this chapter, is a model-free approach towards bipedal locomotion.

4.1 Simplified Linear Model

In [Monje et al., 2013] Monje et al., have successfully tested joint trajectories for bipedal walking gait, generated using the cart-table method, on both a simulated and a real Humanoid for Open Architecture Platform (HOAP-3) robot. The generated joint trajectories of the right and left limb joints are as shown in Figure 4.1 and Figure 4.2 respectively.

As a starting point, these joint trajectories which are nonlinear and varied, are considered. Taking into account all the different features of the trajectories, a model that can approximately fit these trajectories is developed. In the following subsections, trajectory of the right hip joint (in the pitch axis) is considered as reference, and the process of incrementally developing a model, one feature at a time, that fits this trajectory is being explained.

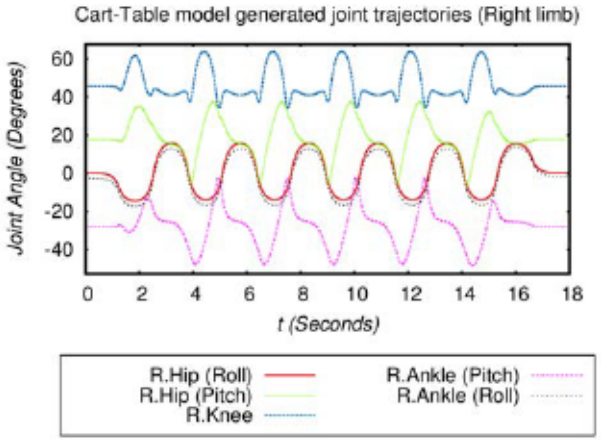


Figure 4.1: *Joint trajectories of the right hip, knee and ankle joints generated based on the cart-table method.*

4.1.1 Asymmetric Triangle Wave

— — – – — (4.1)

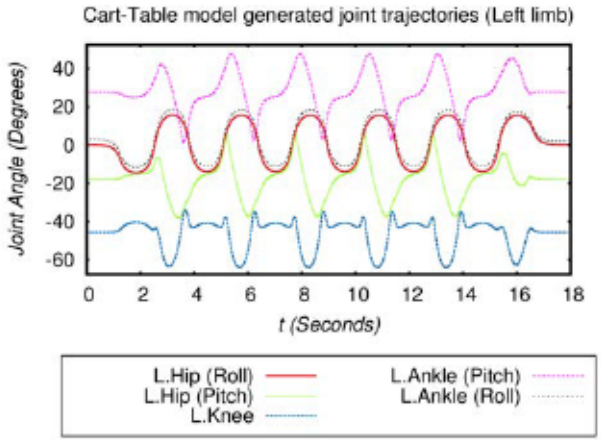


Figure 4.2: *Joint trajectories of the left hip, knee and ankle joints generated based on the cart-table method.*

(4.2)

where ( ) is the amplitude, ( ) is the offset, ( ) is the phase, ( ) is the period and ( ) is the symmetry parameter.

A sawtooth wave function is defined in Equation 4.1. A triangle wave function, as a combination of sawtooth wave and reverse sawtooth wave ( ), is defined in Equation 4.2, where parameter defines the symmetry of the resulting triangle wave. If , then the resulting triangle wave is symmetric, else if , or if then the resulting triangle wave tends to lean towards sawtooth wave and reverse sawtooth wave forms respectively. Figure 4.3 contains examples of symmetric and asymmetric triangle waves generated by function (Equation 4.2).

The cart-table model generated right hip (pitch) joint trajectory, and a linear approximate of it based on the triangle wave function, is as shown in Figure 4.4. Parameters of this linear approximate, which are hand-tuned, are as presented in Table 4.1.

4.1.2 Dual Triangle Wave

From Figure 4.4, simplicity of the triangle wave-based model is quite evident. By tuning the parameter, only one of the two halves, either the top or the bottom-half, of the original trajectory

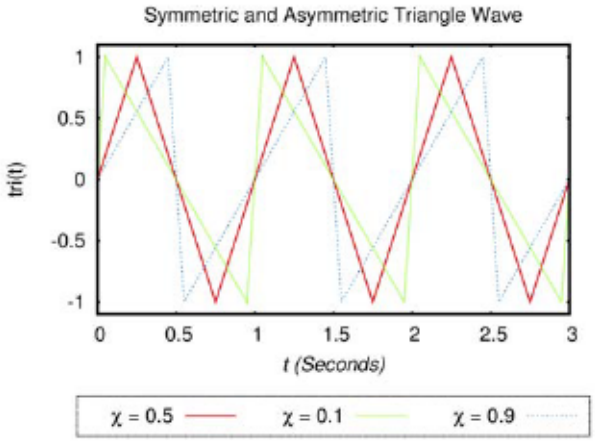


Figure 4.3: *Triangle waves with (red), (green) and (blue).*

Parameter	Value

Table 4.1: *Triangle wave function parameters for generating the linear approximate trajectory.*

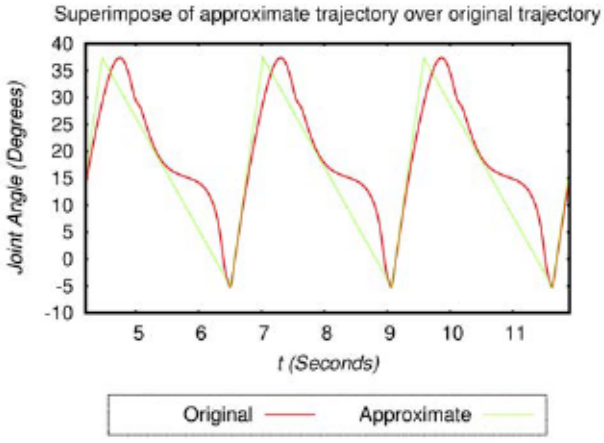


Figure 4.4: *Original trajectory, and the triangle wave based approximate of it.*

can be modeled. So, the current model is extended by considering the two halves independently in the following way:

(4.3)

(4.4)

where is a pair of amplitude parameters, and is a pair of symmetry parameters.

In Equation 4.4, top-half of one triangle wave and bottom-half of another triangle wave (dual triangles), each with independent amplitude and symmetry parameters, are combined together to produce trajectories that have independent halves (top and bottom). Some example trajectories with independent (asymmetric) halves are as shown in Figure 4.5.

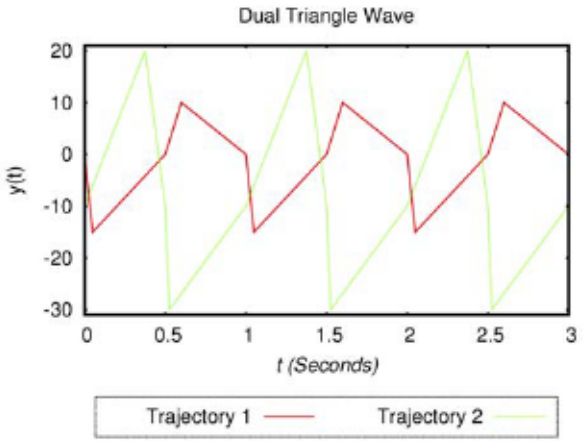


Figure 4.5: *Triangle waves with asymmetry between the top and bottom halves.*

Figure 4.6 contains the original trajectory, and a linear approximate of it based on the dual triangle wave model. Parameters of this linear approximation, which are hand-tuned, are as presented in Table 4.2.

4.1.3 Width Modulation

The dual triangle wave-based approximate, fits the original trajectory much closer than the triangle wave model does, but there still exists a discrepancy between the two. The linear model

Parameter	Value

Table 4.2: *Parameters of the dual triangle wave based function, for generating the linear approximate trajectory.*

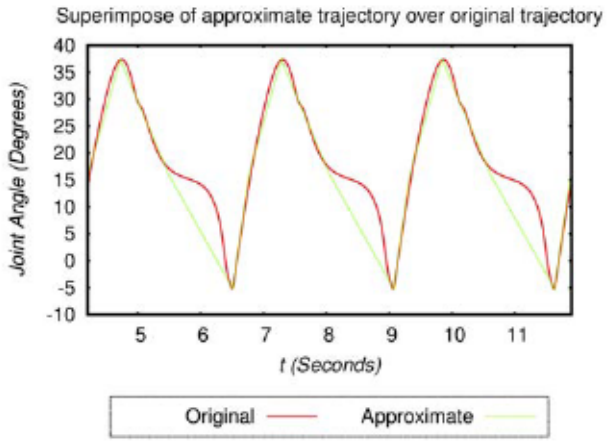


Figure 4.6: *Original trajectory, and the dual triangle wave based approximate of it.*

is further enhanced by adding duty cycle feature to it, as follows:

(4.5)

(4.6)

where is a pair of duty cycle parameters.

In Equation 4.5, width of a sawtooth wave, within a cycle, is modulated. If , then width of the top-half of the sawtooth wave is shrunk inversely proportional to parameter , while if , then the duty cycle of the signal is . Similarly, parameter determines the width of the bottom-half of the sawtooth wave. In Equation 4.6, width-modulated sawtooth and reverse sawtooth waves are combined together to produce a width-modulated triangle wave. Examples of width-modulated triangle waves, with duty cycles of and , are as shown in Figure 4.7.

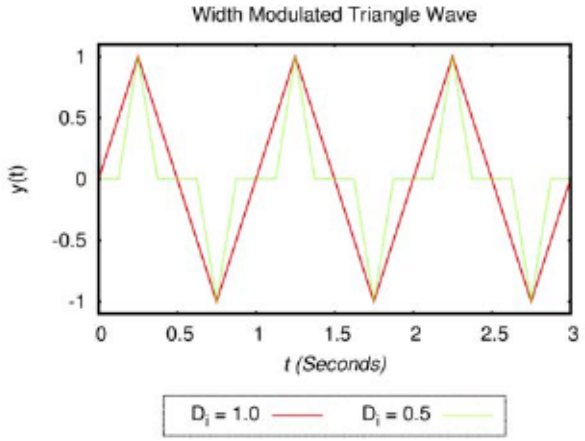


Figure 4.7: *Triangle waves with duty cycle of (red) and (green).*

4.1.4 Skewness

(4.7)

where is a pair of skewness parameters.

Position over the -axis (time) of a width-modulated triangle wave, within a cycle, can also be modulated by introducing the skewness factor into Equation 4.5 as defined in Equation 4.7.

Parameter in Equation 4.7 determines the position, on the -axis, of the upper width-modulated triangle. If then the triangle is positively skewed, else if then the triangle is negatively skewed, else if then the triangle is not skewed. Similarly,

parameter determines the skewness of the bottom triangle. The value of the determines the skewness of the triangle, and is only a factor if the triangle has a modulated-width (i.e. if). If the triangle has a duty cycle of , then has no effect on the resulting triangle. Examples of positively and negatively skewed width-modulated triangle waves are as shown in Figure 4.8.

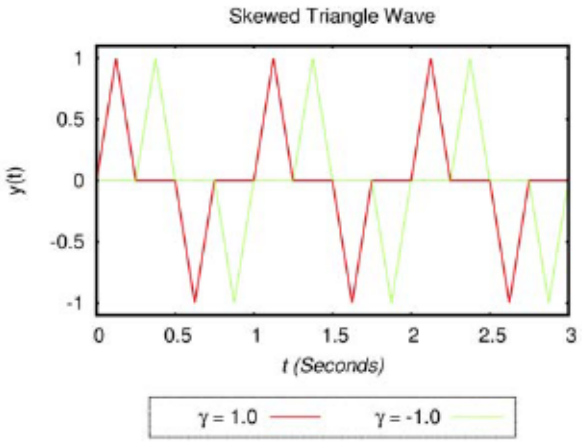


Figure 4.8: *Width-modulated triangle waves with positive skew (red) and negative skew (green).*

The original trajectory in comparison with the linear approximate generated from the updated model is as shown in Figure 4.9. Parameters of this linear approximation, which are hand-tuned, are as presented in Table 4.3.

Parameter	Value

Table 4.3: *Parameters of the triangle wave function, with duty cycle and skewness factors, for generating the linear approximate trajectory.*

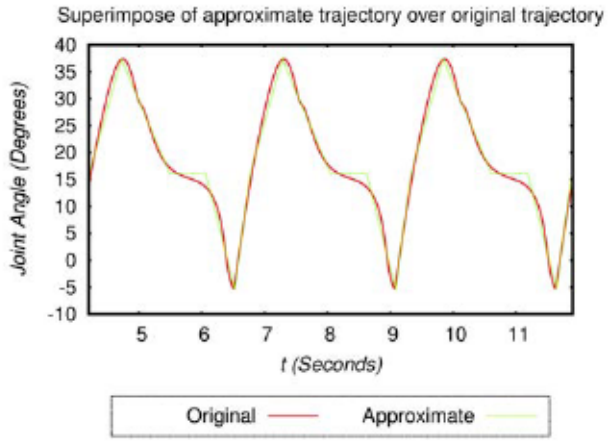


Figure 4.9: *Original trajectory and its linear approximate based on the model with duty cycle and skewness factors.*

4.1.5 Squareness

The updated approximation with modulated-width and skewness factors, fits the original trajectory much better compared to the previous model. The linear model is further enhanced by adding the squareness factor as follows,

(4.8)

(4.9)

where is a pair of squareness parameters.

In Equation 4.9, parameter determines how square or triangular the signal is. If then the upper half is of a perfect triangular shape, else if , then the top part of the upper half of the signal is clipped, and the signal is resized by increasing the amplitude parameter in proportion as defined in Equation 4.8. Magnitude of the parameter determines the squareness of the signal. Similarly, determines the squareness of the bottom-half of the signal. Some examples are as shown in Figure 4.10.

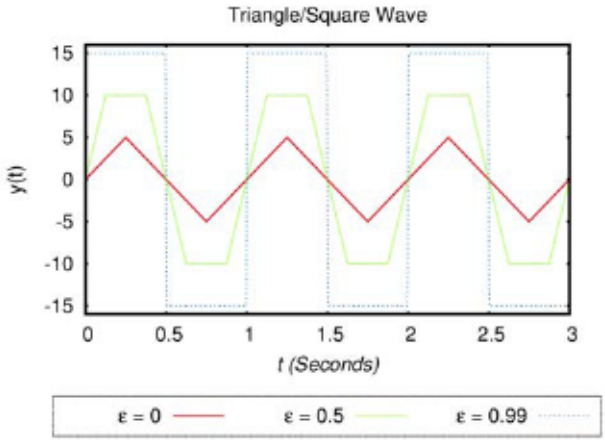


Figure 4.10: *Trajectories with triangular (red), semi-triangular (green) and square wave forms.*

The new linear approximate, with squareness factor included, compared to the original signal is as shown in Figure 4.11. Hand-tuned parameters of this linear approximation are as presented in Table 4.4.

Parameter	Value

Table 4.4: *Parameters of the triangle wave function, with squareness factors, for generating the linear approximate trajectory.*

Similarly to the right hip (pitch) joint trajectory, an approximate linear fit to the original (cart-table model generated) trajectory of right knee joint is modeled, and is as shown in Figure 4.12. Hand-tuned parameters for this linear approximate model are as presented in Table 4.5.

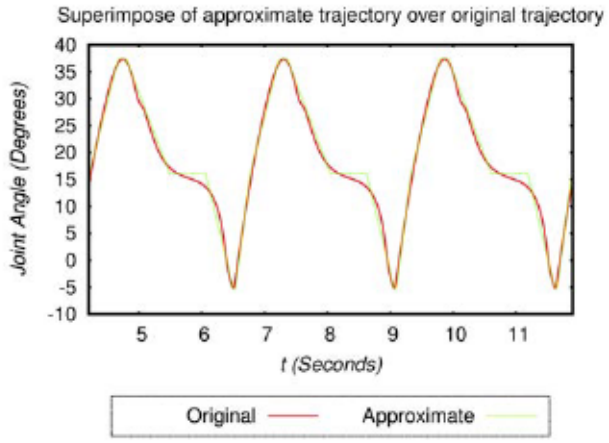


Figure 4.11: *Original trajectory and its linear approximate based on the model with squareness factor.*

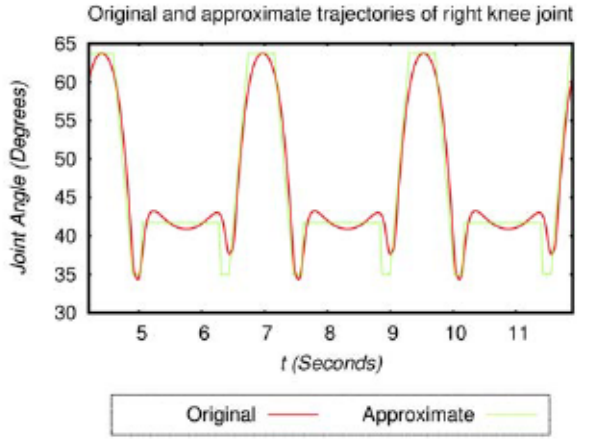


Figure 4.12: *Right knee joint: The original trajectory and its linear approximate.*

4.2 Approximating Cart-Table Model

Similar to right hip (pitch) and right knee joints, linear approximates of the remaining eight cart-table model generated joint trajectories (Figure 4.1 and Figure 4.2) are generated by hand-tuning Triangle/Square wave function parameters. Then, the resulting trajectories are tested on the simulated model of the small-sized ( ) Humanoid for Open Architecture Platform (HOAP-3) robot, in an Open Dynamics Engine (ODE)-based physics simulator OpenRAVE [Diankov and Kuffner, 2008].

Each evaluation starts with the robot at the default stand-still position, where all the joints are at . Since all ten joint trajectories generated, oscillate at a non-zero-center amplitude (i.e.), a sudden displacement of the joint positions at time perturbs the Center of

Parameter	Value

Table 4.5: *Parameters of the Triangle/Square wave function for generating the linear approximate trajectory of the right knee joint.*

Gravity (COG) of the robot, and results in the robot loosing its balance. To overcome this, joint trajectories are modified to crescendo to full intensity very slowly, using the following stabilization filter:

– if

otherwise

where is the stabilization period parameter.

Parameter defines the period, starting from time , during which the joint trajectory linearly increase in intensity, starting from the initial joint position of . Filtered joint trajectories of the right limb joints are as shown in Figure 4.13. In this experiment, stabilization period for all the joint controllers are set to , which is determined empirically.

Stabilized linear joint trajectories, which are pre-generated, are evaluated on the simulated robot, and of the evaluations results in a stable bipedal walking gait. The hip, knee and ankle joints, all start to oscillate slowly, while increasing in intensity at every time step ( ). At around , the robot starts to take the first couple of very small steps backwards, as the torso gradually leans forward at the same time. Then at around , the robot takes a big step forward, compensating for the forward lean, and within the next two-to-three steps, the robot stabilizes its balance, and enters into a rhythmic cycle. Then, the robot continues to walk in a stable gait, at an average walking speed of , until the end of the evaluation (a period of). Screen capture of one gait cycle of the stable gait is as shown in Figure 4.14 [1].

In the failed evaluations ( ), the robot does usually loose its balance at around , as it tries to enter a stable gait cycle. Although the reference joint trajectories are the same for all the

[1] Video at:

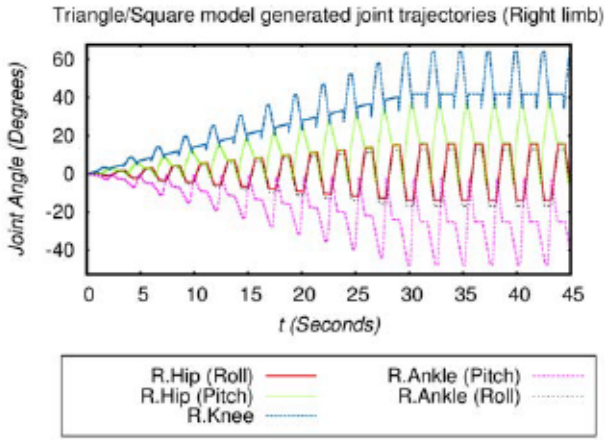


Figure 4.13: *Joint trajectories generated by Triangle/Square model, crescendo to full intensity, starting from .*

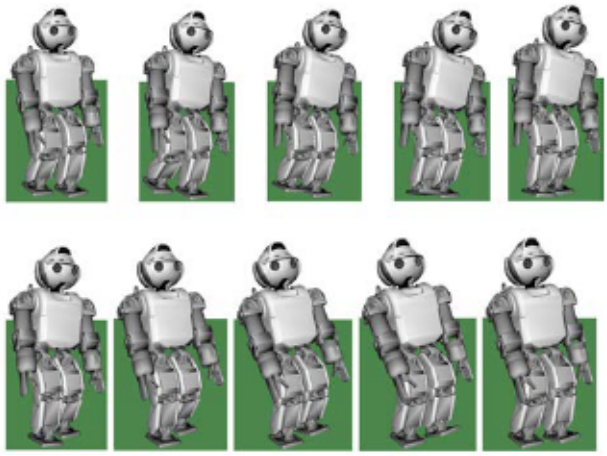


Figure 4.14: *Screen capture of one gait cycle during stable bipedal gait, starting from top-left and ending at bottom-right, one row at a time.*

evaluations, the robot fails to enter a stable gait cycle on instances because of the stochasticity modeled in the simulation environment, combined with the fact that the linear trajectories are only an approximate of the original.

4.3 Learning parameters through GA

The natural next step in this research is to evaluate this controller by learning a gait through a model-free bottom-up approach, with minimum to no modeling at all. The objective is to

optimize control parameters for generating stable bipedal gait on the simulated HOAP-3 robot, with as minimum modeling of the robot as possible. Evolutionary Algorithm (EA) is used for optimizing the control parameters, wherein speed of locomotion is used as the fitness function. The HOAP-3 robot has twenty-eight joints in total, of which, only ten joints, that are relevant for locomotion, are controlled, while the rest of the joints are maintained at a constant default position.

To produce a meaningful gait, , the period parameter, has to be a common value for all the joints, although needing to be optimized. Considering this, the total number of parameters, including one parameter per joint, that needs to be optimized would be a common parameter parameters, making it a search problem in a 131 dimensional space. To reduce the dimension of the search space, the following constraints are applied,

> In bipedal walking gait, there exists symmetry between respective joints of the two legs, such that , and , , where . That is, at any point in time , joint angles of respective left and right leg joints are opposite of each other, along with an approximate phase-difference of between them. By taking advantage of this, dimension of the search space can be reduced by a factor of 2, from 131 dimensions, down to 66 dimensions, by modeling joint trajectories of all the joints of one leg, based on the respective joint trajectories of the other leg.
>
> In bipedal walking gait, there also exists a symmetry between the shapes of hip-roll and ankle-roll joint trajectories of each leg, varying only in amplitude. This feature can be used to further reduce the search space, by modeling the ankle-roll joint over the hip-roll joint, such that the only parameters needed to be optimized for the ankle-roll joint are the amplitude parameters and , further reducing the dimension of the search space from 66 dimensions down to 55 dimensions.
>
> Based on the kinematic model of the HOAP-3 robot, range of some control parameters such as amplitude and offset, and that of the parameter are reduced, resulting in narrowing the width of the search space. Control parameters are optimized in the range as presented in Table 4.6.

Control parameters are optimized through Genetic Algorithm (GA), implementation details of which is presented in the next chapter, in subsection 5.3.1, page 125. Table 4.7 contains GA parameters employed for evolving the gait.

Figure 4.15 plots fitness value of the best candidate and average fitness of the population at the end of each generation. At the end of the evolution, the optimized controller is able to produce a very stable bipedal walking gait, with a success rate of , and at an average locomotion speed of , compared to the average locomotion speed of achieved with the modeled controller. The evolved gait has a low period of . The robot takes small but quick steps, barely lifting its feet off the ground, which ensures stability, while the low period

Parameters	*Minimum*	*Maximum*

Table 4.6: *Range of control parameters used while optimization.*

Parameters	*Value*
Population size	200
Genome size	55
Evaluation period	
Evolution length	23 generations
Crossover rate	
Elite population size	
Mutation rate	$\frac{}{\text{Size of genome}}$

Table 4.7: *GA parameter values used for evolution.*

value result in faster walking gait. Average step length of the evolved gait is with a SD of , while the average step length of the modeled gait is with a SD of .

4.4 Discussion

Here, the objective is to develop a linear periodic function that is feature-based, relatively simple and that can produce a wide range of trajectories for locomotion. As a first step, validity of this control model is tested by creating an approximate of a previously known stable trajectory. COG of the robot is not explicitly considered while modeling the approximate trajectories, but is implicit since the reference trajectories are modeled based on this consideration. All 13 parameters, of each trajectory generator, are hand-tuned to model the respective reference trajectory as close as possible, but a least squares method can be used as an alternate as well. The hand-tuned controller is able to produce a stable bipedal walking gait on the simulated HOAP-3 robot, validating the viability of the developed controller.

Then, as a second step, control parameters are learnt in a bottom-up approach, based only

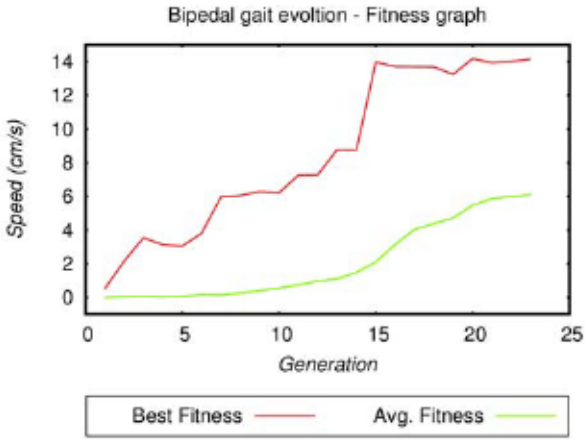


Figure 4.15: *Graph showing the fitness value of best individual and average fitness value of the population during evolution.*

on the stability and speed of the resulting gait. GA is used for optimizing control parameters, during which, candidate controllers resulting in the robot falling over at any point during the evaluation, are give a fitness value of 0. This results in exerting selective pressure towards candidate solutions that do not fall over, albeit moving very little during the early part of the evolution. Also, the first generation of evolution is populated with randomly initialized candidate controllers whose fitness is above the threshold of , which disqualifies all candidate solutions resulting in the robot falling over during evaluation. At the end, the evolution process is able to produce a gait that is very stable, with a success rate of , and almost faster than the gait produced by the trajectories based on the cart-table method, further validating the viability of the developed controller.

The proposed model can also be used as an online controller for producing gaits, both in humanoids and other legged robots. It can also be used as a lower-level controller within a larger framework, wherein another higher-level controller would modulate the lower-level controllers' parameters, based on sensory inputs for maintaining balance, avoiding obstacles, etc. Since the proposed model is feature-based, depending on the complexity of the robot and/or the gait, certain features of the periodic function can either be turned off, or kept at a constant, and thereby reducing the number of parameters that needs to be tuned. By compromising the feature that dictates the asymmetry factor between the upper and the lower half of a trajectory, tunable parameters count per joint can be dropped from 14 down to 10 parameters.

In this primary version of the developed controller, although the reference trajectories produced are strictly linear, the actual trajectory generated by the joint actuators during walking, are nonlinear. Figure 4.16 contains reference and actuator join trajectories of the left knee joint, recorded while evaluating the modeled linear approximates of the cart-table method based joint trajectories.

The main objective of here is not only to develop a method for generating simplified models

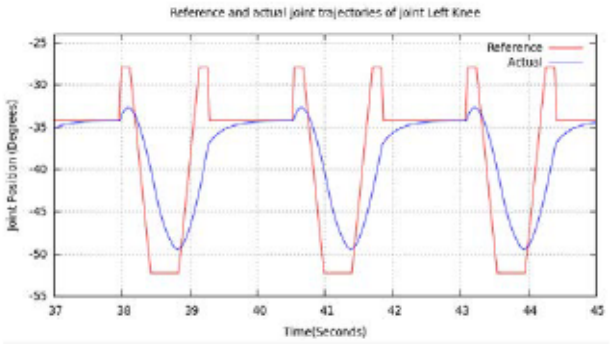


Figure 4.16: *Reference and actuator joint trajectories of the left knee joint, during evaluation.*

of preexisting gaits, but to create a framework through which, stable gaits can be learnt from scratch, needing minimum modeling of the robot. Using the developed controller, it has been able to be proven that a stable bipedal walking gait can be learned through a model-free bottom-up approach.

Summary of the chapter

In this chapter, a feature-based linear period function for producing a wide variety of joint trajectories for locomotion, is presented. The feature-based linear period function is developed incrementally by first combining a sawtooth wave and a reverse sawtooth wave function to produce a triangle wave. Symmetry of the triangle wave is determined based on the proportions of the sawtooth and the reverse sawtooth waves combined. Asymmetry between the two halves of the wave is introduced by combining two asymmetric triangle waves, such that the two halves can be modeled independent of each other. Duty-cycle feature is introduce to this function, such that the width of the wave, within a cycle, can be modulated. Next, skewness feature is added, such that the position of a width-modulated wave, withing a cycle, can be modulated. Finally squareness feature is added, which determines 'how squarish' versus 'how triangular' the generated wave is. With these features, by tuning the parameters, the developed periodic function can generate a wide variety or wave forms, including (virtually) pure sawtooth, reverse sawtooth, triangle and square waves.

Joint trajectories of a previously proven stable bipedal gait, which is developed based on the cart-table method, is taken as reference, and linear approximates of 10 difference joint trajectories are modeled. Modeled linear approximates are evaluated on the simulated HOAP-3 robot, which results in a walking gait that has a success rate of , at an average speed of , and thereby validating the viability of the developed control model.

Next, a stable walking gait is evolved from scratch on the simulated HOAP-3 humanoid, by optimizing controller parameters through EA. Some kinematic aspects of the robot is taking into consideration, resulting reducing size of the parameters dimension, and inturn the size of the search space. The evolved gait is stable and fast, with a success rate of , and at an average speed of .

CHAPTER 5

Locomotion through Embodied Evolution

Introduction

In the chapter 3, page 45, a variety of locomotion controllers for Modular Robot (MR)s are presented, and in the previous chapter, a complex linear trajectory generator, as a locomotion controller for humanoid robot is presented. In both these chapters, gaits are evolved and evaluated on modular robotic configurations, and on a humanoid robot respectively, in simulation environment. Performing evolution in a simulation environment has some obvious advantages, such as:

Fitness calculation, by evaluating performance for a candidate solution, is straight forward in simulation.

Sensory data in simulation is clean and accurate, while noisy and dirty in the real-world.

It is faster, easier and cheaper to construct a robot model in simulation, than it is to construct a real robot.

Evolution can be accelerated by multi-threading in simulation, which is not possible to do so in the real-world.

Unlike real robots, simulated robots do not wear out, overheat, breakdown or run out of battery.

Once evolved in simulation, the optimized controller can be transfered onto a real robot. But there exists a reality gap between the simulation environment and the real-world, as not every aspect of the real-world can be modelled accurately in simulation.

The objective in this chapter is to develop a framework to perform evolution in the real-world, on a physical modular robotic configuration, which is called Embodied Evolution (EE). Some important work in the field of EE includes [Watson et al., 1999], [Ficici et al., 1999] and [Takaya and Arita, 2003]. Contributions in this chapter include:

Constructing a real modular robotics configuration.

Developing, both hardware and software, for estimating module's actuator positions.

Implementation of Kalman Filter (KF) for actuator's state estimation.

Developing a vision system for robot's position estimation.

Developing a communication protocol for communication between the MR and the host personal computer (PC).

Implementation of a EA, as a combination of GA and Evolutionary Strategy (ES), which is used throughout this thesis.

Evolving a gait on a real modular robotics configuration.

5.1 Robot hardware

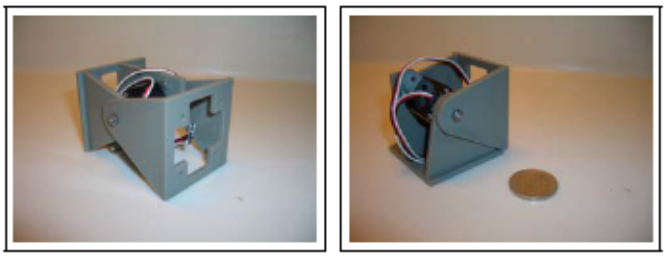
Figure 5.1 shows three-dimension (3D) printed *Y1* robot modules, developed by Dr. Juan Gonzalez-Gomez [Gonzalez-Gomez, 2008] as part of his doctoral research, which are used for constructing a real *Y-bot* modular robotic configuration, in this thesis. The *Y1* platform is modelled on Dr. Mark Yim's PolyBot system [Yim et al., 2000]. *Y1* module is designed to use *Futaba 3003* servomotor, or an equivalent as its actuator. Characteristics of a *Y1* module is as presented in Table 5.1.

A four-module *Y-bot* configuration (Figure 5.2) is constructed using *Y1* modules for experiments. The and the modules are connected to each other using nuts and bolts, while module and are connected to the module, and to each other, using zap-straps.

5.1.1 Electronics

SkyMega (Figure 5.3), an Arduino compatible controller board developed by Dr. Juan Gonzalez-Gomez[1] is used for controlling *Y1* modules. *SkyMega* contains a 16Mhz ATMEGA microcontroller,

[1] http://www.iearobotics.com/wiki/index.php?title=SkyMega



(a) (b)

Figure 5.1: *Y1 modules at (a) rest position and (b)* .

Characteristics	*Value*
Dimension	
Weight	
Material	PLA plastic
Degree of Freedom (DOF)	
Servo	Futaba s3003
Torque	
Rotation velocity	
Rotation range	

Table 5.1: *Characteristics of Y1 modules.*

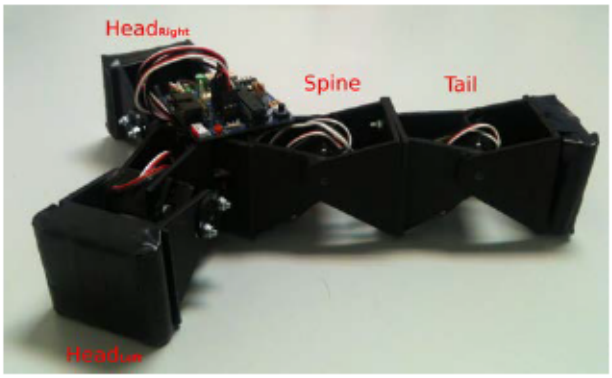


Figure 5.2: *Y-bot configuration.*

and is designed to perfectly fit on either side of a *Y1* module (Figure 5.4). Characteristics of the controller board are as presented in Table 5.2.

High-level controller of each module in a modular robotic configuration, runs on a Linux (Ubuntu 14.04) desktop, while low-level controller for controlling, as well as sensing, actuator position of modules are implemented on the *SkyMega* controller board. A single board can control up to four modules in parallel. During the experiments, a single controller board mounted

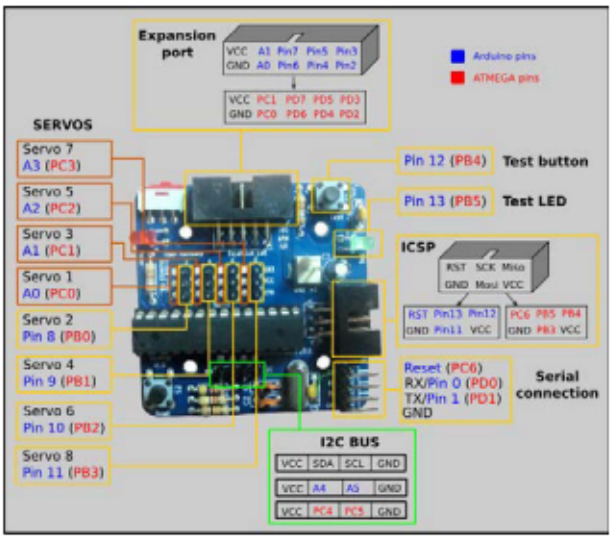


Figure 5.3: *SkyMega controller board with pin layout. Marked in red are ATMEGA microcontroller pings, and marked in blue are Arduino equivalents. Source:*

Figure 5.4: *SkyMega board mounted on a Y1 module. Source:*

on the modular robotic configuration, is used for controlling all the modules, and the board is powered externally.

Unlike several other high-end servomotors available off the shelf, *Futaba S3003* does not feature a position sensor. So, a position sensor is developed by hacking the servomotor to read its potentiometer values. By soldering a wire to the potentiometer of the servomotor, and

Characteristics	*Value*
Microcontroller	ATMEGA 16Mhz (Model:168)
Dimension	
Input ports	4
Output ports	4 to 8
Inter-board communication	I2C
Power input	

Table 5.2: *Characteristics of the SkyMega controller Board.*

connecting it to the analog input of the controller board, and by using the internal Analog to Digital Converter (ADC) of the microcontroller, position of the servomotor is estimated.

5.1.2 Regression model for actuator position estimation

Estimating servomotor position by reading its potentiometer value requires mapping from volts (potentiometer) space to degrees (servomotor position) space. So potentiometer value for every servomotor position, at a resolution of one degree, of six different servomotors are collected. The data is collected by sweeping each servomotor from to and from to , over several sweeps, and recording the potentiometer value at a resolution of one degree. The collected data is as plotted in Figure 5.5.

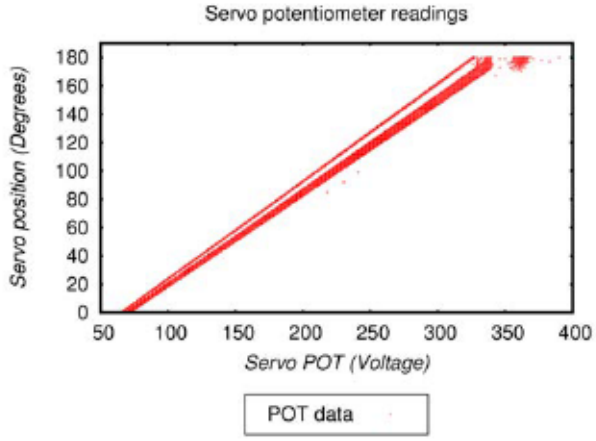


Figure 5.5: *Plot of servomotor position over potentiometer value.*

The collected data has variation between different servomotors, as well as variation based on the servomotor position. For example, potentiometer value recorded at the position of has minimum, maximum, mean and standard deviation (SD) of 65 volts, 72 volts, 69.33 volts and 1.96 volts respectively. While at , minimum, maximum, mean and SD of the recorded potentiometer readings are 196 volts, 212 volts, 206.23 volts and 5.09 volts respectively. SD of

the potentiometer reading increase linearly with respect to the servomotor position as plotted in Figure 5.6. Also, between servomotor positions and , potentiometer readings have very high variance. This is due to the design of the *Y1* module, which restricts the two halves of the module, connected by a servomotor, to swing freely at higher extreme ends, due to friction between the module's hinge.

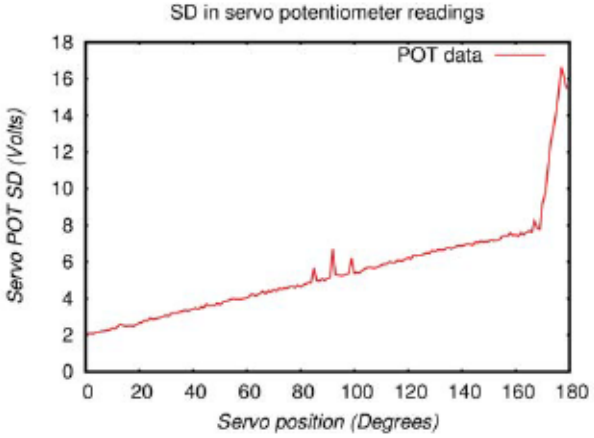


Figure 5.6: *Plot of SD in servo potentiometer reading, with respect to servo position.*

To correctly estimate servo position based on potentiometer value, a linear model is fitted to the collected data, by running the linear regression algorithm over the data. The linear model is a fourth degree polynomial, and is of the form Equation 5.1.

(5.1)

where is the potentiometer value, is the predicted servomotor position and are the learnt weights.

The linear model is trained by splitting the data set into training and test sets of and respectively. The above model fits the data very well, with a training cost of , and a cost of on the test data. The fitted linear model to the data is as shown in Figure 5.7, and the parameters of the learnt model is as presented in Table 5.3.

5.1.3 Actuator state estimation

Using the linear model, given the potentiometer value of a module's servomotor, position of the module's actuator can be estimated. Applying a simple sinusoidal oscillator to a module's actuator, and then estimating the module's position by sourcing it's potentiometer values, results in a noisy estimate of the actuator's position. An example of this is as shown in Figure 5.8.

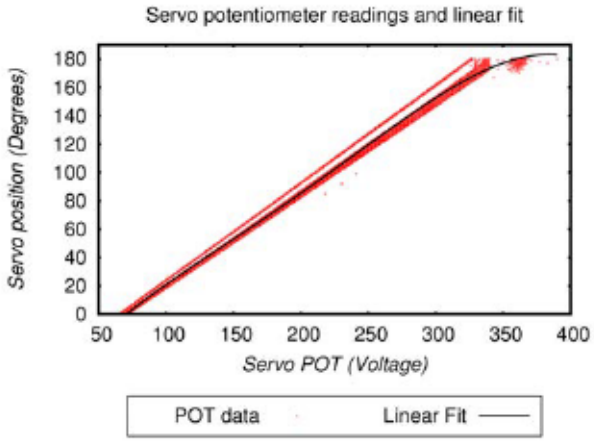


Figure 5.7: *Plot of servomotor position over potentiometer value and the linear model.*

	Value

Table 5.3: *Parameters of the learnt linear model.*

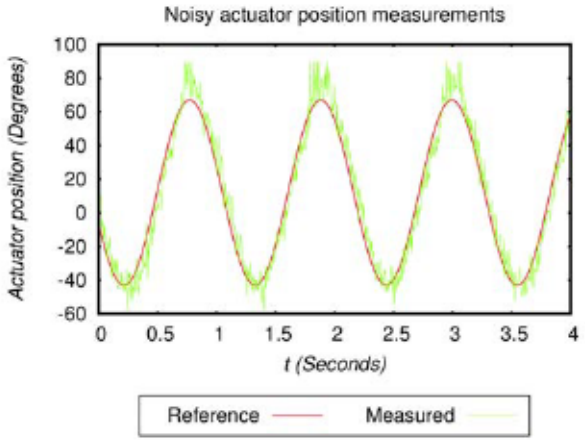


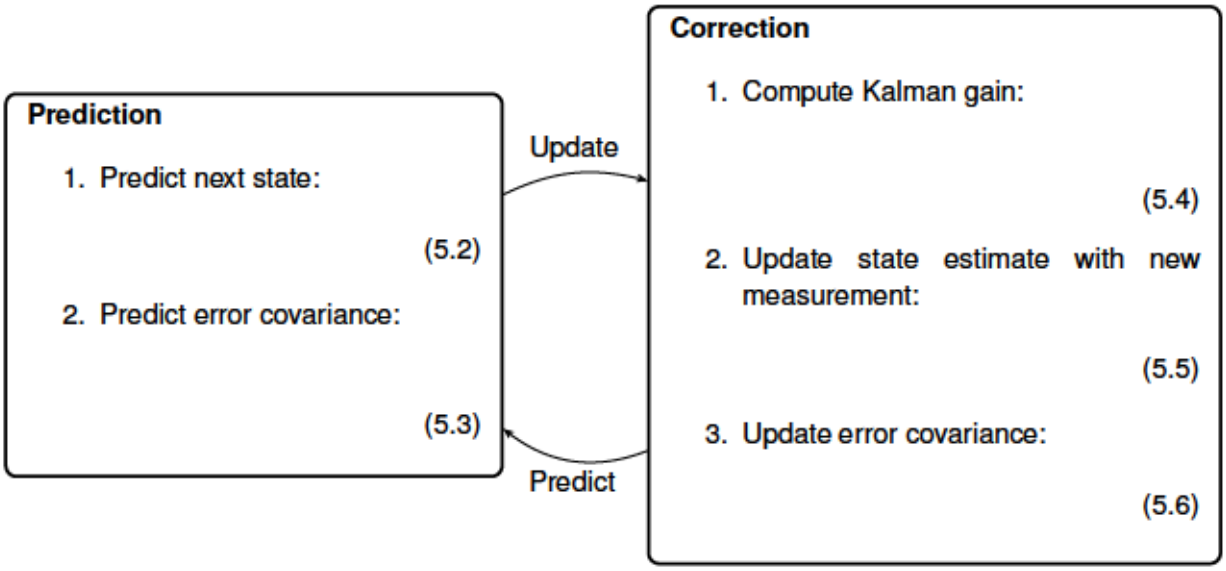
Figure 5.8: *Plot of reference position and noisy measurements of a module's actuator.*

Raw position estimation, based on potentiometer values fitted with the linear model, are very noisy, especially at higher actuator positions. This is because, when a module swings down (i.e,), it presses down on the ground surface, overcoming both, the friction of the ground

surface, as well as the mass of the configuration. So, it consumes high energy at higher actuator positions, which is reflected in the sensor readings. To correctly estimate the true position of a module's actuator, a Kalman Filter (KF) is implemented for filtering noisy sensor data.

5.1.3.i Kalman Filter

A KF is a recursive two-step algorithm used for state estimation based on noisy observations. The algorithm, based on prior knowledge of the system in the form of process model and its uncertainty, and noisy measurements, estimates the true value of the state of the system. The two-step process includes the prediction step, where the algorithm predicts the possible next state of the system, based on a state transition matrix and control variable, followed by a correction step, where the algorithm updates the state and its process model's uncertainty, based on the most recent observation of the state through a (possibly noisy) measurement. The algorithm is as presented below.



where,

: Estimated state

: State transition matrix

: State variance matrix (i.e, error in estimation)

: Process variance matrix (i.e, error due to process)

: Measurement variable

: Measurement matrix (i.e, mapping measurement onto states)

: Kalman gain

: Measurement variance matrix (i,e. error from measurement)

Subscripts indicate the current time step, indicate the previous time step, and indicate the intermediate time step.

Objective of a KF is to estimate the state given measurement . The two-step algorithm can be seen as: (i) Prediction: Project forward (in time) the current state and error covariance estimates to obtain a priori estimates for the next time step, (ii) Correction: Incorporate a new measurement into the a priori estimates to obtain the updated a posteriori estimates.

In Equation 5.2 and Equation 5.3, state estimate and error covariance estimate, respectively, of the system are projected forward in time ( ), given the estimate of the previous time instance ( ).

In the correction step, Kalman gain is calculated first in Equation 5.4, followed by an update to the state estimate in Equation 5.5, incorporating the latest measurement , and an update of the error covariance estimate in Equation 5.6.

In the state estimate correction step, the a posteriori state estimate is calculated as a linear combination of an a priori estimate and a weighted difference between the actual measurement and a measurement prediction . Here, the difference is called the residual, which is the discrepancy between the predicted measurement and the actual measurement . A residual of zero indicates that the two are in complete agreement.

Kalman gain is calculated so as to balance the trust between measurement error and estimated error covariance . That is, as the measurement error covariance approaches zero, the actual measurement is trusted more and more, while the predicted measurement is trusted less and less. On the other hand, as the a priori estimate error covariance approaches zero, the predicted measurement is trusted more and more, while the actual measurement is trusted less and less.

Sinusoidal Model KF

The controller for the actuator, whose position is estimated, is sinusoidal,

(5.7)

and its derivative is,

— (5.8)

where is the amplitude, is the angular frequency, is the phase, is the offset and is the frequency.

For estimating position of an actuator, which follows a roughly sinusoidal trajectory, the system can be modelled as follows,

were is the position of the actuator, and is the constant amplitude of the sinusoidal modelled.

New state at time can be modelled as,

(5.9)

(5.10)

The continuous time state transition matrix is,

(5.11)

The continuous time process noise matrix is,

(5.12)

where is the scalar process noise.

The discrete time state transition matrix would then be,

(5.13)

Given continuous time state transition and process noise matrices and , discrete time process noise matrix can be calculated as follows,

(5.14)

Therefore, discrete time process noise matrix would be,

(5.15)

where , and is the process noise.

Measurement variable in this case is the position estimate based on the voltage of the potentiometer of the servomotor, as explained in subsection 5.1.2, page 109. Since the linear model used for estimating the position of the servomotor maps from voltage space to degree space, measurement matrix would be,

In this case, measurement variance matrix would be a scalar value , that models the noise of the sensor reading. Process noise and measurement noise are assumed to be independent of each other, white, and follow a normal distribution.

Given the above model, random variables and initialized to and respectively, and error covariance matrix and state estimation vector for are initialized as follows,

Given the above model and the initializations, by running the KF on noisy sensor data (Figure 5.8), a more accurate position estimate of the actuator is obtained as shown in Figure 5.9.

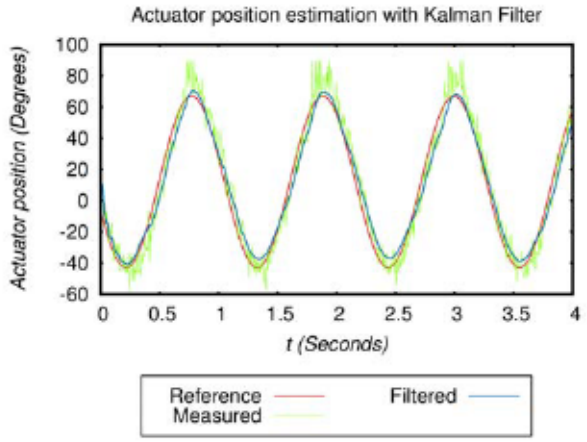


Figure 5.9: *Plot of reference position, noisy measurements and KF based estimate of a module's actuator.*

5.2 Experimental setup

5.2.1 Vision system

In simulation, fitness value of a candidate solution is calculated based on the positions of the robot in the simulated world, prior to and after the evaluation. These positions are easily sourced from the simulation engine. But in Embodied Evolution (EE), where fitness value evaluation of candidate solutions are to be performed on a real robot, a method for estimating two-dimensional (2D) positions of the robot in the real-world, before and after the evaluation, is needed. To achieve this, a two-dimension (2D) vision system, consisting a Red Green Blue (RGB) webcam and a colored marker on the robot, is developed. The EE setup, which includes the 2D vision system, is as shown in Figure 5.10.

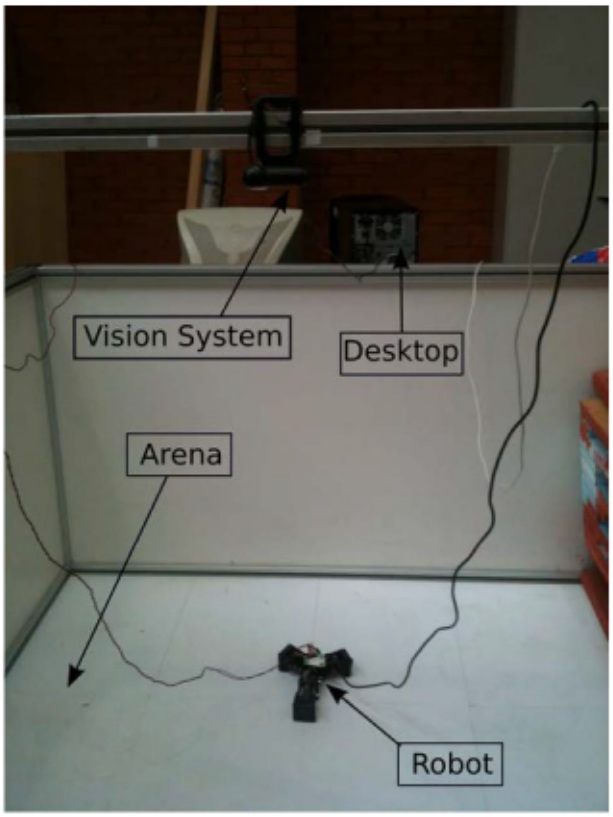


Figure 5.10: *EE arena, with the vision system, host PC and the robot, where the gait is evolved on the real robot.*

The colored-marker placed on the robot is as shown in Figure 5.11. The color of the marker is selected to be a specific shade of green, which is neither used in any part of the robot, nor in the EE arena. The marker consists of three circular-components, placed in an isosceles triangle arrangement, such that the two equal sides measuring , the length of the base of the

triangle. , are the centers of the three circular-components.

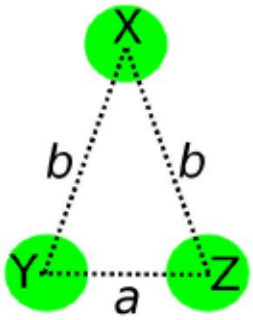


Figure 5.11: *Colored-marker*

The overhead webcam is placed at a known distance of from the ground surface, based on which, position of the robot in the 2D arena is calculated in two parts: (i) By estimating pixel-wise position of the robot in the image space, by first detecting the colored-marker, and then calculating its centroid, and (ii) by calculating 2D position of the robot in the real-world, given camera intrinsics and pixel-wise position of the robot in the image space.

5.2.1.i Marker detection

Algorithm for detecting the colored-marker and calculating its centroid in the image plane is as follows,

Result: Detect colored-marker and calculate its centroid
Reset robot to default state;
while *Marker centroid not estimated* **do**
  Get a new image from the webcam;
  Convert image from RGB to Hue Saturation Value (HSV) space;
  Apply an HSV filter, to filter the color of interest;
  Find all connected components in the filtered image;
  if *No. of connected components 3* **then**
    continue
  end
  Calculate center points ( ) of the three largest connected components;
  if ——— **and** ——— **and** ——— **then**
    Calculate marker centroid ————;
    break
  else
    continue
  end
end

Algorithm 1: Vision based, colored-marker detection and centroid calculation algorithm.

The algorithm starts by first resetting actuators of the robot to default state of , which ensures that the colored-marker placed on the robot is parallel to the image plane, and clearly visible to the overhead camera. Then, a color image of the arena, with the robot and the colored-marker, is captured and converted from RGB to HSV color space. The converted image is then passed through a HSV filter, to filter out all but the regions with color of interest. The minimum and maximum values of Hue, Saturation and Value channels, used for filtering, are as presented in Table 5.4. In the resulting binary image, containing blobs of potential marker components, pixels that are next to each other are grouped together, and bounding box of each such group is calculated. If there are fewer than three groups, then it would mean that the algorithm has failed to detect all three circular-components of the colored-marker, and so a new image needs to be captured and reprocessed. If there are three or more groups, then based on the size of the bounding box, the three biggest groups are selected and their center points ( ) are calculated.

The ratio between the base of the triangle shaped color-marker, and the other two sides is . So, if are the center points of each of the three circular-components of the colored-maker, then it should satisfy the following constraints,

(5.16)

(5.17)

(5.18)

In the algorithm, might not be center points of the circular-components, but some point on each of the three circular component. So to test this, the algorithm checks if the above constraints satisfy, but with a larger margin as shown in line 11. If the constraints do not satisfy, then a new image needs to be captured and reprocessed. On the other hand, if they satisfy, then the (pixel-wise) position of the robot in the image space is calculated as the centroid of the colored-marker with Equation 5.19. An example of detecting the colored-marker and estimating its (pixel-wise) position in the image space is as shown in Figure 5.12.

(5.19)

Channel	Min.	Max.
Hue		
Saturation		
Value		

Table 5.4: *HSV filter parameters.*

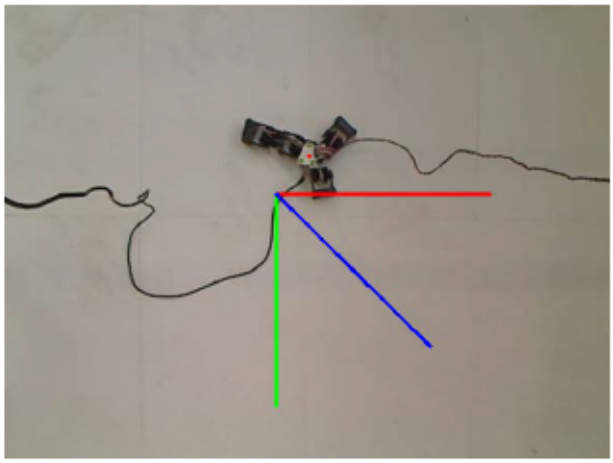


Figure 5.12: *Colored-marker detected, and its centroid estimated.*

5.2.1.ii Robot position calculation

Intrinsic matrix of a camera is defined as,

(5.20)

where and are the focal length in pixels, and are the optical center in pixels, and is the skew coefficient of the camera.

Given the camera intrinsics , known constant distance between the camera's optical center and the arena , and the centroid pixel of the colored-marker, position of the robot in the real-world can be calculated. In , only the first two components and needs to be calculated, as is known. Position of the robot is determined by first calculating the vector in that passes through the optical center and the pixel (Equation 5.21), then calculating the coefficient based on (Equation 5.22), and finally calculating and of as shown in Equation 5.23 and Equation 5.24 respectively.

(5.21)

(5.22)

(5.23)

(5.24)

Given the positions of the robot before and after the evaluation, and respectively, distance traveled by the robot during an evaluation can then be easily calculated as,

(5.25)

5.2.2 Serial Communication Protocol

Low-level controllers, responsible for both driving a module's actuator to a desired position, as well as for estimating a module's actuator position based on its servomotor's potentiometer value, resides on the Arduino based *Skymega* controller board, which is on-board the robot. High-level controller for generating actuator trajectories, as well as the GA code, runs on a

Linux desktop that is off-board the robot. Communication between the robot and the desktop personal computer (PC) is achieved through wired serial communication, by connecting serial communication port of the *Skymega* controller board to a Universal Serial Bus (USB) port of the desktop PC. A *USB TTL Serial Cable*, which is a USB-to-Serial converter, is used for this purpose. The same setup is also used for flashing firmware — Module actuation and sensing controller program — onto ATMEGA microcontroller of the *Skymega* board.

At the lowest level, RS-232 communication protocol is used to transmit byte sized data bi-directionally between the desktop PC and *Skymega* board. A baud rate of 115200, 8 bit data and 1 stop bit are chosen. On top of this RS-232 communication protocol layer, a custom communication protocol, consisting multi-field message frames, for transmitting application specific data between the two devices, is developed as part of this thesis.

5.2.2.i Message frame

#	Metadata																			
	!	ID		%	Type			Address									Time			
								A	-	A	-		-	A						

Table 5.5: *Message frame section 1.*

		Data																
			Data-Field						Data-Field									
			D.Sf	/	D.Sf				D.Sf	/	D.Sf						$	

Table 5.6: *Message frame section 2.*

Table 5.5 and Table 5.6 together form the skeleton of a single message frame. Each message frame has two segments, *Metadata* and *Data*, and each segment has several fields and subfields. American Standard Code for Information Interchange (ASCII) special characters are used as begin and end tags of segments, fields and subfields in the message frame. Table 5.7 describes all the different tags used in the message frame.

Metadata segment of the message frame contains data about each message frame, such as *ID*, *Type*, *Address* and *Time*, while *Data* segment of the message frame contains the actual data that is communicated between the desktop PC and robot. A description of all the fields and subfields in a message frame is as provided in Table 5.8.

ID

Each message has a unique message ID, which is generated by the system (PC/*Skymega*) that creates the message frame. A new message ID is generated by maintaining a counter, which is either initialized to or (for PC or *SkyMega* respectively), and incrementing the counter by

Tag	*Description*
#	Message frame begin
$	Message frame end
!	Message ID begin
	Message ID end
%	Message type begin
	Message type end
	Address begin
	Address end
-	Address segregation
	Time begin
	Time end
	Data segment begin
	Data segment end
	Data-Field begin
	Data-Field end
/	Data subfield segregation

Table 5.7: *Message frame tags description.*

every time a new message frame is generated. So, all message frames generated by the PC would have even-numbered message IDs, while those generated by the *Skymega* would have odd-numbered message IDs.

Type

A message can be one of several different message types, with each message type identified with a unique message type ID. The message type field has unsigned char data type, so the defined message frame architecture can support up to 256 different message types. Message types developed in this work are as follows, which are also summarized in Table 5.9.

Actuate command: Message sent from PC to *SkyMega*, containing address of a module and the desired position of the module's actuator. Address and desired position of either a single module or of multiple modules is supported.

Position request: Message sent from PC to *SkyMega*, requesting current actuator position of one or multiple modules.

Position: Message sent from *SkyMega* to PC, containing current actuator position of one or multiple modules.

Time request: Message sent from PC to *SkyMega*, requesting the current on-board time.

Field / Subfield	*Data type*	*Segment*	*Description*
ID	unsigned long	Metadata	Unique ID of the message frame
Type	unsigned char	Metadata	Type of message
Address	-	Metadata	Addresses of modules the message is sent to / received from
A	unsigned char	Metadata	Unique address of a module
Time	unsigned long	Metadata	Time at which the message is sent
Data-Field	-	Data	One of n data fields containing the actual data
D.Sf	float	Data	Two data subfields within each data field
D.Sf	signed int	Data	

Table 5.8: *Message frame description.*

Time: Message sent from *SkyMega* to PC, acknowledging time request message with current on-board time.

Start position stream: Message sent from PC to *SkyMega*, requesting to start sending position data of all connected modules continuously.

Stop position stream: Message sent from PC to *SkyMega*, requesting to stop position data stream.

Type ID	Message	Origin	Destination
0	*Actuate command*	PC	*SkyMega*
1	*Position request*	PC	*SkyMega*
2	*Position*	*SkyMega*	PC
3	*Time request*	PC	*SkyMega*
4	*Time*	*SkyMega*	PC
5	*Start position stream*	PC	*SkyMega*
6	*Stop position stream*	PC	*SkyMega*

Table 5.9: *Message types*

Address & Time

The *Address* field under *Metadata* section contains addresses of modules the message is targeted at, or addresses of those modules whose information the message frame contains. This field contains varying number of *A* subfields within, each of which in turn contains unique addresses of modules. There is no upper-limit to the number of *A* subfields the *Address* field can contain, but since the data type used for the *A* subfield is unsigned char, the *Address*

field cannot contain any more than 256 unique addresses. Each *A* subfield is separated using the special character '-'. This field is only used for message types *Actuate command*, *Position request* and *Position*, while it is empty for other message types.

The *Time* field holds time, in microseconds, at which the message is sent. Data type of this field is unsigned long, and so every minutes, the time related data rolls over.

Data

The *Data* section contains the actual data transmitted, and holds within it varying number of *Data-Field* fields, each of which contains two subfields *D.Sf* and *D.Sf* within, which are of type float and signed integer respectively. There is no upper-limit to the number of *Data-Field* fields the *Data* section can hold, but longer the *Data* section is, slower the communication (in terms of frames per second) becomes. The reason for having two subfields within each *Data-Field* is to accommodate transmission of data with different data types. Data fields are used for message types *Actuate command* and *Position*, while remains empty for the rest of the message types.

In message type *Actuate command*, the subfield *D.Sf* contains the desired actuator position of a module, while *D.Sf* is unused. A message frame of type *Actuate command* can contain desired actuator positions of multiple modules. For example, a message frame transmitting desired actuator position of three different modules would contain address of the three modules separately in subfields *A* , *A* and *A* , under the *Address* field, while the desired positions of each of the three modules are coded within subfields *D.Sf* of field *Data-Field* , *Data-Field* and *Data-Field* respectively.

Similarly, in message type *Position*, sent from *Skymega* to PC, subfields *D.Sf* contains the estimated position of a module's actuator, while subfields *D.Sf* contains the time at which the potentiometer value of the respective module's servomotor was read. The time information in subfields *D.Sf* is represented in terms of time difference between time at which the potentiometer value was read and the time at which the message is sent. Each *Data-Field* field contains information about a single unique module, the address of which is coded in the respective *A* subfield, under the *Address* field.

5.2.2.ii Implementation

On the PC, which runs the high-level controller and the GA code, an independent thread is implemented for continuous communication between the robot and the PC. This thread in turn executes two subroutines continuously. One subroutine listens to the incoming port (Rx) and decodes all incoming message frames. The other subroutine checks desired actuator positions of all the modules, and packages together into a single message frame, values of those modules whose desired positions (as outputted by the trajectory generator subroutine) have been updated, and sends out the message frame.

On the *SkyMega* controller board, two subroutine are executed continuously as well. One

subroutine continuously checks for any incoming data, decodes received frames, and serves the message accordingly. The second subroutine reads potentiometer of all connected servomotors, records time of read, estimates the module's position based on the value read, packages all the data into a single message frame, and sends it out. The second subroutine is only executed if the *Stream* flag (A boolean type variable) is set to *True*. This variable is set to *False* by default at program initialization, but can be changed to *True* or *False* by sending *Start position stream* or *Stop position stream* messages from the PC, respectively.

With this implementation, and for a configuration with four modules, time from reading potentiometer values to sending a message frame, on average takes . So a position message frequency of . Message frame rate varies based on message frame type.

5.3 Evolving locomotion

For evolving a stable gait for the real *Y-bot* modular robotic configuration through Embodied Evolution (EE), Sinusoidal controller, as explained in subsection 3.2.1, page 49, is used. In this control model, each modules is controlled independently by Equation 5.26, where parameters , and determine amplitude, offset and phase-shift in oscillation of the module's in the configuration. Relative difference in phase-shift value among modules determine the emerged gait in the robotic configuration, while all the modules oscillate with a common frequency .

(5.26)

,

where is the total number of modules in the configuration, which is 4 for the case of *Y-bot* configuration, is the amplitude, is the frequency, is the phase and is the offset of the oscillator for the module in the configuration.

5.3.1 Evolutionary Algorithm (EA)

For parameter optimization through EA, a combination of Genetic Algorithm (GA) and Evolutionary Strategy (ES) has been implemented. GA is an iterative optimization technique based on (i) biological process of natural selection, the process that drives biological evolution, and (ii) genetics. Each parameter to be optimized is considered a gene, and the vector of parameters, a genome. In GA, a set of random parameter vectors are initialized, which is called the population. Each candidate solution of the population is evaluated and assigned a fitness value, based on some objective function of the optimization problem. Then, candidates are ranked and selected based on their fitness value. Best performing candidate solutions of each generation are selected and recombined with each other to produce offspring (candidates for the next generation), which are further randomly mutated. ES, on the other hand, skips the recombination part of GA,

but instead populates the next generation with the best solutions of the previous solution and mutated copies of the best solutions.

As part of this thesis, an EA combining aspects of GA and ES has been developed, in which population of the next generation is generated the following way,

1. Evaluate each individual of the population and assign a fitness value.
2. Rank individuals based on their fitness value (Equation 5.27).
3. Carry forward the top (*Elite*) of the best performing individuals of the current generation, to the next generation.
4. Add to the next generation, one mutated copy of each *Elite* individual. Mutation probability and mutation range are twice than normal in this case.
5. Based on the ranking, stochastically select individuals amounting to of the size of the population, using roulette-wheel selection method (subsubsection 5.3.1.i, page 126). Exclude the *Elites* from this selection.
6. Create a parent population by adding the *Elites* and the stochastically selected individuals.
7. Generate offspring by crossing parents in the parent population using intermediate-crossover method (subsubsection 5.3.1.ii, page 127). First parent for each crossover is selected in an orderly manner, while the second parent (partner) is selected stochastically from the parent population.
8. Generated offspring are mutated stochastically.

5.3.1.i Roulette-wheel selection

Fitness proportion selection method, also known as roulette-wheel selection method, is a method of selecting individuals in a population, to form the mating pool (parent population) for generating offspring of the next generation. In this method, each individual is assigned a rank ( ), which is based on the proportion of its fitness to the total fitness of the population, and is calculates as shown in Equation 5.27. The ranks are normalized, and sum to . Individuals forming the mating pool are then selected stochastically from the population, but with a selection probability of for selecting the individual. In this way, individuals with higher fitness have a higher probability of being selected, but individuals with a low fitness value still have a chance of being selected, but with a lower probability. This ensures maintaining variation in the population, which, to some extent avoids getting stuck in a local maximum.

——— (5.27)

,

where is the fitness value of the individual in the population, and is the population size.

5.3.1.ii Intermediate recombination

Intermediate recombination is a GA method for combining genomes of two parents to produce an offspring, where genes of the produced offspring lie somewhere in between or around those of the parents. This method is only applicable to genomes containing real-valued genes, and an offspring is produced as shown in Equation 5.28

(5.28)

,

where , and refers to the offspring and the two parent genomes respectively, is the gene in the genome of size , is the scaling factor, and is chosen at random uniformly for each gene.

Parameter determines how far from the vicinity of either of the two parents, in the genome space, the generated offspring might fall. If , then the produced offspring would lie in the region spanned by the two parents, and it is called standard intermediate recombination. A value of increases the boundary beyond the region spanned by the parents, in which the offspring might fall (Figure 5.13). In the case of standard recombination, since not all genes of the generated offspring lie on the border of the possible area, this area shrinks over generations, which in turn limits the range (variation) of newly created offspring. In this thesis, a value of is chosen, which ensures (statistically) that the available area for the offspring is same as the area spanned by the parents.

5.3.2 Evolution

For performing evolution on the real *Y-bot* modular robotic configuration, the robot is placed in the arena, connected to two cables. One of which is a 2-pin power supply cable, connecting the *SkyMega* board to a power supply unit, which powers both the controller board, as well as servomotors of connected modules. The other is a 4-pin serial communication cable, connecting *SkyMega* board on the robot, to the desktop personal computer (PC). High-level controller for generating actuator trajectory of modules, as well as EA code runs on the desktop PC, while the low-level controller for controlling and sensing actuator positions, reside on-board the robot.

For the *Y-bot* modular robotic configuration, control parameters to be optimized for evolving a stable gait consists of four amplitude parameters and , four offset parameters and , four phase-shift parameters and , and a common frequency parameter , totaling to 13 independent parameters. Starting from a randomly initialized set of 20 candidate solutions,

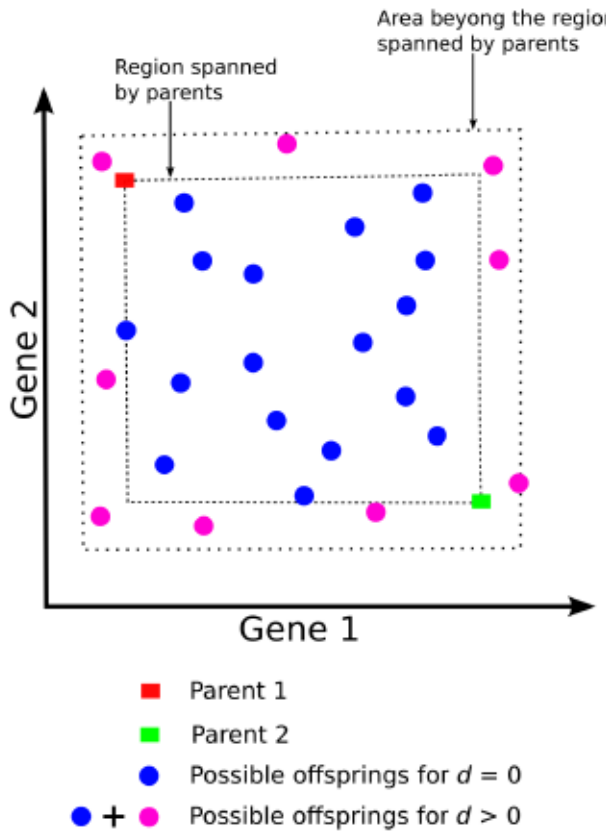


Figure 5.13: *Region, in genome space, spanned by parent and offspring genome with* $d \geq 0$.

where each solution is a vector of 13 parameters, each candidate solution is evaluated on the real robot for a period on $\tau = 25s$. At the end of the evaluation, fitness of the candidate solution is calculated as distance traveled by the robot, over the period of evaluation. Distance traveled is calculated with Equation 5.25, as the euclidean distance between start and the end positions of the robot, which are determined by the vision system as explained in subsection 5.2.1, page 116.

For the case of Embodied Evolution (EE), parameter range for optimization is set to a smaller range compared to simulation based evolution. Parameter range set for EE is as shown in Table 5.10. Reasons for the reduced range are: (i) to ensure stability of the gait and to avoid the robot flipping over during evolution, (ii) to avoid mechanical damage and over heating of module's servomotor.

Evolution process starts with the robot placed at the center of the arena, and then each candidate solution of the first generation is evaluated one at a time. Before each candidate solution is evaluated, the robot is first reset to its default state (i.e, $\theta_i = 0\,^\circ, \forall i \in \{1, \cdots, M\}$). Then, position of the robot in the arena before the evaluation — $\mathbf{r}^{(b)} = \begin{bmatrix} x^{(b)} & y^{(b)} & h \end{bmatrix}^\top$ (i.e, x and y position of the robot in the two-dimensional (2D) plane) — is determined based on

Parameter	*Min.*	*Max.*

Table 5.10: *Range of sinusoidal controller parameters for EE.*

the colored-marker on the robot. After these two steps, control signal for all the modules in the configuration are generated using Equation 5.26, with genome of the evaluated candidate solution as control parameters, for , with and . At the end of the evaluation, the robot is once again reset to its default state, and its position in the arena after the evaluation — — is determined. Based on and , euclidean distance (Equation 5.25) traveled by the robot during evaluation is calculated, and a fitness value, as calculated with Equation 5.29, is assigned to the evaluated candidate solution.

– (5.29)

Similarly, each candidate solution of the population is evaluated and assigned a fitness value based on its performance. Field of View (FOV) of the overhead camera spans an area of , so the colored-marker on the robot has to be within this area, before and after each evaluation. The EA framework developed in this work facilitates pausing evolution during and in between evaluations. This is necessary to move the robot back to the center of the arena, if and when the robot moves out of the arena during an evaluation, or ends up close to the border of the arena at the end of an evaluation. The user can pause the evaluation through a keyboard interrupt, then manually move the robot back to the center of the arena, and then resume evolution through another keyboard interrupt. In those cases when the robot moves out of the arena during an evaluation, the user can pause the evolution, move the robot back to the center of the arena, resume the evolution and then re-evaluate the interrupted candidate solution afresh. The developed EA framework facilitates going back to any previously evaluated candidate of the current generation, for re-evaluating it. Similarly, it is also possible to skip over evaluating candidate solutions, in which case, a fitness value of is assigned to such candidates.

After completing each evaluation, state of the evolution, including all EA parameter values assigned, candidates of all the populations generated so far, and their respective fitness values, are all saved in a data file on the desktop PC running the EA. Functionality to resume evolution from the last known state of the evolution, given the saved data file as input, has also been developed in this EA framework. This ensure that time and effort are not lost if evolution crashes for any reason. Also, when evolving with a large population size and/or with a long evaluation period, which would result in evolution running over several hours or days, it makes it possible to run the evolution process in multiple sessions.

Once the robot is placed in the arena, connected to a power source and host PC, and the

evolution process is started by the user, rest of the EE subsystems such as candidate solution evaluation, robot position estimation, fitness value calculation and resetting state of the robot, are all performed autonomously, without needing human intervention. But once the robot goes close to or beyond the FOV boundary of the arena, the user need to intervene to replace the robot back to the center of the arena. So, the developed EE framework can be called as semi-automated.

Fitness graph of gait evolution on the *Y-bot* configuration is as shown in Figure 5.14. The gait is evolved for 15 generations, and evolution takes close to seven hours to complete. Rest of the parameters of the Evolutionary Algorithm (EA) used for evolving the gait are as provided in Table 5.11. None of the candidate solutions were skipped over during evolution, and around 15 candidates solutions were re-evaluated due to the robot crossing over the boundary during evaluation. Evaluation of the initial few generations was much fast than later generations, as many individuals in the later generations moved close to the boundary at the ending of the evaluation, leading to operator intervention, and in turn leading to longer periods of intervals between evaluations. A video summarizing EE process for evolving gait on the *Y-bot* configuration is available on `https://youtu.be/Qzy9vm9NSW4`.

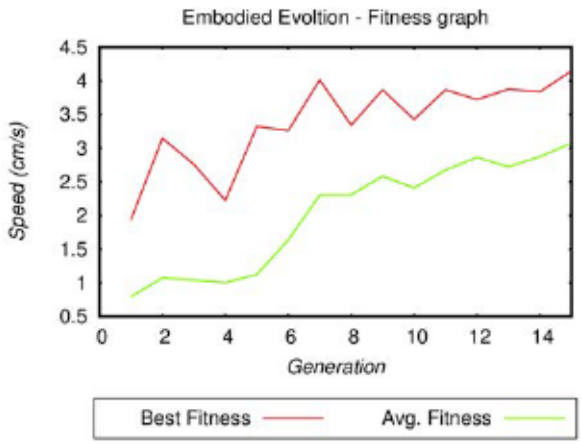


Figure 5.14: *Graph showing fitness value of best individual and average fitness value of the population during EE.*

5.3.3 Evaluation

Modules in the *Y-bot* configuration, when evaluated with the best evolved controller of the final generation, oscillate in the following range,

:

:

:

Parameters	*Value*
Population size	20
Genome size	13
Evaluation period	
Evolution length	15 generations
Crossover rate	
Elite population size	
Mutation rate	$\frac{1}{\text{Size of genome}}$
Mutation range	

Table 5.11: *EA parameter values used for EE.*

:

Evolved control parameters are as presented in Table 5.12. The produced gait in *Y-bot* is very similar to the gait that emerge in the simulated version of this configuration. In the emerged gait, modules and oscillate in phase[footnote 3], while there exists a phase-difference of between the modules and the module, and a phase-difference of between the module and the module. This phase-relation among modules result in the typical caterpillar gait, propelling the robot in the direction of the module. In 10 evaluations on each, average speed of locomotion achieved with this controller is , with a standard deviation (SD) of . This is significantly slower than the speed achieved by the simulated version of this configuration with the same controller. Reasons for reduced performance include: (i) the range of control parameters set for Embodied Evolution (EE) are much narrower compared to simulated evolution, (ii) surface of the arena, on which the robot crawls, has lower friction compared to simulation, resulting in slippage when the robot tries to move, and (iii) reduced population size and generation of evolution, compared to simulated evolution.

Parameter	*Module*			

Table 5.12: *Sinusoidal controller parameters of the best performing individual for Y-bot configuration, optimized through EE.*

Summary of the chapter

In this chapter a framework for evolving locomotion on a physical modular robotic configuration in real-world is presented. A real *Y-bot* configuration is constructed using *Y1* robot modules, which are fabricated and assembled as part of this thesis. A *SkyMega* controller board is used on-board the robot, for low-level control. Servomotors of the modules in the configuration are hacked to read potentiometer values, which are used for estimating actuator position. A linear model is fitted to potentiometer data, through linear regression, for estimating actuator position. A Kalman Filter (KF) is implemented for module's actuator state estimation, based on sinusoidal model.

A two-dimension (2D) vision system is developed, as part of EE experiment setup. The vision system consists of a Red Green Blue (RGB) webcam and a colored-marker attached to the robot. An algorithm for detecting and estimating position of the colored-marker, in the image space, is developed. Position of the robot in the two-dimensional (2D) plane is calculated based on: (i) estimated pixel position of the marker, (ii) fixed distance between the webcam and the target image-plane, and (iii) camera intrinsics.

A communication protocol for establishing communication between the host personal computer (PC) — on which the high-level controller, and the Evolutionary Algorithm (EA) are implemented — and the *SkyMega* controller board, which is on-board the robot, is developed. This communication protocol is high-level and application specific, featuring two segments — metadata and data — with multiple fields and subfields within.

An EA, combining features of Genetic Algorithm (GA) and Evolutionary Strategy (ES), is implemented, using which, gait on a real *Y-bot* modular robotic configuration is evolved. The evolved gait is stable but slower compared to the gait evolved in simulation, which is due to constraints on controller parameters and EA hyper-parameters, and due to low friction of the locomotion surface.

CHAPTER 6

Learning Locomotion

Introduction

In [Pfeifer and Bongard, 2006], the authors present three time-scale perspectives towards building an intelligent agent: (i) “Here-and-Now”, (ii) Ontogenetic and (iii) Phylogenetic. The “Here-and-Now” perspective refers to behavioral mechanism of an agent. The Ontogenetic perspective refers to lifelong development of an agent, and the Phylogenetic perspective refers to evolution over several generations. The “Here-and-Now” is the short-term perspective, where the agent follows some set rules to react to stimuli. Ontogenetic is the intermediate to long-term perspective, where the agent learns how to react to stimuli. Phylogenetic is the very long-term perspective, where behaviors in an agent are evolved over several generations.

In this thesis so far, locomotion in Modular Robot (MR) and humanoid robot have been approached from a Phylogenetic or very long-term perspective, where gaits are evolved in the robot offline through EA. In this chapter, locomotion in MRs from a Ontogenetic perspective is presented, where gaits in modular robotic configurations are learnt in an online fashion.

6.1 Introduction to Reinforcement Learning

Reinforcement Learning (RL) is an area under Machine Learning (ML) that focus on learning optimal control policy (i.e, mapping situations to actions) by interacting with the environment, so as to maximize cumulative reward. In Supervised Learning (SL),which is another and a

more widely researched field in ML, a knowledgeable external-supervisor provides a set of labeled training examples to the learning agent, based on which the agent should learn to predict labels of previously unseen examples. Here, a training example can be seen as a situation in the environment (state), and its label as an appropriate action, or a class of action, to this situation. So, objective of the agent in SL is to learn to generalize previously unseen states of the environment, without having the notion of reward. On the other hand, a RL agent is not provided with a set of training data, instead learns how to react to situations, by interacting with the environment and learning through trial-and-error. As a consequence of its action, a RL agent is either rewarded or punished by the environment. So, the agent should in essence have the ability to sense the state of the environment, and have a goal or goals related to the state of the environment. A reinforcement signal, in the form of reward or punishment an agent receives, is a numerical value provided by the environment as a consequence of its action. An action not only influences the immediate reward, but the next state (input), and with that, all future rewards. So, the objective of a learning agent is not to just learn an action that maximize its immediate reward, but to learn actions that maximizes total reward it receives over a long run.

In RL, a *policy* is a mapping from perceived states of the environment to actions to be taken by the agent when in that state. It defines the way an agent should behave to perceived states. At each time step, the environment sends RL agent a single number as a reinforcement signal. A high number indicates reward, where as a low or negative number indicates punishment. The goal of a RL agent is to maximize total reward it receives over the long run. Rewards are stochastic, and are based only on the current state and the action taken. An agent cannot alter the source that generates this reward, but can only change its own action so as to get a different reward, and with that maybe a different resulting state.

A reward signal can only indicates what is good in the immediate sense, but cannot tell what is good in the long term. A *value function*, on the other hand, is the total reward an agent can accumulate starting from that state. For example, a state might always result in a low reward signal, but still have a high *value function*, if it is regularly followed by states that generate high reward signals. The opposite could be true as well. Although the goal of an agent is to maximize accumulated reward, the next action is chosen based on *value function* of states, and not their expected reward, as this would result in maximizing future rewards. Reward signal of a state can be easily obtained from the environment directly, but it is not so straightforward in the case of *value functions*. It has to be estimated and re-estimated iteratively through observations the agent makes over its lifetime.

In RL, a *model* of the environment is something that allows inference to be made about how the environment would behave. It mimics the behavior of the environment, for example, give a state and an action, the *model* could predict the resulting next state. A model of the environment might not always be explicitly available, and it is something that the agent learns implicitly through trial-and error. This kind of learning is called *model-free* learning, which is the focus of this work.

6.2 Markov Decision Process (MDP)

From [Sutton and Barto, 1998], a MDP is a tuple , where is a set of states, with ; is a set of actions; is a probability distribution over the next sates if action is taken in state , and therefore is the probability of transitioning from state to state when action is taken; is the reward the agent receives when the sequence occurs; is the discount factor that balances out the importance between immediate reward verses future rewards; and is the distribution over the initial state . and , together constitute the model of an MDP.

A *trajectory* is a sequence such as , where the initial state , and each state following is generated by the state transition distribution such that, . Each action in the trajectory is based on some policy that maps each state to an action. Reward in the trajectory is nothing but .

For a given policy , *value function* of a state is,

(6.1)

which is the expected sum of discounted rewards of an agent, starting at state and following policy from there on. Similarly, a *state-action value function* of a state-action pair is the expected sum of discounted rewards of an agent, starting at state , taking action , and then following policy from there on,

(6.2)

So, the goal of solving an MDP is to, given a model of an MDP, find an optimal policy ( ), that maximizes the expected cumulative discounted reward of all the states Equation 6.3.

(6.3)

Then the optimal *value function* would be,

(6.4)

and the optimal *state-action value function* would be,

(6.5)

It is well known that the optimal *value function* satisfies the Bellman Optimality Equation:

(6.6)

6.2.1 Solving Markov Decision Process (MDP)

Given a perfect model of an MDP (and) with a finite state space, optimal policies can be computed in a iterative manner, through a continuous 2-step process of (i) policy evaluation and (ii) policy improvement. In the policy evaluation step, *value functions* of all or some states are estimated, given a fixed policy, and in the policy improvement step, the previous policy is improved based on the values obtained from the policy evaluation step. An illustration of this process is as shown in Figure 6.1.

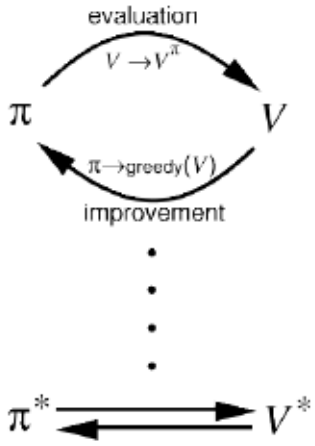


Figure 6.1: *Policy Iteration. Source: [Sutton and Barto, 1998]*

6.2.1.i Policy Evaluation

As explained in [Sutton and Barto, 1998], the *value function* (Equation 6.1) can be derived recursively as:

(6.7)

The vector form of which is,

(6.8)

where is a vector of state values, is a matrix encoding transition probability of policy , with , is a vector of rewards, and .

Since the state space is assumed to be finite, state values can be calculated for all the states by solving linear equations:

(6.9)

where is the identity matrix.

6.2.1.ii Policy Improvement

Based on the calculated values , a policy can be improved by selecting the action that is "greedy" with respect to the expected return:

(6.10)

In case of more than one best available action for a state, ties are broken by selecting an action uniformly randomly, amongst the best actions for the state.

6.2.2 Policy Iteration

Once a policy has been improved using state values , yielding a better policy , new state values can be computed based on the new and improved policy , leading to

even better policy , and so on, until an optimal policy and optimal (or close to optimal) state values are found. So, putting together the policy evaluation Equation 6.7 and policy improvementEquation 6.10 together, results in the *policy iteration* algorithm, which is as shown in algorithm 2.

Result: Improved policy
1 Initialize and , arbitrarily ;
2 changed **True** ;
3 **while** *changed* **do**
4 **do**
5 ;
6 **for** **do**
7 ;
8 ;
9 ;
10 **end**
11 **while** ;
12 **for** **do**
13 ;
14 **end**
15 changed ;
16 ;
17 **end**
18 ;

Algorithm 2: Policy iteration algorithm

The input to the algorithm is the model of the MDP (and), and parameter . In line 1 policy and its corresponding state values are initialized. The initial policy can be chosen to be uniformly random from , which is the set of actions possible for state , while the initial state values are set to . Policy evaluation (line 4-line 11) is itself an iterative process, which is executed for as long as the largest change in state values ( ) is above a certain set threshold of . Smaller the value of , more accurate the estimated state values would be, but at a higher computational cost. It is not necessary to estimate the exact value of each state, for a given policy, before proceeding to policy improvement step (line 13), as long as the estimated values are close enough to optimal values. Also, it is important to note that policy evaluation starts with state value estimates of the previous policy, and results in quick convergence to new state values, as they do not change much from policy to policy. The algorithm ends when there is no more change in policy (line 15).

6.2.3 Value Iteration

One of the drawbacks of the policy iteration algorithm is, that each policy evaluation step needs several sweeps over the state space, before proceeding to policy improvement. An alternate to this is to update the policy after every single "Bellman Backup" (line 8 in algorithm 2), so that each backup has access to the best possible action, rather than a fixed action. This is realized by implicitly combining Equation 6.7 and Equation 6.10 in Equation 6.11.

(6.11)

The value iteration algorithm is as presented in algorithm 3.

```
   Result: Improved policy
 1 Initialize              arbitrarily        ;
 2 changed    True ;
 3 do
 4 |          ;
 5 |   for       do
 6 |   |              ;
 7 |   |                                          ;
 8 |   |                                             ;
 9 |   |                                 ;
10 |   end
11 while       ;
12          ;
```

Algorithm 3: Value iteration algorithm

6.3 Temporal Difference Learning

In practical domains, such as in the case of this thesis of learning locomotion in modular robots, model of the Markov Decision Process (MDP) (and) is unknown, and random access to arbitrary states in the state space is not available either. Instead, the learning agent has to interact with the environment, generating trajectories, and learn from them. Temporal-Difference (TD) learning is a Reinforcement Learning (RL) technique, that learns by (i) sampling the environment according to some policy, and (ii) by approximating its current estimate of values based on previous estimates (which is known as "Bootstrapping"). **Q-Learning** is a model-free TD learning algorithm for learning optimal policies. The method works by first learning *state-action value function* , and then using these values to construct an optimal policy. State-action value of a state gives the expected total discounted reward to be earned by the

agent, by taking action from state , and then following the optimal policy thereafter. With learnt state-action values, an optimal policy can then be constructed by simply choosing the action with the highest value for each state. State-action value update rule for Q-Learning is,

(6.12)

where is the learning rate. The above Q-Learning update rule works by assuming the old value of a state-action pair, and then making the correction based on new information. New information comes about in the form of reward and new state , when the learning agent takes action from state . The complete Q-Learning algorithm is as shown in algorithm 4,

Result: Optimal policy
1 Initialize arbitrarily and ;
2 **for** *each episode* **do**
3 Initialize state ;
4 **repeat**
5 Choose action from state , using a policy derived from Q (e.g.: -greedy) ;
6 Take action , observe reward and the resulting next state ;
7 ;
8 ;
9 **until** *end of episode*;
10 **end**
11 greedy w.r.t ;

Algorithm 4: Q-Learning algorithm

A trade-off between exploration and exploitation is an important challenge in RL. To obtain a lot of rewards, a learning agent has to select from each state, a previously tried action which has the maximum expected total discounted reward. But to discover such actions, the agent needs to keep exploring and tryout previously unexplored actions. -greedy is one such method in which the agents chooses the greedy action most of the time, with probability of , and chooses an action at uniformly random, with probability (Equation 6.13).

with probability
with probability (6.13)

The update rule of the algorithm (line 7) is based on the agent following an optimal policy from the resulting state onward (i.e,). But this is not necessarily the case, because the true policy followed by the agent is not optimal all the time (e.g.:). So, this kind of learning is called *off-policy* learning, because the policy followed by the agent and the one used by the update rule are not necessarily the same for each update.

6.4 Locomotion through Q-Learning

Considering a modular robotic configuration as the learning agent, for the purpose of learning an optimal gait through RL, it is necessary to first define the state space, action space and the reward function.

6.4.1 State space

In the context of learning locomotion in modular robots through Reinforcement Learning (RL), a state can be represented as a vector ( ) of discrete actuator positions of modules in a configuration with modules. That is, , where is the actuator position of the module in the configuration. Then , the state space of the model, would be all possible discrete actuator positions of all the modules in the configuration. For representing a state, actuator position of a module is discretized such that , with a constant step-size , where and are the lower and upper bounds of the module's actuator. That is, . So, each module in the configuration can be in any one of (Equation 6.14) distinct positions, which mean that a robot with modules can be in any one of possible states, which is the size of the state space.

———— (6.14)

A module with actuator position , can be categorized to a discrete non-corner position , where , if ———— ————. Similarly, module can be categorized to discrete corner positions and , if ———, or if ——— respectively.

For example, if , and , then for module , and . Now, if module with actuator position , then module it would be discretized to state , or if , then it would discretized to state .

6.4.2 Action space

Each action can also be represented as a vector ( ), such that , the component of the vector, is the action for the module in the configuration. For representing an action, actions are discretized such that each action component can be one of three possible values, . So, a robot with modules would have an action space of actions. Vectorized representation of states and actions facilitates easy formulation of state-transitions in the form of simple vector addition, as shown in Equation 6.15.

(6.15)

where is the expected next state, with some unknown transition probability , when action is taken from state .

Consider an example case of a two-module robotic configuration, with modules and . Action space for this robot can represented in matrix form as,

where each row is a distinct action , and each column is the action for the module in the configuration. If the robot is in some state , where , taking action for example, would transition the robot to expected next state , with some unknown transition probability .

In the example case with two modules, size of the action space actions, but not all 9 actions are applicable in each state . For example, in the state , actions where or are invalid. Formally this can be defined as,

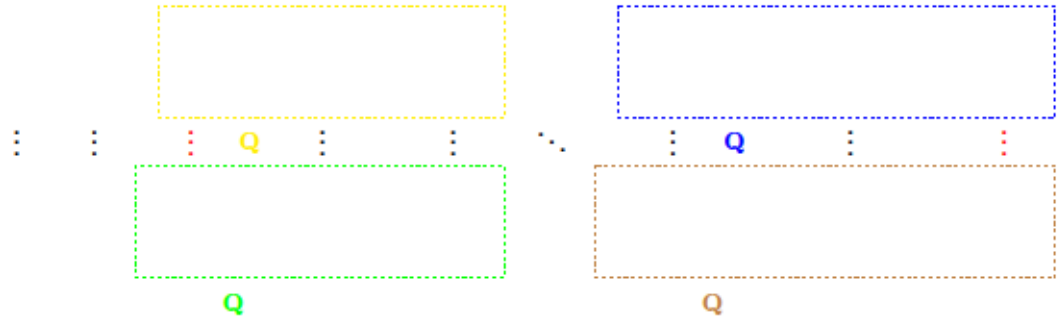
if (6.16)

where , is a set of all valid actions in state .

6.4.3 State-action space

State-action space can be defined as, . Size of the state-action space would then be , i.e, sum of action space size, of all the state .

For the case of two-module configuration, state-action pairs can be, from a geometric perspective, arranged into a square matrix as,



where , , and each element of the matrix represents a unique state-action pair. Contiguous sub-matrices of size within the matrix (like the ones enclosed in dotted colored boxes in the above matrix), represent state-action pairs of a single state , while each element within such a sub-matrix represent a unique action. For example, in the above matrix, elements within the yellow box ( ) form a sub-matrix of state-action pairs for state . Similarly, elements withing the green box ( ) constitute a sub-matrix of state-action pairs for state . So, there are sub-matrices ( ) within such a matrix. Definition of some of the elements of this matrix are as follows,

, where and

, where and

, where and

, where and

, where and

, where and

With this representation of state-action pairs, it is then possible to visualize state-transitions as moving from one sub-matrix at time to an adjacent sub-matrix at time . If, for example, the robot is in state — which would be from the sub-matrix perspective — and take action (i.e,), then the robot would most likely transition to the state whose sub-matrix is one-step to the right (i.e,). Similarly, from the same state, taking action (i.e,) would most likely transition the robot to the state whose sub-matrix is one-step below and one-step right to the current sub-matrix (i.e,).

Given the geometric perspective of state-transitions, invalid state-actions are the ones that lie on the boundary of the two-dimensional (2D) matrix. That is,
, which are the elements marked in red, in the above matrix. Total number of invalid state-action pairs can then be calculated as,

(6.17)

That is, elements on each of the four edges of the 2D matrix, plus four corner elements.

For a three-module configuration, state-action pairs can be represented as a three-dimensional (3D) matrix (a cube). Invalid state-action pairs would then lie on the surface of the cube, which can be calculated as,

(6.18)

That is, elements that lie on each of the six sides of the cube, elements on the 12 edges of the cube, plus the eight corner elements.

In general, for a configuration with modules, state-action pairs can be represented as a M-dimensional hypercube. Invalid state-action pairs would then lie on the surface of this M-dimensional hypercube, which can be calculated as,

(6.19)

(6.20)

where refers to the number of -dimensional hypercubes on the boundary of a -dimensional hypercube.

So, size of the state-action space, given state space , action space , and invalid state-action pairs , would simply be,

(6.21)

6.4.4 Reward

For locomotion through evolution, which has been explored in previous chapters of this thesis, speed of locomotion is used as the objective function for evaluating individual candidate solutions. Locomotion speed is calculated as total distance traveled by the robot, over the time of evaluation. Distance traveled is calculated as the euclidean distance between start and end position of the robot, during evaluation. In terms of Reinforcement Learning (RL), this can be seen as reward from the environment at the end of the learning period. But for Q-learning, a reward signal from the environment is needed at every step (or at least every few steps), for value iteration, and so the final speed of locomotion cannot be used as reward signal as it is. Even if speed of locomotion is calculated at every step of the learning period, as distance traveled over the duration of the step, this quantity lacks a sign, and so a robot that moves one unit forward has the same reward when it moves one unit in the opposite direction. This could lead to a policy which makes the robot sway back and forth rapidly, instead of a policy that makes the robot move consistently in a single direction.

Instead, velocity of locomotion makes for a good reward signal for learning locomotion through RL. Given and , global position and orientation of the robot before taking an action, and and , global position and orientation of the robot after taking the action, velocity of the robot in both and axis can then be calculated as,

—

— (6.22)

where — is the velocity vector, is the rotation matrix,

is the inverse of the euclidean transformation matrix .

Velocity in the -axis ( ) is used for calculating reward signal, as this pertains to the velocity in the forward/backward directions of the robot. Velocity calculated per step of learning is in the range . Reward based on velocity is calculated as,

(6.23)

which increases the reward exponentially for positive velocity values, while bounding the reward between when .

When the robot enter an exit state by flipping over, then the action leading to the exit state is penalized by setting reward .

6.4.5 Algorithm

Finally, for learning the gait itself, an optimal policy that maximizes accumulated discounted rewards is learnt through Q-Learning. The robot is setup in the simulation environment and it follows algorithm 5, which is the standard Q-Learning algorithm, modified for the case of learning locomotion in modular robots.

Result: Optimal gait in the form of policy
Initialize ;
;
repeat
  Initialize state to a random state ;
  do
    Choose action from state , using a policy -greedy ;
    Take action , and wait until **or** ;
    Observe reward , resulting next state , and ;
    if *robot upright* **then**
      a ;
      ;
    else
      ;
    end
  while *robot upright*;
until *end of learning period* ;
greedy w.r.t ;

Algorithm 5: Q-Learning algorithm for learning locomotion in Modular Robot (MR).

The algorithm begins by initializing state-action values to a positive value . This is an exploration technique that encourages the agent to explore during the initial state of learning. As the agent tries a new state-action pair, updating its value and moving it closer towards the true state-action value, unexplored state-action pairs with a higher initial positive value become more attractive, encouraging the agent to explore previously explored states-action pairs. The learning algorithm is executed for a predefined learning period . Each new episode (outer loop of the algorithm) starts by setting the robot to a random initial position, chosen uniformly random from the state space .

At each step of learning (inner loop of the algorithm), the agent chooses an action following -greedy policy, then executes the chosen action by setting actuator positions to . After sending out the actuator commands, actuator positions of the robot are sourced until: (i) all the modules have reached their respective expected next position (i.e,), or (ii) until , where is the time since action was taken, and is a predefined time-limit. After executing the action, velocity, in the form of reward (Equation 6.23), and the resulting next state , are observed. Then, roll and pitch angles of the robot are calculated, to determine if

the robot is in upright position, or if it has flipped-over. If the robot is in upright position, then state-action value of the current state and action taken , is updated asper the standard Q-Learning update rule (line 10). On the other hand, if the robot has flipped-over, then the resulting state is considered the exit-state, and state-action value is updated based only on the negative reward (line 13).

6.5 Learning and Evaluation

Gait learning through Q-Learning is performed on the *Minibot*, *Tripod*, *Quadropod* and *Y-bot* modular robotic configuration, in the simulation environment. Resulting gait in each case is presented in the following subsections. Common Q-Learning parameters values used across all the configurations during learning, are as presented in Table 6.1.

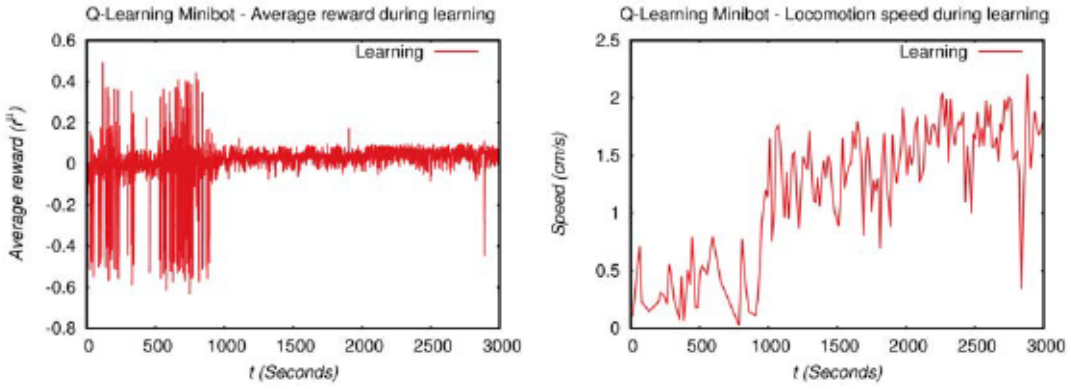
Parameter	*Value*

Table 6.1: *Q-Learning parameters.*

6.5.1 Minibot

As has been followed throughout this thesis, learning algorithm is first applied on the two-module *Minibot* configuration. The state, action and state-action space for this configuration consists 49 states, 9 actions and 361 state-action pairs, respectively. Figure 6.2a contains a plot of average reward ( ) received by the robot during the first 3000 seconds of the learning period, where average reward is calculated as a running average with Equation 6.24, and with a step-size parameter . Figure 6.2b contains a plot of locomotion speed of the robot, during the learning period, which is calculated as euclidean distance traveled by the robot, over time take, calculated approximately every 15 seconds. Learning period is set to 20,000 seconds, but as could be observed in the following graphs, a fairly stable gait emerged within the first 1000 seconds of learning, which is approximately 12,800 learning steps. During the initial period, that is , the robot flips over several times while exploring state-action space, and hence the high variation in average reward values during this period. Then, at around , the robot falls into a fairly stable gait. The same can be observed in the locomotion speed graph.

(6.24)



(a) *Average reward* **(b)** *Locomotion speed*

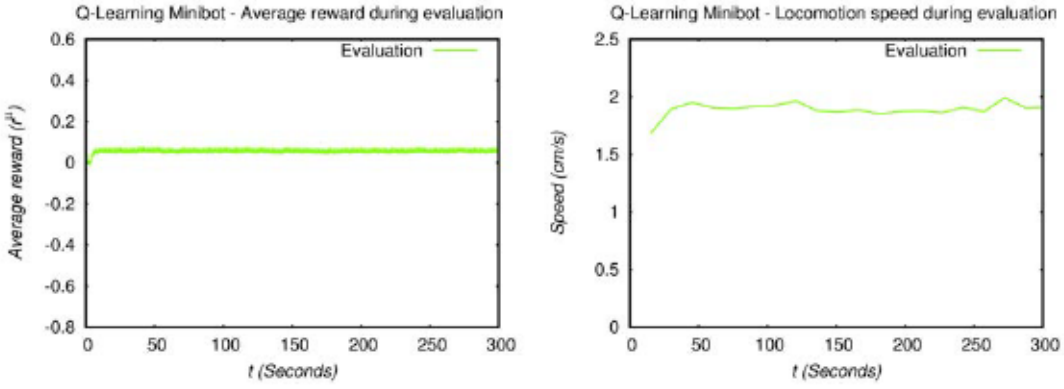
Figure 6.2: *Average reward and locomotion speed, during gait learning in the Minibot configuration.*

The learnt gait is evaluated for a period of 100 seconds, by following a greedy policy (Equation 6.25). Average reward (6.3a) and locomotion speed (6.3b) during evaluation are as plotted in Figure 6.3, where, range of -axis of the two graphs are the same as the respective graphs during learning, which are plotted in Figure 6.2. The learnt gait is very stable and consistent. Figure 6.4 contains reference trajectories, generated by the learnt optimal policy ( ), while Figure 6.5 contains actuator trajectories of *Minibot* modules, generated during evaluation, by following the reference trajectories.

following the learnt optimal policy greedily , while Figure 6.5 contains actuator trajectories of *Minibot* modules, generated by following reference trajectories.

(6.25)

In the generated gait, oscillation amplitude, offset, range and frequency of *Head* and *Tail* modules are as provided in Table 6.2. Modules oscillate with a negative phase-difference of between them, resulting in the robot moving in the direction of the *Tail* module. The reason for this is the way in which the robot is setup in the simulation environment, resulting in positive velocity in the direction of the *Tail* module. By negating the calculated velocity as , during learning period, the robot learns to crawl in the opposite direction. Locomotion speed of the learnt gait during evaluation is .



(a) *Average reward* **(b)** *Locomotion speed*

Figure 6.3: *Average reward and locomotion speed, during gait evaluation in the Minibot configuration.*

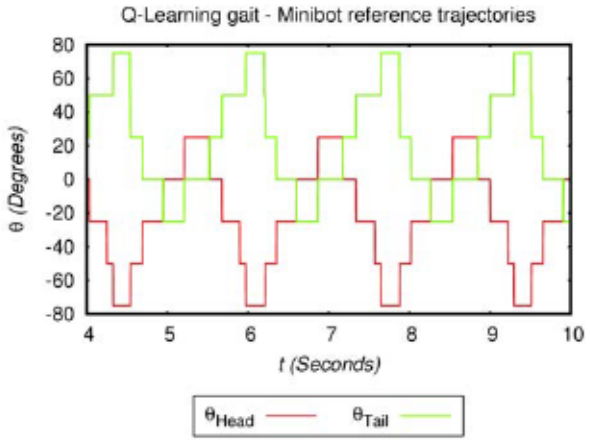


Figure 6.4: *Reference trajectories generated by the learnt optimal policy ($^+$), in the Minibot configuration.*

6.5.2 Tripod

State, action and state-action space for the *Tripod* configuration consists of 343 states, 27 actions and 6,859 state-action pairs. Learning period is set to 50,000 seconds, and the learnt policy is evaluated for a period of 100 seconds, post learning. Figure 6.6 contains actuator trajectories of *Tripod* modules during evaluation. Oscillation amplitude, offset, range and frequency of *Tripod* modules during evaluation are as presented in Table 6.3.

In the learnt gait, modules oscillate at a high frequency, but a shorter oscillation range. Mean and standard deviation (SD) of phase-difference between modules in the *Tripod* configuration, during evaluation, is as presented in Table 6.4. No two modules oscillate in phase, resulting in

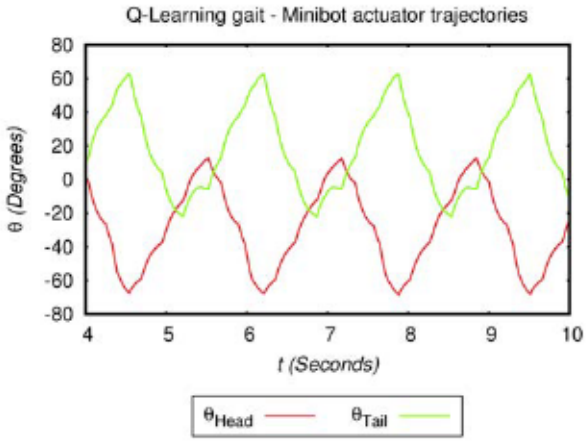


Figure 6.5: *Actuator trajectories of the Minibot modules, during learnt gait evaluation.*

Oscillation	Module	
	Head	*Tail*
Amplitude		
Offset		
Range		
Frequency		

Table 6.2: *Oscillation characteristics of Minibot modules, following the learnt policy.*

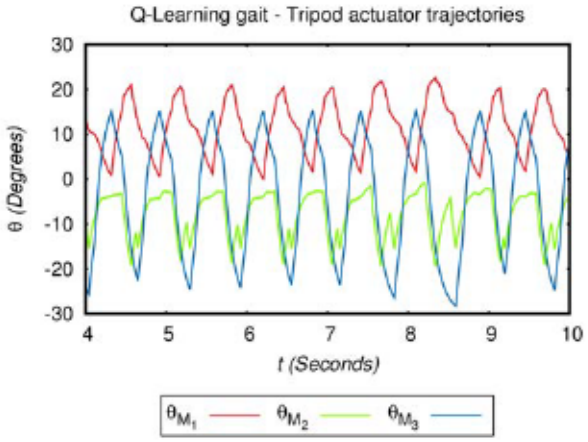


Figure 6.6: *Actuator trajectories of the Tripod modules, during learnt gait evaluation.*

a gait that makes the robot move on a circular trajectory. But consistency in phase-difference between modules (Figure 6.7) results in the robot moving consistently in the same direction. Speed of locomotion, which is calculated by integrating velocity values in the -axis, in this gait

Oscillation	Module		
Amplitude			
Offset			
Range			
Frequency			

Table 6.3: *Oscillation characteristics of Tripod modules, following the learnt policy.*

is

		Tripod Modules		
	Mean			
	SD			
	Mean			
	SD			
	Mean			
	SD			

Table 6.4: *Mean and SD of phase-difference between modules in the Tripod configuration, following the learnt policy.*

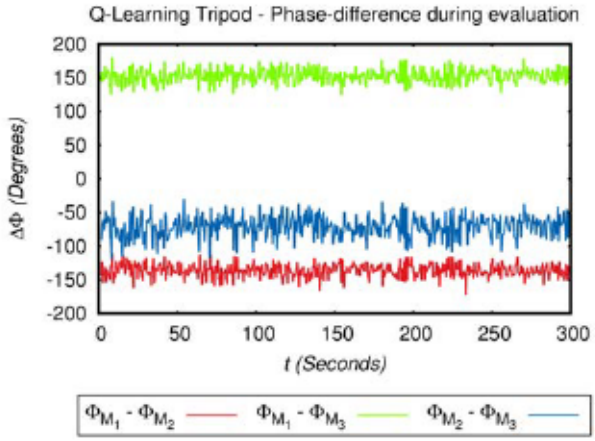


Figure 6.7: *Phase-difference between all modules pairs in the Tripod configuration, following the learnt policy.*

6.5.3 Quadropod

State, action and state-action space for the *Quadropod* configuration consists of 2401 states, 81 actions and 130,321 state-action pairs. Learning period is set to 100,000 seconds, and the resulting gait is evaluated for a period of 100 seconds, post learning. Figure 6.8 contains actuator trajectories of *Quadropod* modules during evaluation. Oscillation amplitude, offset, range and frequency of *Quadropod* modules during evaluation, are as presented in Table 6.5.

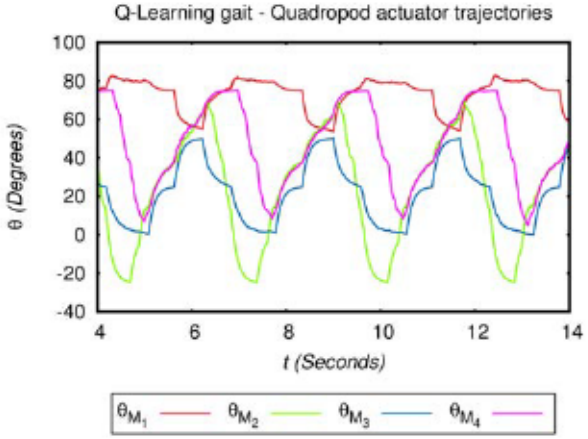


Figure 6.8: *Actuator trajectories of the Quadropod modules, during learnt gait evaluation.*

Oscillation	Module			
Amplitude				
Offset				
Range				
Frequency				

Table 6.5: *Oscillation characteristics of Quadropod modules, following the learnt policy.*

Mean and standard deviation (SD) of phase-difference between modules in the *Quadropod* configuration, during evaluation, is as presented in Table 6.6. In the learnt gait, modules oscillate at a very low frequency, but also with consistent phase-difference between oscillating modules (Figure 6.9), resulting in a consistent gait. Modules and oscillate in phase[footnote 3], but there exists consistent phase-difference between rest of the modules. The gait emerged makes the robot move consistently in the same direction, but on a circular trajectory, as shown in Figure 6.10. Speed of locomotion in this gait is .

		Quadropod Modules			
	Mean				
	SD				
	Mean				
	SD				
	Mean				
	SD				
	Mean				
	SD				

Table 6.6: *Mean and SD of phase-difference between modules in the Quadropod configuration, following the learnt policy.*

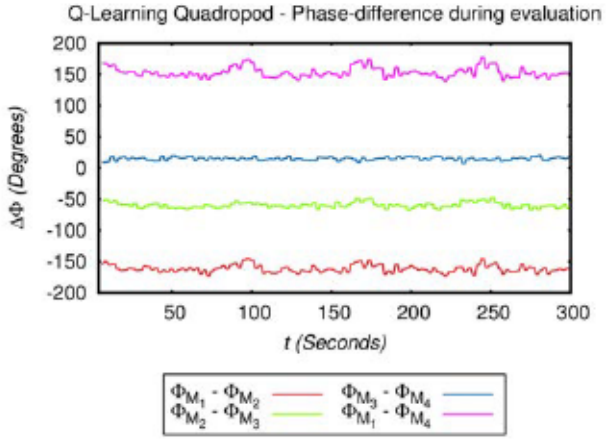


Figure 6.9: *Phase-difference between all modules pairs in the Quadropod configuration, following the learnt policy.*

6.5.4 Y-bot

Size of state, action and state-action space in the *Y-bot* configuration, is the same as that of *Quadropod* configuration, both of which have 4 module configurations. Learning period is once again set to 100,000 seconds. In *Minibot*, *Tripod* and *Quadropod* configurations, reward is calculated based only on velocity in the -axis. This is because, from the local coordinate frame of the robot, the -axis pertains to forward/backward directions of the robot, but the same does not hold true for *Y-bot* configuration. Figure 6.11 contains a bird's eye view of *Minibot*(6.11a) and *Y-bot* (6.11b) configurations, along with their respective local coordinate frames, embedded within, and the global coordinate frames at the top-right corner of the respective sub-figures. So, for the *Y-bot* configuration, by calculating reward based on either or alone would result in the robot learning gaits that would make it (roughly) rotate on its own axis, either in clock wise or anti-clock wise directions respectively. To learn a forward crawling gait, reward is calculated

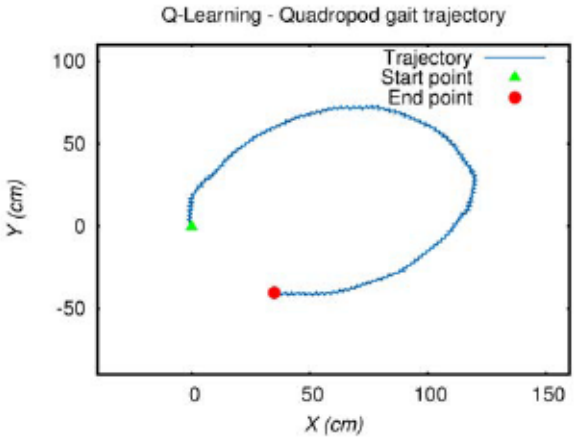


Figure 6.10: *Gait trajectory of the Quadropod configuration, following the learnt policy.*

based on velocity in both and axis (Equation 6.26), which makes the robot crawl in the direction indicated by black dotted arrow in sub-figure 6.11b.

(6.26)

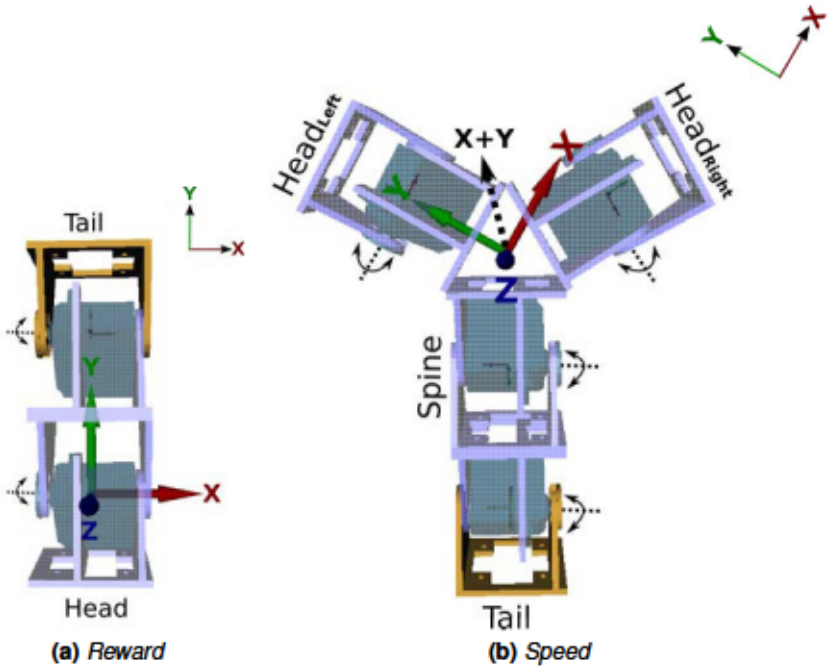
The learnt gait is evaluated for a period of 100 seconds. Figure 6.12 contains actuator trajectories of *Y-bot* modules during evaluation. Oscillation amplitude, offset, range and frequency of *Y-bot* modules during evaluation are as presented in Table 6.7.

Oscillation	Module			
Amplitude				
Offset				
Range				
Frequency				

Table 6.7: *Oscillation characteristics of Y-bot modules, following the learnt policy.*

In the resulting gait, modules oscillate with phase-difference as presented in Table 6.8. In this gait, no two modules oscillate in phase, and there exists positive phase-difference between *Head* and *Spine* modules, as well as between *Spine* and *Tail* modules, resulting a sine wave starting from the *Tail* module and moving in the direction of the *Head* module, which in turn propels the robot in the direction of the *Head* module. The learnt gait is once again very stable (Figure 6.13), with low standard deviation (SD) in phase-difference between modules, and results in the robot moving on a very circular trajectory, which is as seen in Figure 6.14. Locomotion speed in this gait is .

For the six-module *Lizard* configuration, state, action and state-action space would consists



(a) *Reward* **(b)** *Speed*

Figure 6.11: *Bird's eye view of Minibot and Y-bot configurations, and their respective local and global coordinate frames.*

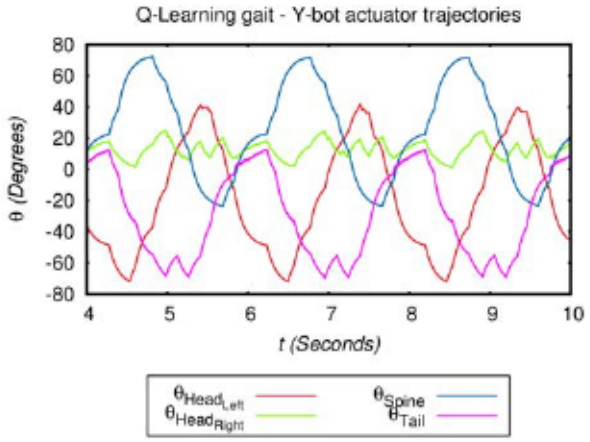


Figure 6.12: *Actuator trajectories of the Y-bot modules, during learnt gait evaluation.*

of 117649 states, 729 actions and 85,766,121 state-action pairs, which would make it infeasible to learn a gait, due to the sheer size of the state-action space.

		Y-bot Modules			
	Mean				
	SD				
	Mean				
	SD				
	Mean				
	SD				
	Mean				
	SD				

Table 6.8: *Mean and SD of phase-difference between modules in the Y-bot configuration, when evaluated with the learnt gait.*

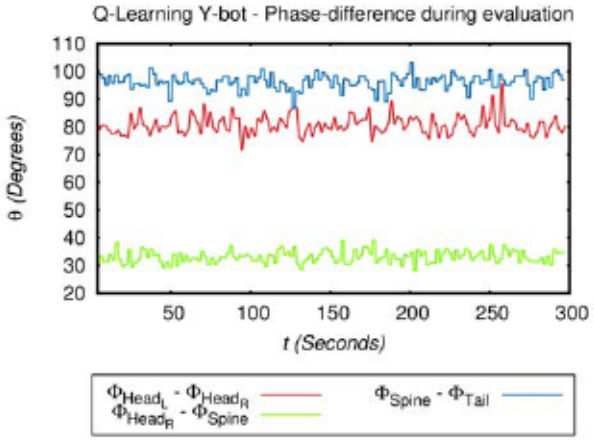


Figure 6.13: *Phase-difference between all modules pairs in the Y-bot configuration, following the learnt policy.*

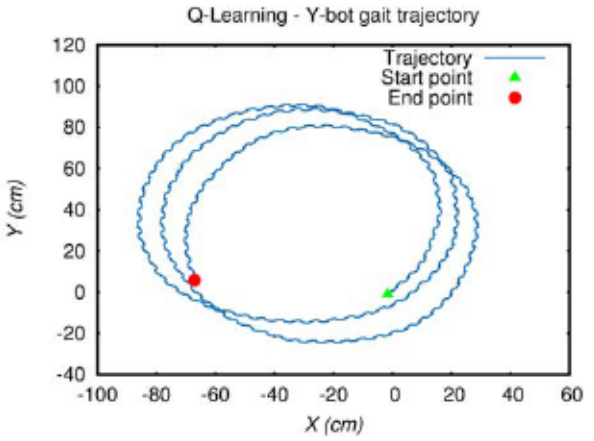


Figure 6.14: *Gait trajectory of the Y-bot configuration, following the learnt policy.*

Summary of the chapter

In this chapter a novel introduction to Reinforcement Learning (RL), and how it differs from other Machine Learning (ML) techniques, is provided. A Markov Decision Process (MDP), which is an extension of Markov chains, and used for making decisions in situations when outcomes are partly random and partly dependent on the controlling agent, are explained. *Value Iteration* and *Policy Iteration* are two popular model-based techniques for solving MDPs, given a complete model of the environment, by finding optimal policy ( ).

Q-Learning, which is a model-free, *off-policy*, Temporal-Difference (TD) learning algorithm, is explained. As part of this thesis, Q-Learning algorithm is adapted for learning locomotion in modular robotic configurations. State, action and state-action spaces, and reward function implemented for the purpose of learning locomotion are explained. Gaits are learnt in four different modular robotic configurations through Q-Learning, and evaluation results post learning are presented.

CHAPTER 7

Conclusions

Locomotion is an important feature for a robot in most application domains. Compared to wheel-based locomotion, legged locomotion gives a robot the ability to navigate rugged and unstructured environment. Limbless gaits give small-sized robots the ability to navigate narrow and inaccessible areas. This PhD thesis tackles the problem of providing locomotion for legged and limbless robots through: (i) Morphology, (ii) Evolution, and (iii) Learning. A wide variety of locomotion controllers for Modular Robot (MR)s, and for a humanoid robot has been developed in this thesis. Five different two-dimension (2D) modular robotic configurations, a simulated Humanoid for Open Architecture Platform (HOAP-3) robot, and a real modular robotic configuration has been constructed and used as target platforms for locomotion experiments, in this thesis.

As a first step, sinusoidal oscillators, which are commonly used as locomotion controller in MRs, is implemented. Gaits on all the modular robotic configurations are evolved by optimizing Sinusoidal controller parameters through Evolutionary Algorithm (EA), and evaluated for benchmarking. Fastest gait in four out of five configurations is achieved with this controller. Sinusoidal controller is distributed and scalable, but it is heterogeneous and cannot adapt to changes in configuration.

A second periodic function controller, based on Fourier series, is developed as part of this thesis. This controller can produce complex trajectories, which are essential for multiple Degree of Freedom (DOF) legged locomotion. Two variants of this controller are tested on different modular robotic configurations, and the fastest gait in the *Minibot* configuration is achieved with this controller. Control parameters are optimized through EA, and in the second version of this controller, generated trajectories are non-sinusoidal. This controller is also distributed and

scalable, but heterogeneous and cannot adapt to changes in configuration. It is parameter heavy, making it undesirable for producing creeping/crawling gaits in two-dimensional (2D) modular robotic configurations.

One of the objectives of this thesis is to study the relationship that exist between robot morphology and the gait that emerges in it, and use the results for developing locomotion controllers that are morphology-dependent. To this end, forces that exist between module in a configuration, as a result of interaction between physically connected modules, and between the modules and their environment, are measured and quantified. This phenomenon is termed Intra-Configuration Force (ICF), based on which two different morphology-dependent locomotion controllers for MRs are developed.

The first controller is Artificial Neural Network (ANN) based, which features a neural-oscillator mechanism for generating oscillatory signals for robot modules. This controller is distributed, homogeneous, and uncoupled. Control parameters are optimized through EA for learning locomotion in modular robotic configurations. All the modules in a configuration would have the same controllers, start with the same initial conditions, but emerge to act differently for producing a stable gait. In this control model, although modules do not communicate with each other explicitly, coordination among them for producing a gait, comes about as a function of ICF that exist between modules in the configuration. Because of the homogeneity of the controller, a controller optimized for one configuration can be evaluated on another configuration of any size. So, controllers optimized on each of the five configurations are cross-evaluated on rest of the four configurations. 15 successful gaits are produced out of 20 total cross-evaluations, further proving a strong link between the morphology (body) and the controller (brain) of the robot.

The second morphology based controller developed in this thesis is inverse sinusoidal function based. As part of which, a general purpose inverse sinusoidal function is developed. Unlike the Neural-oscillator controller, Inverse sinusoidal controller is trajectory based. That is, in the Neural-oscillator controller, only crest and trough points of oscillation, per cycle of oscillation, are provided. But with the Inverse sinusoidal controller, a continuous and smooth trajectory is produced by the controller for the module's actuator to follow. Inverse sinusoidal controller features a sinusoidal oscillator at its core, and so combines the smooth trajectory feature of a Sinusoidal controller, and the morphology-dependent feature of the Neural-oscillator controller. EA is used for evolving gaits in MRs, and the evolved gaits are smooth, stable and morphology-dependent. This controller is also distributed, scalable, homogeneous and uncoupled.

For bipedal locomotion in a humanoid robot, complex trajectories are needed. One such set of trajectories, available in the literature, is developed based on cart-table method, and has been successfully evaluated on the HOAP-3 robot. This is a model-based method of producing locomotion, where all aspect of the target robot needs to be explicitly modeled into the trajectory generator. In this thesis, a model-free, feature based linear periodic function for generating trajectories is developed. The periodic function consists of features such as symmetry, duality, signal-width, skewness and squareness, which dictates the shape of the generated trajectory. The periodic function is conceived by adding a sawtooth wave to a reverse sawtooth wave, and

then by introducing one feature at a time. As a first step, linear approximates of cart-table method generated trajectories are modeled, and evaluated on the simulated HOAP-3 robot. Then, a walking gait is evolved afresh by optimizing linear periodic function parameter, in a model-free bottom-up approach. The evolved gait is stable and much faster than the cart-table generated gait.

For performing Embodied Evolution (EE), a framework for running EA on a physical robot is developed. As part of this, a modular robotic configuration is constructed with *Y1* robot modules. A hardware-software system for estimating actuator position of a module, based on its servomotor's potentiometer reading is developed. A Kalman Filter (KF) for module's state estimation, based on noisy data which is in turn based on potentiometer readings, is developed. A vision system for estimating position of the robot in the evolution arena is developed. A communication protocol for establishing communication link between the host desktop personal computer (PC), running the high-level controller and the EA, and the *SkyMega* board on-board the robot, running the low-level controller, is implemented. A gait is evolved on the physical *Y-bot* configuration through an EA combining features on Genetic Algorithm (GA) and Evolutionary Strategy (ES).

Finally, an optimal control policy for producing locomotion in MRs is learnt through RL. State space, action space and reward function are defined, and the standard Q-Learning algorithm is adapted for learning locomotion is MRs. Gaits are successfully learnt on four different modular robotic configurations through this method.

Bibliography

[Adolph et al., 1997] Adolph, K. E., Bertenthal, B. I., Boker, S. M., Goldfield, E. C., and Gibson, E. J. (1997). Learning in the development of infant locomotion. *Monographs of the Society for Research in Child Development*, pages i–162. [6]

[Alexander, 2003] Alexander, R. M. (2003). *Principles of animal locomotion*. Princeton University Press. [6]

[Arnold and Bennett, 1988] Arnold, S. J. and Bennett, A. F. (1988). Behavioural variation in natural populations. v. morphological correlates of locomotion in the garter snake (thamnophis radix). *Biological Journal of the Linnean Society*, 34(2):175–190. [6]

[Bonardi et al., 2012] Bonardi, S., Moeckel, R., Sproewitz, A., Vespignani, M., and Ijspeert, A. J. (2012). Locomotion through reconfiguration based on motor primitives for roombots self-reconfigurable modular robots. *Robotics; Proceedings of ROBOTIK 2012; 7th German Conference on*, pages 1 –6. [24, 33, 39, 40]

[Bongard and Pfeifer, 2003] Bongard, J. C. and Pfeifer, R. (2003). Evolving complete agents using artificial ontogeny. In *Morpho-functional Machines: The New Species*, pages 237–258. Springer. [8]

[Brunete et al., 2012a] Brunete, A., Hernando, M., Gambao, E., and Torres, J. (2012a). A behaviour-based control architecture for heterogeneous modular, multi-configurable, chained micro-robots. *Robotics and Autonomous Systems*, 60(12):1607–1624. [37]

[Brunete et al., 2012b] Brunete, A., Torres, J., Hernando, M., and Gambao, E. (2012b). Heterogenous multi-cofigurable chained micro-robot for exploration of small cavities. *Automation in Construction Magazine*, 21:184 – 198. [18, 19, 25, 26, 30, 31]

[Castano et al., 2002] Castano, A., Behar, A., and Will, P. (2002). The Conro modules for reconfigurable robots. *IEEE/ASME Trans. Mechatronics*, 7(4):403–409. [15, 35]

[Castano et al., 2000] Castano, A., Shen, W.-M., and Will, P. (2000). CONRO: Towards deployable robots with inter-robots metamorphic capabilities. *Autonomous Robots*, 8(3):309–324. [15, 35]

[Chiu et al., 2007] Chiu, H., Rubenstein, M., and Shen, W.-M. (2007). Multifunctional superbot with rolling track configuration. San Diego, CA. IROS 2007 Workshop on Self-Reconfigurable Robots, Systems & Applications. [30]

[Cruse, 1990] Cruse, H. (1990). What mechanisms coordinate leg movement in walking arthropods? *Trends in Neurosciences*, 13(1):15–21. [42]

[Cruse et al., 2002] Cruse, H., Dean, J., Dürr, V., Kindermann, T., Schmitz, J., and Schumm, M. (2002). *A decentralized, biologically based network for autonomous control of (hexapod) walking*, pages 383–400. Neurotechnology for Biomimetic Robots. MIT Press. [42]

[Davey et al., 2012] Davey, J., Kwok, N., and Yim, M. (2012). Emulating self-reconfigurable robots - design of the smores system. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 4464 –4469. [22, 23, 32]

[Diankov and Kuffner, 2008] Diankov, R. and Kuffner, J. (2008). Openrave: A planning architecture for autonomous robotics. Technical Report CMU-RI-TR-08-34, Robotics Institute, Pittsburgh, PA. [3, 97]

[Dürr et al., 2003] Dürr, V., Krause, A. F., Schmitz, J., and Cruse, H. (2003). Neuroethological concepts and their transfer to walking machines. *International Journal of Robotics Research*, 22(3-4):151–167. [43]

[Endo et al., 2008] Endo, G., Morimoto, J., Matsubara, T., Nakanishi, J., and Cheng, G. (2008). Learning cpg-based biped locomotion with a policy gradient method: Application to a humanoid robot. *The International Journal of Robotics Research*, 27(2):213–228. [40]

[Ficici et al., 1999] Ficici, S. G., Watson, R. A., and Pollack, J. B. (1999). Embodied evolution: A response to challenges in evolutionary robotics. In *Proceedings of the eighth European workshop on learning robots*, pages 14–22. [106]

[Fukuda and Kawauchi, 1990] Fukuda, T. and Kawauchi, Y. (1990). Cellular robotic system(cebot) as one of the realization of self-organizing intelligent universal manipulator. In *Proceedings of the 1990 IEEE International Conference on Robotics and Automation*, pages 662–667. [13]

[Gatesy and Biewener, 1991] Gatesy, S. and Biewener, A. (1991). Bipedal locomotion: effects of speed, size and limb posture in birds and humans. *Journal of Zoology*, 224(1):127–147. [6]

[González-Fierro et al., 2014] González-Fierro, M., Balaguer, C., Swann, N., and Nanayakkara, T. (2014). Full-body postural control of a humanoid robot with both imitation learning and skill innovation. *International Journal of Humanoid Robotics*, 1(04):613–636. [40]

[Gonzalez-Gomez, 2008] Gonzalez-Gomez, J. (2008). *Modular Robotics and Locomotion: Application to Limbless robots*. PhD thesis. [3, 25, 26, 49, 106]

[Gonzalez-Gomez and Boemo, 2005] Gonzalez-Gomez, J. and Boemo, E. I. (2005). Motion of minimal configurations of a modular robot: Sinusoidal, lateral rolling and lateral shift. In *CLAWAR*, pages 667–674. [25, 26, 35, 45, 46]

[Hirose, 1993] Hirose, S. (1993). *Biologically Inspired Robots: Snake-Like Locomotors and Manipulators*. Oxford University Press, New York, USA. [13]

[Iida and Pfeifer, 2004] Iida, F. and Pfeifer, R. (2004). Self-stabilization and behavioral diversity of embodied adaptive locomotion. In Iida, F., Pfeifer, R., Steels, L., and Kuniyoshi, Y., editors, *Embodied Artificial Intelligence*, volume 3139 of *Lecture Notes in Computer Science*, pages 119–129. Springer Berlin Heidelberg. [41, 42]

[Ijspeert, 2008] Ijspeert, A. J. (2008). Central pattern generators for locomotion control in animals and robots: A review. *Neural Networks*, 21(4):642–653. [36]

[Kajita et al., 2003] Kajita, S., Kanehiro, F., Kaneko, K., Fujiwara, K., Harada, K., Yokoi, K., and Hirukawa, H. (2003). Biped walking pattern generation by using preview control of zero-moment point. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 2, pages 1620–1626. Ieee. [40]

[Kamimura et al., 2003] Kamimura, A., Kurokawa, H., Yoshida, E., Tomita, K., Murata, S., and Kokaji, S. (2003). Automatic locomotion pattern generation for modular robots. In *ICRA*, pages 714–720. [36]

[Kaneko et al., 2002] Kaneko, K., Kanehiro, F., Kajita, S., Yokoyama, K., Akachi, K., Kawasaki, T., Ota, S., and Isozumi, T. (2002). Design of prototype humanoid robotics platform for hrp. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 3, pages 2431–2436. IEEE. [40]

[Kernbach et al., 2011a] Kernbach, S., Liedke, J., Matthias, R., Schlachter, F., Schwarzer, C., Girault, B., and Alschbach, P. (2011a). Heterogeneity of autonomous robotic modules for the reliability of a self-reconfigurable multi-robot organisms. In *Proceedings of IROS11 workshop on Reconfigurable Modular Robotics: Challenges of Mechatronic and Bio-Chemo- Hybrid Systems*, San Francisco. [16]

[Kernbach et al., 2011b] Kernbach, S., Schlachter, F., Humza, R., Liedke, J., Popesku, S., Russo, S., Ranzani, T., Manfredi, L., Stefanini, C., Matthias, R., et al. (2011b). Heterogeneity for increasing performance and reliability of self-reconfigurable multi-robot organisms. *arXiv preprint arXiv:1109.2288*. [16, 18, 32]

[Kernbach et al., 2011c] Kernbach, S., Scholz, O., Harada, K., Popesku, S., Liedke, J., Raja, H., Liu, W., Caparrelli, F., Jemai, J., Havlik, J., et al. (2011c). Multi-robot organisms: State of the art. *arXiv preprint arXiv:1108.5543*. [31]

[Khatib et al., 2004] Khatib, O., Sentis, L., Park, J., and Warren, J. (2004). Whole-body dynamic behavior and control of human-like robots. *Int. J. of Humanoid Robotics*, 1(1):29–43. [40]

[Kurokawa et al., 2003] Kurokawa, H., Kamimura, A., Yoshida, E., Tomita, K., Kokaji, S., and Murata, S. (2003). M-tran ii: metamorphosis from a four-legged walker to a caterpillar. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 3, pages 2454–2459. IEEE. [25, 30]

[Kutzer et al., 2010] Kutzer, M. D. M., Moses, M. S., Brown, C. Y., Scheidt, D. H., Chirikjian, G. S., and Armand, M. (2010). Design of a new independently-mobile reconfigurable modular robot. In *Proceedings of the 2010 IEEE International Conference on Robotics and Automation (ICRA)*. [23]

[Lal et al., 2006] Lal, S. P., Yamada, K., and Endo, S. (2006). Studies on motion control of a modular robot using cellular automata. In *Proceedings of the 19th Australian joint conference on Artificial Intelligence: advances in Artificial Intelligence*, AI'06, pages 689–698, Berlin, Heidelberg. Springer-Verlag. [34]

[Lal et al., 2008] Lal, S. P., Yamada, K., and Endo, S. (2008). Evolving motion control for a modular robot. *Applications and Innovations in Intelligent Systems XV*, pages 245–258. [37]

[Lipson and Pollack, 2000] Lipson, H. and Pollack, J. B. (2000). Automatic design and manufacture of robotic lifeforms. *Nature*, 406(6799):974–978. [8]

[Liu et al., 2012] Liu, P., Zhu, Y., Cui, X., Wang, X., Yan, J., and Zhao, J. (2012). Multisensor-based autonomous docking for ubot modular reconfigurable robot. In *Mechatronics and Automation (ICMA), 2012 International Conference on*, pages 772 –776. [21]

[Matthias et al., 2012] Matthias, R., Bihlmaier, A., and Worn, H. (2012). Robustness, scalability and flexibility: Key-features in modular self-reconfigurable mobile robotics. In *Multisensor Fusion and Integration for Intelligent Systems (MFI), 2012 IEEE Conference on*, pages 457–463. [16]

[McGeer, 1990] McGeer, T. (1990). Passive dynamic walking. *The International Journal of Robotics Research*, 9(2):62–82. [41]

[Meng and Jin, 2011] Meng, Y. and Jin, Y. (2011). Morphogenetic self-reconfiguration of modular robots. In Meng, Y. and Jin, Y., editors, *Bio-Inspired Self-Organizing Robotic Systems*, volume 355 of *Studies in Computational Intelligence*, pages 143–171. Springer Berlin Heidelberg. [33, 38]

[Meng et al., 2011] Meng, Y., Zhang, Y., Sampath, A., Jin, Y., and Sendhoff, B. (2011). Cross-ball: A new morphogenetic self-reconfigurable modular robot. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 267 –272. [19, 20]

[Meng et al., 2010] Meng, Y., Zheng, Y., and Jin, Y. (2010). A morphogenetic approach to self-reconfigurable modular robots using a hybrid hierarchical gene regulatory network. In *Proc. of the Alife XII Conference, Odense, Denmark, 2010*, pages 765–772. [19, 20, 39]

[Miura et al., 2009] Miura, K., Morisawa, M., Nakaoka, S., Kanehiro, F., Harada, K., Kaneko, K., and Kajita, S. (2009). Robot motion remix based on motion capture data towards human-like locomotion of humanoid robots. In *Humanoid Robots, 2009. Humanoids 2009. 9th IEEE-RAS International Conference on*, pages 596–603. IEEE. [41]

[Moeckel et al., 2005] Moeckel, R., Jaquier, C., Drapel, K., Dittrich, E., Upegui, A., and Ijspeert, A. J. (2005). Yamor and bluemove - an autonomous modular robot with bluetooth interface for exploring adaptive locomotion. In *CLAWAR*, pages 685–692. [36]

[Mombaur et al., 2010] Mombaur, K., Truong, A., and Laumond, J.-P. (2010). From human to humanoid locomotion—an inverse optimal control approach. *Autonomous robots*, 28(3):369–383. [40]

[Monje et al., 2013] Monje, C., Pierro, P., Ramos, T., González-Fierro, M., and Balaguer, C. (2013). Modeling and simulation of the humanoid robot hoap-3 in the openhrp3 platform. *Cybernetics and Systems*, 44(8):663–680. [40, 88]

[Morimoto and Atkeson, 2007] Morimoto, J. and Atkeson, C. G. (2007). Learning biped locomotion. *Robotics & Automation Magazine, IEEE*, 14(2):41–51. [40]

[Nagasaka et al., 1999] Nagasaka, K., Inoue, H., and Inaba, M. (1999). Dynamic walking pattern generation for a humanoid robot based on optimal gradient method. In *Systems, Man, and Cybernetics, 1999. IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on*, volume 6, pages 908–913. IEEE. [40]

[Nakanishi et al., 2004] Nakanishi, J., Morimoto, J., Endo, G., Cheng, G., Schaal, S., and Kawato, M. (2004). Learning from demonstration and adaptation of biped locomotion. *Robotics and Autonomous Systems*, 47(2):79–91. [40]

[Or, 2010] Or, J. (2010). A hybrid cpg–zmp control system for stable walking of a simulated flexible spine humanoid robot. *Neural Networks*, 23(3):452–460. [40]

[Paul, 2006] Paul, C. (2006). Morphological computation: A basis for the analysis of morphology and control requirements. *Robotics and Autonomous Systems*, 54(8):619 – 630. [41]

[Pfeifer and Bongard, 2006] Pfeifer, R. and Bongard, J. (2006). *How the body shapes the way we think: a new view of intelligence*. MIT press. [133]

[Pouya et al., 2010] Pouya, S., van den Kieboom, J., Spröwitz, A., and Ijspeert, A. J. (2010). Automatic gait generation in modular robots: "to oscillate or to rotate; that is the question". In *IROS*, pages 514–520. [29, 36]

[Russo et al., 2012] Russo, S., Harada, K., Ranzani, T., Manfredi, L., Stefanini, C., Menciassi, A., and Dario, P. (2012). Design of a robotic module for autonomous exploration and multimode locomotion. *Mechatronics, IEEE/ASME Transactions on*, PP(99):1–10. [16, 27, 28, 29]

[Ryan et al., 2012] Ryan, T. M., Silcox, M. T., Walker, A., Mao, X., Begun, D. R., Benefit, B. R., Gingerich, P. D., Köhler, M., Kordos, L., McCrossin, M. L., et al. (2012). Evolution of locomotion in anthropoidea: the semicircular canal evidence. *Proceedings of the Royal Society of London B: Biological Sciences*, 279(1742):3467–3475. [6]

[Salemi et al., 2006] Salemi, B., Moll, M., and Shen, W.-M. (2006). SUPERBOT: A deployable, multi-functional, and modular self-reconfigurable robotic system. Beijing, China. [35]

[Schlachter et al., 2012] Schlachter, F., Schwarzer, C., Girault, B., and Levi, P. (2012). A modular software framework for heterogeneous reconfigurable robots. In *Autonomous Mobile Systems 2012*, pages 193–201. Springer. [16]

[Shafii et al., 2009] Shafii, N., Javadi, M. H. S., and Kimiaghalam, B. (2009). A truncated fourier series with genetic algorithm for the control of biped locomotion. In *2009 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pages 1781–1785. IEEE. [87]

[Shan et al., 2000] Shan, J., Junshi, C., and Jiapin, C. (2000). Design of central pattern generator for humanoid robot walking based on multi-objective ga. In *Intelligent Robots and Systems, 2000.(IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, volume 3, pages 1930–1935. IEEE. [40]

[Shan and Nagashima, 2002] Shan, J. and Nagashima, F. (2002). Neural locomotion controller design and implementation for humanoid robot hoap-1. In *20th annual conference of the robotics society of Japan*. [40]

[Shen et al., 2006] Shen, W.-M., Krivokon, M., Chiu, H., Everist, J., Rubenstein, M., and Venkatesh, J. (2006). Multimode locomotion via superbot reconfigurable robots. *Autonomous Robots*, 20(2):165–177. [30]

[Shen et al., 2000] Shen, W.-M., Lu, Y., and Will, P. (2000). Hormone-based control for self-reconfigurable robots. In *Proceedings of the 2000 International Conference on Autonomous Agents*, Barcelona, Spain. [35]

[Shen et al., 2002] Shen, W.-M., Salemi, B., and Will, P. (2002). Hormone-inspired adaptive communication and distributed control for CONRO self-reconfigurable robots. 18(5):700–712. [15, 35]

[Sproewitz et al., 2009] Sproewitz, A., Billard, A., Dillenbourg, P., and Ijspeert, A. J. (2009). Roombots-mechanical design of self-reconfiguring modular robots for adaptive furniture. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 4259–4264. IEEE. [23, 36, 39]

[Spröwitz, 2010] Spröwitz, A. (2010). *Roombots: Design and implementation of a modular robot for reconfiguration and locomotion*. PhD thesis, ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE. [23, 36, 39]

[Spröwitz et al., 2008] Spröwitz, A., Moeckel, R., Maye, J., and Ijspeert, A. J. (2008). Learning to move in modular robots using central pattern generators and online optimization. *I. J. Robotic Res.*, 27(3-4):423–443. [36]

[Stoy et al., 2002] Stoy, K., Shen, W.-M., and Will, P. (2002). Using role-based control to produce locomotion in chain-type self-reconfigurable robots. *Mechatronics, IEEE/ASME Transactions on*, 7(4):410–417. [29, 36]

[Støy et al., 2003] Støy, K., Shen, W.-M., and Will, P. (2003). A simple approach to the control of locomotion in self-reconfigurable robots. *Robotics and Autonomous Systems*, 44(3–4):191 – 199. Best papers presented at IAS-7. [25, 26, 30, 35]

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge. [135, 136]

[Takaya and Arita, 2003] Takaya, Y. U. and Arita, T. (2003). Situated and embodied evolution in collective evolutionary robotics. In *In Proc. of the 8th International Symposium on Artificial Life and Robotics*. Citeseer. [106]

[Wang et al., 2009] Wang, W., Wang, K., and Zhang, H. (2009). Crawling gait realization of the mini-modular climbing caterpillar robot. *Progress in Natural Science*, 19(12):1821–1829. [27, 28]

[Watson et al., 1999] Watson, R., Ficiei, S., and Pollack, J. (1999). Embodied evolution: Embodying an evolutionary algorithm in a population of robots. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 1, pages –342 Vol. 1. [106]

[Wieber, 2006] Wieber, P. (2006). Trajectory free linear model predictive control for stable walking in the presence of strong perturbations. In *Humanoid Robots, 2006 6th IEEE-RAS International Conference on*, pages 137–142. IEEE. [40]

[Williams, 1999] Williams, T. M. (1999). The evolution of cost efficient swimming in marine mammals: limits to energetic optimization. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 354(1380):193–201. [6]

[Wisse, 2004] Wisse, M. (2004). Three additions to passive dynamic walking; actuation, an upper body, and 3d stability. In *Humanoid Robots, 2004 4th IEEE/RAS International Conference on*, volume 1, pages 113–132 Vol. 1. [41]

[Wolfe et al., 2012] Wolfe, K., Moses, M., Kutzer, M., and Chirikjian, G. (2012). M 3 express: A low-cost independently-mobile reconfigurable modular robot. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 2704 –2710. [23, 24]

[Yamane and Nakamura, 2003] Yamane, K. and Nakamura, Y. (2003). Dynamics filter-concept and implementation of online motion generator for human figures. *Robotics and Automation, IEEE Transactions on*, 19(3):421–432. [40]

[Yang et al., 2006] Yang, L., Chew, C.-M., Poo, A. N., and Zielinska, T. (2006). Adjustable bipedal gait generation using genetic algorithm optimized fourier series formulation. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4435–4440. IEEE. [87]

[Yim et al., 2000] Yim, M., Duff, D., and Roufas, K. (2000). Polybot: A modular reconfigurable robot. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, pages 514–520. [15, 34, 106]

[Yim et al., 2001] Yim, M., Duff, D., and Roufas, K. (2001). Evolution of polybot: A modular reconfigurable robot. In *2001 COE/Super-Mechano-Systems Workshop*, Tokyo, Japan. [15]

[Yim et al., 2007] Yim, M., Shen, W.-M., Salemi, B., Rus, D., Moll, M., Lipson, H., Klavins, E., and Chirikjian, G. S. (2007). Modular self-reconfigurable robot systems [grand challenges of robotics]. *IEEE Roboticsand Automation Magazine*, 14(1):43–52. [15]

[Zhang et al., 2009] Zhang, H., Gonzalez-Gomez, J., and Zhang, J. (2009). A new application of modular robots on analysis of caterpillar-like locomotion. In *Mechatronics, 2009. ICM 2009. IEEE International Conference on*, pages 1–6. [35, 45]

[Zhao et al., 2011] Zhao, J., Cui, X., Zhu, Y., and Tang, S. (2011). A new self-reconfigurable modular robotic system ubot: Multi-mode locomotion and self-reconfiguration. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1020 –1025. [21, 22]

[Zykov et al., 2005] Zykov, V., Mytilinaios, E., Adams, B., and Lipson, H. (2005). Self-reproducing machines. *Nature*, 435:163–164. [16, 17]

Abbreviations

1D one-dimension 48
1D one-dimensional 59
2D two-dimension 159
2D two-dimensional 160
3D three-dimension 106
3D three-dimensional 144
ACM Active Connection Mechanisms 39
ACM Active Cord Mechanism 13
ADC Analog to Digital Converter 109
AMP Atomic Motor Primitive 39
ANN Artificial Neural Network 160
ASCII American Standard Code for Information Interchange 121
CC Central Control 33
CC central controller 37
CEBOT Cellular Robotic System 13
CMP Composed Motor Primitive 39
CoBoLD Cone Bolt Locking Device 18
COF Coefficient of Friction 68
COG Center of Gravity 97
CPG Central Pattern Generator 36

DC Distributed Control . . . 33
DOF Degree of Freedom . . . 159
EA Evolutionary Algorithm . . . 159
EE Embodied Evolution . . . 161
EPFL Swiss Federal Institute of Technology in Lausanne . . . 36
ER Evolutionary Robotics . . . 8
ES Evolutionary Strategy . . . 161
***ffc*-Fourier controller** first frequency component Fourier controller . . . 85
FOV Field of View . . . 129
GA Genetic Algorithm . . . 161
GRN Gene Regulatory Network . . . 38
HOAP-3 Humanoid for Open Architecture Platform . . . 159
HRP Humanoid Robotics Platform . . . 40
HSV Hue Saturation Value . . . 118
ICF Intra-Configuration Force . . . 160
KF Kalman Filter . . . 161
ODE Open Dynamics Engine . . . 97
MDP Markov Decision Process . . . 157
ML Machine Learning . . . 157
MR Modular Robot . . . 159
PC personal computer . . . 161
PID Proportional–Integral–Derivative . . . 72
PWM Pulse Width Modulation . . . 23
RB Roombots . . . 36
RGB Red Green Blue . . . 132
RL Reinforcement Learning . . . 157
SD standard deviation . . . 154
SL Supervised Learning . . . 133
SMORES Self-assembling MOdular Robot for Extreme Shape-shifting . . . 33
SRMR Self-Reconfigurable Modular Robot . . . 38
TD Temporal-Difference . . . 157
***tfc*-Fourier controller** two frequency component Fourier controller . . . 60
USB Universal Serial Bus . . . 121
ZMP Zero Moment Point . . . 40

Index