Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Hierarchical Control for Bipedal Locomotion using Central Pattern Generators and Neural Networks

Master's Thesis

at Knowledge Technology, WTM
Department of Informatics
MIN-Faculty
Universität Hamburg

Sayantan Auddy

Intelligent Adaptive Systems
Student-ID number: 6640595

December 29, 2017

Reviewers: Prof. Dr. Stefan Wermter
Dr. Sven Magg

Adviser: Dr. Sven Magg

Abstract

The walking movement of humans is graceful and robust. This movement is the result of synchronization between the neural mechanisms which generate rhythmic motion, and the dynamics of the skeletal structure. The overall movement is optimized by high-level control centers. Drawing inspiration from this mechanism, a hierarchical controller for bipedal locomotion of robots is proposed in this thesis.

Artificial central pattern generators (CPGs) mimic the behavior of the neural mechanisms which produce rhythmic motion in animals. Many existing methods use CPG networks for bipedal locomotion, but most of them focus solely on the CPGs. The proposed controller augments the functionality of a CPG network by adding a novel high-level controller on top of it. Thus, at the lower level, a CPG network with feedback pathways controls the individual joints. The parameters of the CPG network are optimized by a genetic algorithm. At the higher level, a neural network modulates the output of the CPG network in order to optimize the robot's movements with respect to an overall objective. In this case, the objective is to minimize the lateral deviation while walking. The neural network is trained using reinforcement learning.

The proposed controller was successfully used to produce stable bipedal locomotion for the NICO robot in simulation. Results of experiments show that the high-level controller was able to improve the performance of the low-level CPG network. Additionally, by comparing the performance of CPG networks with and without feedback mechanisms, the relative effectiveness of low-level feedback has been shown. The proposed controller is not strongly coupled to a particular robot model and is modular in design. The results obtained, by using this controller in simulation, encourage its use on the physical robot in the future.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years, rapid progress in the fields of artificial intelligence and robotics has enabled robots to become ever more sophisticated and capable of performing a wide variety of tasks. Today, robots can no longer be considered to be machines restricted only to laboratory or industrial environments, but have begun to proliferate into our homes and offices. As robots continue to grow advanced and become more integrated into human society, they would need to operate in environments which have been designed for humans. Hence, being able to walk is a fundamental ability that such robots must possess.

Walking is an activity that many of us take for granted. Yet, this apparently 'simple' action involves many complex mechanisms which are tied together to produce our own bipedal movements. Achieving stable and robust bipedal locomotion in robots is a very difficult task. Humanoid robots contain many joints and this multiplicity of degrees-of-freedom greatly compounds the control problem for walking. How such joints can be effectively controlled and made to work in a synchronous manner, is a challenging research question. Humanoid robots have come a long way since the days of early anthropomorphic robots such as WABOT-1 [49], WABOT-2 [90] and WL-10RD [34]. Today, advanced humanoid robots such as Honda's Asimo [82, 27], Kawada's HRP [45, 46, 47], IIT's iCub [69] and Boston Dynamics' Atlas [18] can perform many difficult tasks such as climbing stairs or walking over uneven terrain. However, there is still a long way to go before robots can be as robust as animals when it comes to walking in different kinds of environments.

Some of the most popular approaches for bipedal walking are model-based techniques such as Zero Moment Point (ZMP) [99, 38]. These techniques rely heavily on accurate information about the kinematics and dynamics of the robot, and using some criteria for stability, they compute trajectories for all the joints using inverse kinematics. Although this leads to robust walking in controlled environments, there are certain problems. Firstly, such methods are computationally intensive due to the matrix inversion operations that are needed. Secondly, they rely on precise joint control and so expensive servos are better suited. Thirdly, the natural dynamics of the body is not utilized and often, it is overridden to maintain control over the robot. This results in a walk, which although may be stable, is not energy efficient or graceful and does not scale well to unpredictable conditions.

The problem of bipedal locomotion in a robot involves a number of sub-problems such as movement generation and synchronization of the different joints, adapting to environmental disturbances and achieving overall gait-related objectives like maintaining balance or walking in a particular direction or with the desired speed. Instead of trying to solve all the sub-problems together, a better approach is to handle them separately. In the current work, the problem of bipedal locomotion has been decomposed into two separate tasks: motion generation and motion modulation. These tasks are solved by using a hierarchical controller composed of a network of central pattern generators and a neural network.

1.1 Motivation

In many areas of artificial intelligence, mimicking biological processes has proven to be an attractive way of solving difficult problems. There is evidence to suggest that neural circuits called central pattern generators (CPGs) are responsible for producing the rhythmic patterns responsible for movement in animals [89, 6, 24]. The CPGs can produce rhythmic output on their own and can be modulated by higher brain centers to produce specific kinds of motion [84]. The control structure for bipedal locomotion, proposed in this thesis, is motivated by the neural mechanism for walking that is found in nature.

At the lower level, a network of artificial CPGs autonomously produces rhythmic patterns for controlling the individual joints. The CPG parameters are optimized according to specific criteria so that a stable gait is achieved. Feedback mechanisms are utilized so that the joint movements can synchronize with the overall motion of the robot. The CPG network is based on the work of Cristiano et al. [11]. For the motion modulation task, a neural network acts as a high-level controller and modulates a few parameters of the CPG network to achieve certain characteristics in the final walking motion. This approach greatly simplifies the process of overall control since the higher control mechanism does not need to control the joints individually.

A CPG-based method for bipedal locomotion provides some advantages over model-based methods. CPGs are implemented using coupled differential equations which are computationally inexpensive to solve. They exhibit stable limit cycle properties and are quite robust to external disturbances. So, they can be implemented on most kinds of robots, not only the most expensive ones with high precision motors. The biggest advantage of CPGs is that they can adapt to the natural dynamics of the robot. Instead of suppressing the natural dynamics, they can work in unison with it. This greatly improves the energy efficiency of the movement.

CPGs such as the Matsuoka oscillator [65, 66] and the Hopf oscillator [79] have been around for a long time and they have been used in several studies which aim to develop algorithms for bipedal locomotion [96, 10, 11, 12, 52, 79, 35]. Although these approaches have achieved bipedal locomotion in different robots, they do not deal with the question of a high-level controller which is capable of learning. When we walk, instead of explicitly controlling every joint, we focus on higher forms of

control such as balance, direction or speed and make the desired adjustments to our gait. Moreover, as children grow up, they learn how to modulate their gait and gradually get better at walking.

CPGs provide parameters through which their outputs may be modulated. This provision is exploited by using a neural network as a mechanism of trainable high-level control. Additionally, many of the previously mentioned studies use some form of feedback mechanism to modulate the CPGs. However, these mechanisms are not compared on the basis of measurable parameters. An objective analysis of different types of feedback will shed light on their relative advantages and disadvantages. This comparative analysis is also taken up in this thesis.

CPG based algorithms are not tightly coupled with a particular robot model and they can be easily adapted to different robots. As such, they provide a promising way of developing walking capabilities for new robots. The Neuro-Inspired Companion (NICO) robot [51] is a humanoid robot being developed by the Knowledge Technology group at Universität Hamburg. Since the NICO does not have an algorithm for walking yet, developing one for it is also a motivation for this thesis. The current work has been carried out using the NICO in the Virtual Robot Experimentation Platform (VREP) simulator [14].

1.2 Research Objectives

The objective of this thesis is to develop a bio-inspired, hierarchical controller for bipedal locomotion based on CPGs and neural networks, which can be used to make the NICO robot walk in simulation and can potentially also be used on other similar robots. The primary research question that this thesis aims to address is:

> *Can bipedal locomotion be achieved by using a low-level CPG-based controller modulated by a high-level neural network controller, and is such a control mechanism beneficial for walking?*

An analysis of the comparative performances of different low-level feedback mechanisms can demonstrate the effects induced by the feedback loops. This will also help in choosing an appropriate type of feedback. In this respect, the secondary research question that this thesis aims to answer is:

> *Do feedback mechanisms for CPGs improve the gait? How do different feedback mechanisms compare against each other?*

The focus of this work is on making the higher level controller capable of learning. At the same time, the controller should not require information about the kinematics or dynamics of the robot, and so it should be independent of specific robot models. The overall design of the controller should be modular so that its functionality can be extended in the future.

1.3 Organization

The purpose of this chapter has been to provide an overview of some of the challenges involved in bipedal locomotion and the approach followed in this thesis to overcome them. The proposed method is inspired by biological control mechanisms for walking. It also involves the use of Matsuoka oscillators, genetic algorithms and policy gradient based reinforcement learning. Chapter 2 contains an overview of these different topics and also provides references from where more detailed information may be gathered. This is followed by chapter 3, which discusses different relevant studies dealing with bipedal locomotion. Different methods are summarized and their relative strengths and weaknesses are discussed.

The next three chapters form the core of this thesis and contain the details of the proposed method and the experiments and analyses that were conducted. Chapter 4 explains the problem statement in detail and provides a complete overview of the system architecture, including the configuration of the CPG network, details of the feedback mechanisms and the internal structure of the high-level neural network controller.

Chapter 5 deals with the problem of basic gait generation using CPGs. This is done by optimizing the parameters of a network of Matsuoka oscillators using a genetic algorithm. Topics discussed in this chapter include the design of the fitness function, parameter search space bounding, hyperparameter setting and results of the optimization experiments. Three kinds of basic gait are developed - one without any feedback and two others using different feedback types. These gaits are then compared to each other using some parameters which can objectively measure the quality of the robot's walk.

Chapter 6 deals with high-level control. Although the robot is able to walk by using the CPG network and feedback loops, it tends to deviate from a straight line trajectory due to systematic errors and slippage. A neural network is used as a high-level controller in order to rectify this issue. The network is trained using the Deep Deterministic Policy Gradient algorithm [61]. The topics discussed in this chapter include reinforcement learning problem formulation, reward function design, hyperparameter tuning, experimental setup, and results.

Chapter 7 contains concluding remarks. It summarizes the key results presented in this thesis and discusses some areas of future work, such as, how the proposed approach may be tested on a physical robot and how this hierarchical controller may be extended for optimizing other aspects of the robot's gait.

Chapter 2

Background

The method for bipedal locomotion control, proposed in this thesis, is based on findings in neuroscience and artificial intelligence. The control architecture is inspired by the locomotion control mechanism in vertebrates. Some important aspects of these studies are discussed in this chapter, such as the organization of the neural components for motor control in animals.

Different kinds of artificial CPG models exist in the literature for locomotion control of robots. The general features of these models, which makes them suitable for this task, are discussed, along with examples of different model types. The fundamentals and background of two of the most popular models, the Hopf oscillator and the Matsuoka oscillator are presented. Additionally, reasons for choosing the Matsuoka oscillator for the current work are also stated.

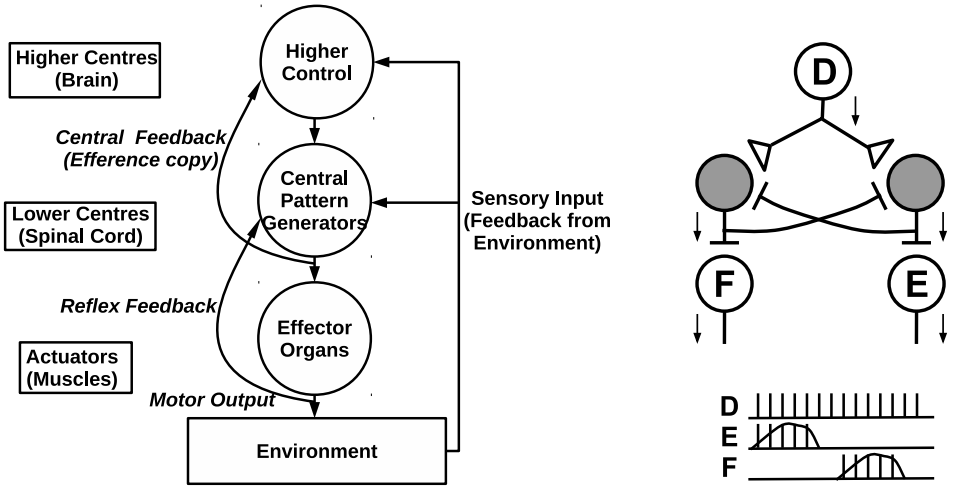
For optimizing the parameters of the CPGs, a genetic algorithm has been used. This chapter includes an overview of this evolutionary optimization technique and discusses a few of its salient characteristics.

The neural network used in this work was trained using the Deep Deterministic Policy Gradient (DDPG) algorithm [61]. An overview of this algorithm, and of policy gradient reinforcement learning in general, is presented in the last section.

2.1 Biological Motivation

In both higher level vertebrates as well as in some invertebrates, neural circuits known as central pattern generators (CPGs) are responsible for producing the rhythmic movements necessary for locomotion [31]. These CPGs can produce rhythmic patterns of neural activity without the need of rhythmic or sensory inputs, and they are the basis of fundamental activities such as chewing, breathing and locomotion. Several studies have proven the existence of CPGs in animals and their responsibility for the movements of the animals [6, 24, 89]. Although sensory feedback is not required to generate the rhythmic patterns by CPGs, it plays an important role in modulating the rhythmic patterns and for coordinating the movements of the body [31]. Vertebrate locomotor systems are organized such that the CPGs produce basic rhythmic patterns, and the higher brain centers modulate

these patterns based on external sensory inputs [31]. Such an organization has important consequences, even from a robotics perspective, since it drastically reduces the dimensionality of higher level control signals. The higher centers do not need to specify individual muscle (or joint) activity and instead only need to modulate the CPG activity using a small set of parameters [31].



(a) Neural components of motor systems. Feedback information flows from the lower centers to the higher centers. Sensory input to CPG is only used for modulation. The CPGs can produce rhythmic output in the absence of sensory input as well [84].

(b) Half-center model: D denotes a driver neuron which supplies a constant input, E and F are the extensor and flexor motor neurons. Filled circles indicate inhibitory neurons, unfilled circles denote excitatory neurons. Spike trains shown at the bottom [84].

Figure 2.1: Neural components of motor systems and a half-center CPG model [84].

A general framework for the nervous control of motor systems is illustrated in figure 2.1a. The first element in this framework is the CPG, located in the spinal cord. This is responsible for generating coordinated rhythmic outputs for the motor neurons. The CPG, in turn, is controlled and modulated by the higher motor centers. There also exist feedback circuits which relay information from the muscles (proprioceptive feedback) and from the environment through other sensory pathways. The feedback information is transmitted from lower to higher centers. The proprioceptive feedback informs the higher center about any deviations that exist between the commanded signal and the output [84].

The internal circuitry of the CPG generally belongs to one of three major categories: the half-center model, the closed-loop model and the pacemaker model [84]. Only the half-center model is discussed here as it is relevant to the material presented in this thesis. The half-center model, as shown in figure 2.1b, was first proposed by Brown [5] to explain the alternating activation of the flexor and extensor muscles of a decerebrated cat's limbs during walking on a treadmill. Brown proposed that the neural networks responsible for generating rhythmic patterns could do so 'centrally', without input from the sensory neurons [31]. Each pool of

motor neurons for flexor (F) or extensor (E) muscles is activated by a corresponding 'half-center of interneurons' shown with filled circles in figure 2.1b). Another set of neurons (D) provides a constant excitatory drive to the half-centers. Each half-center has inhibitory connections to the other half-center, and this ensures that when one pool is activated, the other one is deactivated. According to Brown, as the first half-center was activated, a process of fatigue would build up in the inhibitory connection between the half-centers which would ultimately lead to a switching of activity between the half-centers. The process of fatigue can be modeled by any process that induces self-inhibition of the active half-centers [84].

The work proposed in this thesis is motivated by Brown's half-center neuron model as well as by the organization of the neural control structure for locomotion in animals. The Matsuoka oscillator is closely modeled on the half-center neuron model with alternating centers of activity (figure 2.1b). This CPG forms the basic building block of the proposed control mechanism. The usage of a neural network to modulate a CPG network is directly inspired by the hierarchy shown in figure 2.1a. The design of feedback strategies is also modeled on the feedback pathways shown here.

2.2 Artificial Central Pattern Generators

Artificial central pattern generators exist in numerous forms [4, 30, 54, 101, 63, 73, 9, 60], but almost all of them use coupled differential equations that are numerically integrated to obtain the output signals [31]. The inherent properties of CPGs make them a suitable candidate for the locomotion control of legged robots. First of all, CPG based models display stable limit cycle behavior, which means that although the generated patterns may be modified under the effect of a disturbance, as soon as the disturbance is removed, the CPG returns to its normal behavior. Secondly, since CPGs handle the low-level motor commands, the higher level controllers can focus on tasks such as maintaining balance. Thirdly, CPGs can handle sensory feedback easily, since the feedback signals can be conveniently added as coupling terms in the differential equations of the CPGs. This also leads to the property of entrainment, where the CPG tunes itself to the dynamics of the mechanical system and can effectively utilize the dynamics of the robot. Due to their modular nature, whereby each CPG can perform its computations independently, CPGs are also suitable for use in modular robots consisting of multiple interlinked sections [31].

While developing a locomotion controller for a robot, the first and perhaps the most important decision that needs to be made is regarding the type of CPG to use. In [30] Ijspeert and Crespi have used a CPG model inspired by the neural circuits controlling locomotion in a lamprey. Lewis et al. have used a spiking CPG model for locomotion in a bipedal robot in [60]. In [63], Lu et al. used a cyclic inhibitory CPG model for controlling the movements of a snake-like robot. However, two of the most popular CPG models for robot locomotion are the Hopf oscillator and the Matsuoka oscillator, and these are discussed in the following sub-sections.

2.2.1 Hopf Oscillator

The Adaptive Frequency Hopf oscillator was developed in [32] and [80]. A special property of this kind of oscillator is that they can learn the frequency of an external periodic signal without any kind of additional optimization process and the learned frequency remains embedded within the oscillator even after the teaching signal has been withdrawn. Hence the oscillator can learn and adapt to the input signal [79]. The Hopf oscillator is defined by the following differential equations:

$$\dot{x} = \gamma(\mu - r^2)x - \omega y + \epsilon F(t) \tag{2.1a}$$

$$\dot{y} = \gamma(\mu - r^2)y + \omega x \tag{2.1b}$$

$$\dot{\omega} = -\epsilon F(t)\frac{y}{r} \tag{2.1c}$$

where (x, y) denotes the Cartesian coordinates in the phase plane, $r = \sqrt{x^2 + y^2}$, μ is a parameter which controls the amplitude of the oscillations, ω is the intrinsic frequency of the oscillator, γ controls the speed with which the oscillator returns to its intrinsic oscillation after being perturbed, $F(t)$ is the external periodic signal which the oscillator must synchronize with and $\epsilon > 0$ is a coupling constant [32, 80, 79]. Since the phase dynamics is of interest here, sometimes it is more convenient to rewrite the above system of equations using polar coordinates [80]. Setting $x = r\cos\phi$ and $y = r\sin\phi$, we get:

$$\dot{r} = (\mu^2 - r^2)r + \epsilon F\cos\phi \tag{2.2a}$$

$$\dot{\phi} = \omega - \frac{\epsilon}{r}F\sin\phi \tag{2.2b}$$

If the input signal $F(t)$ contains multiple frequency components, the frequency of the CPG will adapt to one of the components of this signal. However, an appropriate linear combination of several of these oscillators can reproduce any kind of periodic signal, much like a Fourier series representation, with each oscillator adapting to one frequency component of the teaching signal [79]. Such a linear combination of Hopf oscillators is shown in figure 2.2. In this setup, each oscillator receives the same learning signal $F(t) = P_{teach}(t) - \sum_i \alpha_i x_i$, which is the difference of the input teaching signal $P_{teach}(t)$ and the signal already learned by the network $\sum_i \alpha_i x_i$. Here α_i denotes the amplitude that is associated with the i^{th} oscillator. Additionally, each oscillator is coupled with the 0^{th} oscillator, using a coupling term R_i, to synchronize the phase between all the oscillators. Using this general scheme, the network of Hopf oscillators can learn arbitrary signals. Due to the additional coupling terms, the equations 2.1-2.2 are slightly modified, the details of which can be referred to in [79].

The setup shown in figure 2.2 can be used to control a single joint using the combined learned signal $Q_{learned}$, once the Hopf oscillator network has converged. In order to control multiple joints for locomotion, multiple such networks can be linked up to form a larger network. The interconnections would be between the 0^{th} oscillators of the smaller networks. Apart from the coupling terms for inter-oscillator

connections, additional terms can be introduced into the differential equation 2.1 to account for proprioceptive feedback information [79]. Similar networks of coupled Hopf oscillators have been used for robot locomotion in [52, 20, 81, 79].

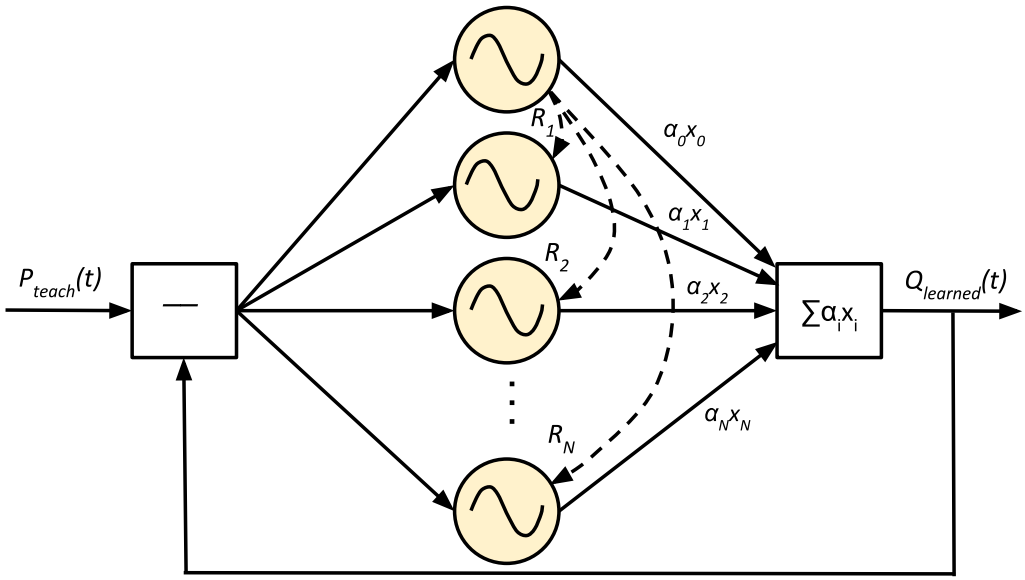


Figure 2.2: A network of Hopf oscillators [79]. Individual oscillators are denoted by the circles.

2.2.2 Matsuoka Oscillator

The Matsuoka oscillator is a mathematical model representing the general class of neural rhythm generators known as mutual inhibition networks [66]. It consists of a set of antagonistic neurons which inhibit each other's activities. The structure of this oscillator closely resembles the structure of the half-center neuron model proposed by Brown [5] (figure 2.1b).

Being a connectionist model, the individual neurons of this artificial CPG represent neuron pools in the biological model and the activity of each neuron is a generalization of the behavior of the respective neuron pools. A continuous-time, continuous-variable neuron model was adopted for the Matsuoka oscillator [66]. The autonomous dynamics of the model are based on a piece-wise linear set of differential equations, which model the discharge rates of neuron pools. The original CPG equations proposed in [66, 67] have been generalized in [43], so that the model is applicable to a system with multiple CPGs with arbitrary configurations. Here the generalized Matsuoka model is discussed and from here on, this generalized version will also be referred to as the Matsuoka model.

One of the most popular configurations of the Matsuoka oscillator, which is also the configuration used in this work, consists of two tonically-excited neurons - the extensor neuron responsible for movement in one direction and a flexor neuron which is responsible for movement in the opposite direction. From a biological

standpoint, this is similar to the way muscles control joint movement in animals, with antagonistic pairs of muscles controlling individual degrees-of-freedom.

The following equations define the behavior of the Matsuoka oscillator [43]:

$$\begin{cases} \tau \dot{u_{1i}} = -u_{1i} - w_0 y_{2i} - \beta v_{1i} + u_e + f_{1i} + s_{1i} \\ \tau' \dot{v_{1i}} = -v_{1i} + y_{1i} \\ \qquad y_{1i} = max(0, u_{1i}) \text{ and } i = 1, ..., num \end{cases} \tag{2.3}$$

$$\begin{cases} \tau \dot{u_{2i}} = -u_{2i} - w_0 y_{1i} - \beta v_{2i} + u_e + f_{2i} + s_{2i} \\ \tau' \dot{v_{2i}} = -v_{2i} + y_{2i} \\ \qquad y_{2i} = max(0, u_{2i}) \text{ and } i = 1, ..., num \end{cases} \tag{2.4}$$

Here, each CPG (denoted by the subscript i) is a nonlinear oscillator having four state variables (u_{1i}, v_{1i}, u_{2i}, v_{2i}). The subscript 1 corresponds to the extensor neuron and the subscript 2 corresponds to the flexor neuron. u_{1i} controls the discharge rate and v_{1i} controls the degree of self-inhibition of the extensor neuron. u_{2i} controls the discharge rate and v_{2i} controls the degree of self-inhibition of the flexor neuron. u_e is the tonic input to both the neurons. τ is the time constant for the rate of discharge and τ' is the time constant of adaptation or fatigue. These two time constants determine the frequency of the CPG. β is the self-inhibition constant and w_0 is the constant of mutual inhibition between the flexor and extensor neurons. num indicates the number of CPG units. The terms f_{1i} and f_{2i} are feedback variables which can be used to control the amplitude or phase of the CPG. s_{1i} and s_{2i} represent the interaction between different neighboring oscillators [11]:

$$\begin{aligned} s_{1i} &= w_{ij} u_{1j} \\ s_{2i} &= w_{ij} u_{2j} \end{aligned} \tag{2.5}$$

where i and j denote different oscillator units and w_{ij} denotes the weight of the connection from oscillator j (master) to oscillator i (slave). If w_{ij} is set to $+1$, the oscillator i will have the same phase as oscillator j (excitatory connection), if w_{ij} is set to -1, the oscillator i will have an anti-phase relationship with oscillator j (inhibitory connection) [11]. In the absence of inputs s_{1i}, s_{2i}, f_{1i} and f_{2i}, the oscillator will oscillate according to its own parameters, provided these parameters satisfy the conditions for stable oscillations [65, 66]. When s_{1i}, s_{2i}, f_{1i} and f_{2i} are provided, it implies that there exists a group of connected oscillators and these will converge to a specific pattern and limit cycle determined by the oscillator parameters and connection weights [11].

The final output of an oscillator is a scaled linear combination of the output of the extensor and flexor neurons and can be used to control the position of a single joint. The output o_i of oscillator i is:

$$o_i = -m_1 y_{1i} + m_2 y_{2i} \tag{2.6}$$

where m_1 and m_2 are scaling constants. Further, in order to have a more fine-grained control over the frequency of the oscillator, an additional parameter k_f is introduced [104, 11]. Thus, the two time constants are rewritten as:

$$\begin{aligned} \tau &= \tau_0 k_f \\ \tau' &= \tau'_0 k_f \end{aligned} \tag{2.7}$$

where τ_0 and τ'_0 are the original time constants and τ and τ' are the modified time constants after the scaling factor k_f is applied [11]. A lower or higher value of k_f results in a higher frequency or lower frequency of oscillation respectively. A diagrammatic representation of the Matsuoka oscillator is given in figure 2.3.

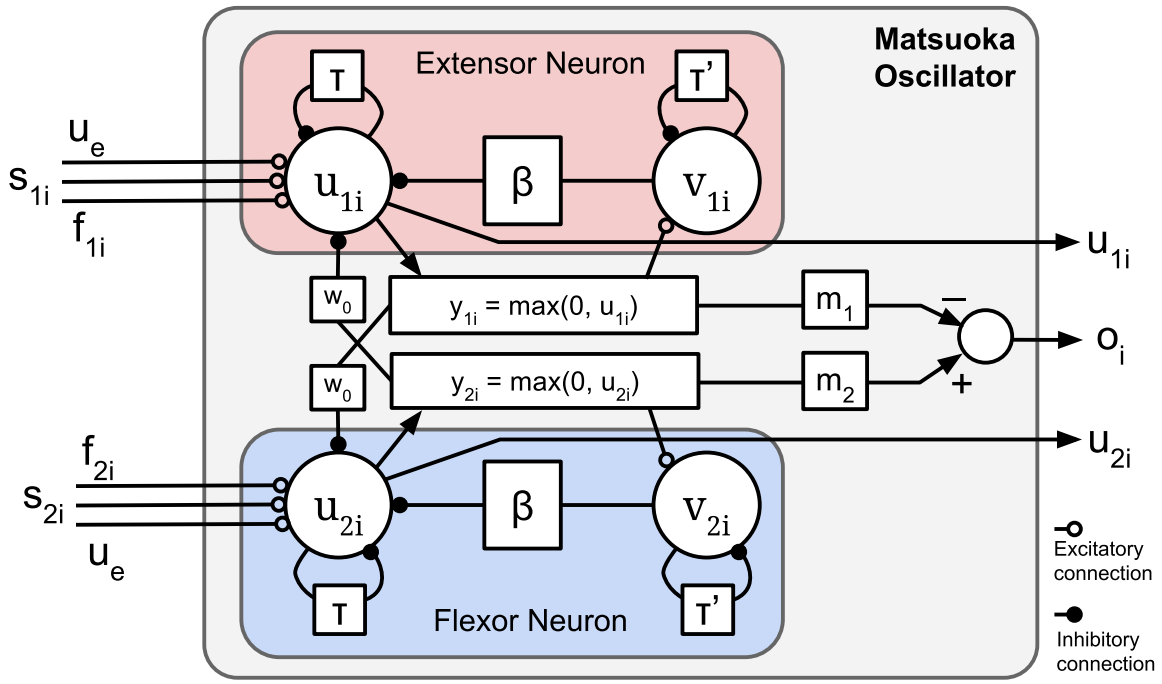


Figure 2.3: A Matsuoka oscillator. Symbols used from equations 2.3-2.7. Adapted from [43].

The configuration of an oscillator, with two mutually inhibiting neurons acting together as a unit, is one out of several possible configurations. In [66], Matsuoka has presented a number of different configurations including oscillator units comprising of 2, 3, 4 or 6 neurons with different types of interconnections. The features of the various oscillator models are also discussed. The 2-neuron model has 2 neurons acting in anti-phase and by subtracting the activation of one neuron from the activation of the other, a periodic signal can be obtained. In other scenarios, it may be desirable to have signals with intermediate phase differences and this is possible if the other configurations are utilized. Some examples, where different types of oscillator configurations have been used for robotics related applications, are [35], [11], [12] and [10].

2.2.3 Choice of Oscillator

Both the Hopf oscillator and the Matsuoka oscillator provide a number of features which makes them a suitable choice for robot locomotion tasks.

The Hopf oscillator is well defined in terms of amplitude and frequency, is easy to control and has strong limit cycle properties [52]. Moreover, different oscillators can be easily combined to produce complex output patterns consisting of multiple frequency components [79]. However, one disadvantage is that the Hopf oscillator needs an appropriate teaching signal to produce a suitable pattern. In [79], the authors have used the joint angle patterns produced by the manufacturer-provided walking algorithm as the teaching signal to train the oscillator network. The problem with this approach is that the CPG based walking algorithm will produce the same gait as the default algorithm, and will also replicate any deficiencies of that algorithm. Furthermore, for a new robot, a default walking algorithm may not be available. In [52], the author has used a system of optimized sine generators as the teaching signal for a network of Hopf oscillators. Here, extra effort is spent in optimizing the sine equation parameters, and additionally, the teaching signals themselves are very simple and cannot efficiently utilize the robot's dynamics.

The Matsuoka oscillator does not require any teaching signal. Suitable values for the parameters of the oscillator can be found by using an optimization process where the fitness function is defined by a high-level performance criterion, such as the speed or stability during walking [31]. Stochastic population-based evolutionary algorithms such as Particle Swarm Optimization (PSO) [15, 50] or Genetic Algorithms (GA) [28] are popular choices for finding optimal parameters for the oscillators. In [53] and [36] PSO variants have been used for optimizing Matsuoka oscillators. In [11] and [35], a GA was used for the same purpose. Additionally, Matsuoka oscillators can be entrained with the natural dynamics of the robot [52]. The disadvantage of using Matsuoka oscillators is that the use of optimization algorithms necessitates a high reliance on simulators and sometimes, finding a near-optimal solution may take a long time [31].

Since the NICO robot does not have a walking algorithm, the question of a ready-made teaching signal does not arise. Moreover, since a simulation model of the NICO has already been developed, using a simulator was not a concern. Hence, keeping in mind the relative advantages of the Matsuoka oscillator, it has been chosen as the oscillator model for implementing the CPGs in this thesis.

2.3 Genetic Algorithms

Genetic algorithms, introduced by Holland [28], are a class of evolutionary optimization techniques. The guiding principle of genetic algorithms, as in most evolutionary techniques, is to find an optimal solution to a problem using the process of natural selection. Given a set of individuals (population), environmental pressure causes selection of fitter individuals. This causes the less fit individuals to eventually die out, resulting in an overall increase of the fitness of the population [16].

The general scheme of this process is depicted in figure 2.4.

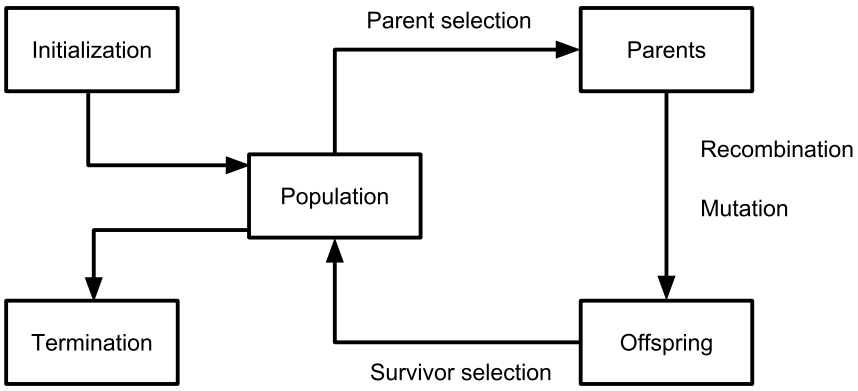


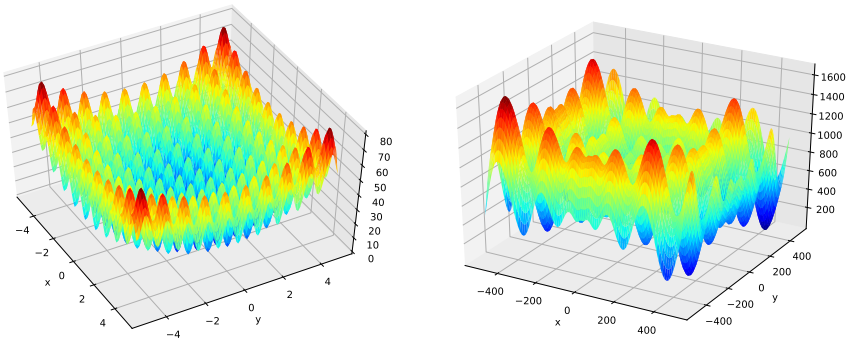
Figure 2.4: Flowchart of an evolutionary algorithm [16]

Genetic algorithms have seen used extensively in science and engineering, mainly because of their efficiency in handling problems that cannot be solved using regular techniques such as gradient-descent. The solutions generated by genetic algorithms are often unintuitive for humans, but nonetheless, highly effective. The sequence of steps and features of a simple genetic algorithm are discussed next.

Problem Representation

The first step in a genetic algorithm is the process of representing a real-world problem in a form that can be understood by the genetic algorithm. This involves abstraction and simplification of the problem statement into an objective criterion that needs to be optimized. This objective criterion is referred to as the 'fitness function' [16]. The fitness function defines a surface in an n-dimensional space, where n is the number of parameters in a potential solution. The objective of the optimization process is to find the location of the optimum on the surface defined by the fitness function. Fitness functions which represent uniform uni-modal surfaces (a single optimum) can be easily optimized using gradient-descent (-ascent) algorithms. However genetic algorithms are particularly useful when the fitness surface is highly irregular with multiple local optima or is discontinuous. In most cases, the shape of the fitness surface is not known at all. For a 2-dimensional search space, a few examples of irregular multi-modal fitness surfaces are shown in figure 2.5. These surfaces were generated using benchmark functions which are generally used to evaluate the performance of optimization algorithms [62, 83, 77].

The next important issue is the encoding of possible solutions. In the 2D case, for the fitness surfaces in figure 2.5, the solution is the coordinate pair (x, y). One way of encoding the solutions is to use real valued numbers for x and y. Another way is to use binary bit strings for the values of x and y. Based on



(a) Rastrigin Function [77] (b) Schwefel Function [77]

Figure 2.5: Irregular multi-modal fitness surfaces

the problem, many other kinds of encoding schemes can also be used, such as integer representation, permutation representation or tree representation [16]. The observable traits of the possible solutions are called phenotypes and the encoded forms of those solutions (such as a binary bit string representing (x, y)) are referred to as genotypes. In the genetic algorithm parlance, individual genotypes are also called chromosomes or individuals and the elements inside them are called genes or alleles [16].

Population

The population consists of a set of chromosomes, and forms the unit of evolution. All the operations of a genetic algorithm are performed at the population level. Usually, the population contains a fixed number of individuals and this number remains constant throughout the process of evolution. As evolution proceeds, less fit individuals are slowly replaced by fitter individuals due to the selection pressure created by the parent selection and survivor selection operators [16]. It is a common practice to initialize the individuals of the first population by using random values within a predefined range. There also exist techniques where quasi-random sequences such as Sobol sequence [88] or Van der Corput sequence [25] are used to make the individuals of the initial population spread out uniformly and cover as much of the search space as possible.

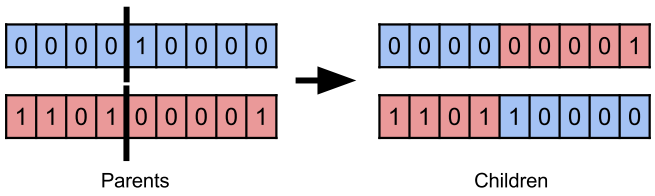
Parent Selection

The parent selection operator is responsible for selecting individuals from a given population which will be used to produce new individuals for the next population. Parent selection is a stochastic process where fitter individuals have a higher probability of being selected as parents. However, individuals with low fitness also have a non-zero probability of being selected as parents, so as to prevent the optimization process from becoming greedy and getting stuck at a local optimum [16]. Well known strategies of selection include roulette-wheel selection, tournament selection and ranking selection [23]. The basic idea behind the selection of fitter individuals

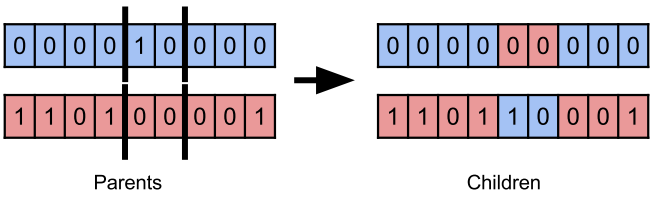
as parents is that genotypes which resulted in successful phenotypes should be propagated to successive generations.

Recombination and Mutation

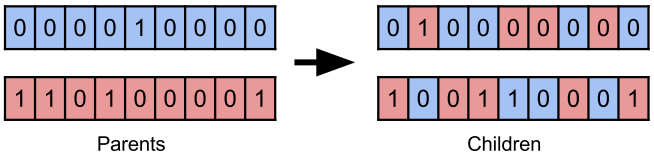
The purpose of recombination and mutation is to create new individuals from old ones. Recombination is a binary operation where two parent chromosomes are taken and parts of their chromosome are swapped to produce two child chromosomes. This process is also known as crossover and a number of different types of crossover are possible based on the position and number of gene-sequences that are swapped. In a one-point crossover scheme, a single position for crossover is marked in the two parent chromosomes. In an n-point crossover scheme there are n such positions [16]. Another type of crossover strategy is to take each gene of a child chromosome at a time and randomly choosing which parent that particular gene should be inherited from. This is known as uniform crossover [94]. The different types of crossover operations are depicted in figure 2.6.



(a) 1-point crossover



(b) n-point crossover with n=2



(c) Uniform crossover

Figure 2.6: Types of crossover operations [16]

Mutation is a unary operation in which a single chromosome is taken and individual genes are randomly altered to produce a new chromosome. The exact way in which a gene is mutated depends on the encoding scheme used. For example, when binary encoding is used, it is common practice to randomly flip 0s to 1s and

vice versa. When floating-point representation is used, a gene can be mutated by adding or subtracting a floating point number (randomly drawn from some probability distribution) from the number inside the gene. Several other varieties of mutation also exist [16]. The mutation operation is important for introducing diversity into a population and helps the algorithm to explore the search space more efficiently. However, mutation also has the tendency for disrupting good chromosomes and so it should be used rather sparingly. Like recombination, mutation is also a stochastic process and a small probability is used to decide whether a particular gene should be mutated or not.

Survivor Selection

Survivor selection, like parent selection, also aims to distinguish between individuals based on their fitness. This is especially important for genetic algorithm variants which produce more than 2 children by recombining 2 parents and is necessary for maintaining a constant population size. Selection of individuals is a stochastic process, where the probability of selection is dependent on the fitness or age of an individual. One approach for selection, while passing from one generation to the next, is to consider all the parent and child individuals together and then to select a constant number of individuals from this set. Another approach, known as elitism, is to always clone a certain number of the fittest individuals into the next generation and then perform selection to fill the remaining spots. This ensures that high-quality individuals are never lost during the evolution process [16].

Termination

The termination criterion makes certain that the process of evolution does not continue indefinitely. A commonly used termination criterion is to terminate the process if there is no fitness improvement in a small fixed number of consecutive generations. An easier alternative is to use a fixed number of generations for which to run the algorithm [16].

A number of variants of genetic algorithms are possible due to the large number of choices which can be made regarding the operations such as selection, mutation and recombination. In addition, there also exist many hybrid approaches, where the features of genetic algorithms and other kinds of evolutionary algorithms have been combined [48, 37, 85].

2.4 Policy Gradient Reinforcement Learning

The concept of a goal-directed agent, which can learn to perform tasks by interacting with the environment, is especially useful in robotics. For example, if a robot is being trained to grasp an object, it is infeasible to provide label actions such as target joint angles for each state that it encounters (as needed in supervised learning). However, it is very easy to tell the robot how good it is performing by

giving it a positive reward if it completes the task successfully or some negative reward if it hits any obstacles. A special class of reinforcement learning, known as policy gradient methods, is especially suited to robotics because these methods are able to deal with continuous, real-valued action spaces, which is a characteristics of physical environments. Policy gradient-based methods have been successfully used for robotics applications in [21], [64], [17] and [57]. Examples of other applications can be found in [56]. This section aims to provide a general overview of policy gradient methods, with a focus on the Deep Deterministic Policy Gradient algorithm (DDPG) [61], which has been used in this work. However, before that, a brief overview of reinforcement learning is presented, so that the terminology used in the remainder of this section can be introduced.

2.4.1 Basics of Reinforcement Learning

Reinforcement learning is concerned with the problem of learning to control an agent, operating within some environment, so as to maximize a long-term performance measure [95]. The entity responsible for taking decisions and performing actions is called the **agent** and the thing that it interacts with, comprising of everything outside the agent, is called the **environment**. The agent monitors the current **state** of the environment and based on that decides to perform an **action**. The environment reacts to that action and presents the agent with a **reward** (a numerical performance measure) and a new environment state [92].

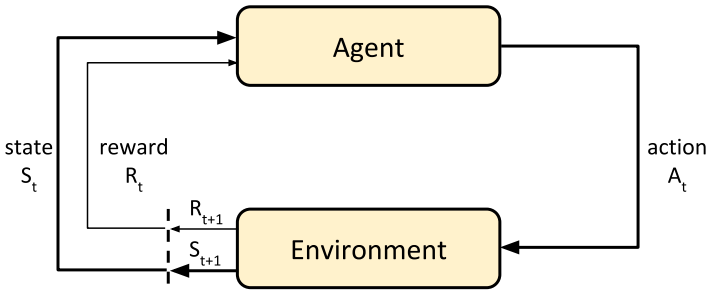


Figure 2.7: The reinforcement learning scenario [92]

This process is depicted in figure 2.7, where at time t the agent observes the environment's state $S_t \in \mathcal{S}$ (where $\mathcal{S}$ is the set of all possible states) and the last reward $R_t \in \mathbb{R}$ and decides to perform an action $A_t \in \mathcal{A}(S_t)$ ($\mathcal{A}(S_t)$ is the set of actions which can be performed in state S_t). In the next time step $t+1$, the environment reacts by transitioning to a new state S_{t+1} and sending a new reward $R_{t+1} \in \mathbb{R}$. The agent's goal in the reinforcement learning scenario is to maximize not the immediate reward that it receives, but the cumulative reward in the long run. The cumulative reward which an agent receives is known as the **return**. Specifically, return at time t is the expected cumulative reward starting from time $t+1$ upto the terminal time step T. This is denoted in equation 2.8.

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T \tag{2.8}$$

In most situations, it is useful to give more importance to immediate rewards than rewards which the agent expects to achieve many time steps in the future. This is taken care of by using a **discounting** factor. The agent tries to maximize the expected discounted return G_t, denoted in equation 2.9, where γ is a parameter, $0 \leq \gamma \leq 1$, called the **discount rate** [92].

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.9}$$

The sequence of all states, actions and rewards of an agent from time $t = 0$ till the current time is known as its **history**, H_t, denoted by:

$$H_t = S_0, A_0, R_1, ..., S_{t-1}, A_{t-1}, R_t, S_t, A_t \tag{2.10}$$

One of the most important concepts in reinforcement learning is that of the **Markov property**. When the agent takes an action, the environment's response at $t+1$ may depend on the entire history, that is, the probability of the next state S_{t+1} and reward R_{t+1} depends on H_t. However, if the state has the Markov property, then the next state is only dependent on the current state and action, not the entire history. In this case the state is said to be a **Markov state**. Since it is desirable for states to be a good basis for predicting what the future rewards and future states will be, the assumption of Markov property is a fundamental theoretical consideration in reinforcement learning. In practical situations, where this assumption may not hold completely, it is useful to consider a state as an approximation of a Markov state, in order to get good performance from reinforcement learning systems [92]. A reinforcement learning task that satisfies the conditions of the Markov property is known as a **Markov Decision Process** (MPD).

In a reinforcement learning task, an agent needs to have a way to choose an action out of many possible actions, when it finds itself in a particular state. It does so by following its **policy**, which is defined as a mapping from states to actions. Policies can be deterministic (where the agent takes the same action whenever it sees the same state), or they can be stochastic (where the agent is most probable to take a certain action in a given state, but not always). Stochastic policies are useful since they allow the agent some room for exploration. For a state $s \in \mathcal{S}$ and action $a \in \mathcal{A}(s)$, a deterministic or stochastic policy π is defined as [92]:

$$\text{Deterministic Policy:} \quad a = \pi(s) \tag{2.11a}$$

$$\text{Stochastic Policy:} \quad \pi(a|s) = \mathbb{P}[A = a \mid S = s] \tag{2.11b}$$

A basic idea in reinforcement learning is to predict how good a certain state is, based on the expected return of that state. The agent then tries to choose actions which result in good states and hence better returns. The goodness of a state is determined by using a **value function**. A value function can be of two types: state-value function and action-value function. The **state-value function** $v_\pi(s)$

of a state s, under a policy π, is defined as the expected return when starting in state s and following policy π thereafter [92].

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \;\middle|\; S_t = s\right] \tag{2.12}$$

The value of taking an action a in a state s and thereafter following policy π is defined by the **action-value function** $q_\pi(s, a)$ [92].

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \;\middle|\; S_t = s, A_t = a\right] \tag{2.13}$$

Value functions satisfy a recursive relationship between the value of a state s and the value of its potential successor states. This recursive relationship is defined by the **Bellman equations**, which form the basis of numerous theoretical proofs in reinforcement learning [92].

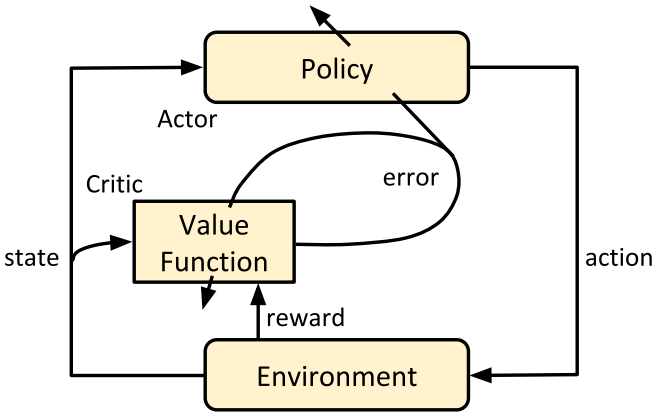


Figure 2.8: Actor-critic reinforcement learning, where the policy is learned separately and is critiqued by the value function [92].

Based on the evaluation of values or policies, reinforcement learning methods can be categorized into different classes. **Value-based** methods use the value function of state-action pairs and then use those action values directly to select actions. They do not explicitly evaluate policies. **Policy-based** methods, on the other hand, directly represent policies and use them to select actions. In a third category, known as **actor-critic** methods, both value functions and policies are used. The policy (known as the actor) is represented independently of the value function (known as the critic). The policy is used to take actions and the value-function is used to critique the policy being followed [92]. The general scheme of actor-critic methods is shown in figure 2.8.

2.4.2 Policy Gradient Methods

When a reinforcement learning agent needs to decide what action to take, it has a number of ways of making that decision. One way is to compute the action-value function $q_\pi(s, a)$ for the current state s and all possible actions which can be taken in that state, and then use the computed value of the action-value function to form the policy as $\pi(s) = \underset{a \in \mathcal{A}(s)}{\operatorname{argmax}}\, q_\pi(s, a)$.

Using such a greedy approach does not allow any exploration. A better approach is to form a stochastic policy by using an ϵ-greedy method, where the best action is selected with a probability of $(1 - \epsilon)$ and a random action is selected with a probability of ϵ (a small probability). However, this process of generating the policy by using the value-function has one major problem: the use of the argmax operator is only possible if the actions $a \in \mathcal{A}(s)$ are discrete. If the action space is continuous, to find the argmax, an optimization needs to be performed at every step. This is clearly not feasible for an agent operating in real time [61].

The above mentioned limitation can be avoided if the action is generated directly by using the policy. In **policy-based methods**, the policy is parameterized using some parameter θ. As long as the policy $\pi_\theta(a|s)$ can be differentiated with respect to its weights ($\nabla_\theta \pi_\theta(a|s)$ exists and is finite), any kind of parameterization can be used. For example, a neural network, with a weight vector θ, can be used to predict actions. In the case of continuous actions, the input to this network will be the state and the output will be the action. For a discrete action space, the output of the neural network can be converted into probabilities of the actions by using the softmax function [93].

Given a policy $\pi_\theta(a|s)$ with parameters θ, the goal of policy-based learning is to find the best possible θ. To determine the quality of the policy π_θ a **policy objective function**, $J(\theta)$ is used [93].

$$J(\theta) = \begin{cases} v_{\pi_\theta}(s_1) & \text{for a discrete state space} \\ \sum_s d_{\pi_\theta}(s) v_{\pi_\theta}(s) & \text{for a continuous state space} \end{cases} \tag{2.14}$$

where $v_{\pi_\theta}(s)$ is the state-value function of state s, s_1 is the initial state in the discrete case and $d_{\pi_\theta}(s)$ is the probability of being in state s in the continuous case. **Policy-gradient based methods** seek to find the θ that maximizes $J(\theta)$ in equation 2.14 by using the gradient $\nabla_\theta J(\theta) = [\frac{\partial J(\theta)}{\partial \theta_1}, ..., \frac{\partial J(\theta)}{\partial \theta_n}]^T$. Typically methods such as gradient-descent are used for this. Further details and mathematical proofs related to policy gradients can be found in [93].

2.4.3 Deep Deterministic Policy Gradient

Policy gradient algorithms are widely used in reinforcement learning applications, where the policy is directly used to select an action. In [87], Silver et al. presented Deterministic Policy Gradient algorithms (DPG), where **deterministic policies** such as $a = \mu_\theta(s)$ are used (μ_θ is a function parameterized by θ, a is the action and s is the state). In order to encourage exploration, the authors have used an **off-policy** strategy, where a stochastic behavior policy is used to learn about a deterministic target policy [87]. This means that during training, variability is introduced into the action selection process, but once the training is over, the learned policy does not have any stochasticity in action selection. This allows the target policy to utilize the knowledge gained by the behavior policies. The performance benefits of deterministic policy gradients, particularly in high-dimensional tasks, has been cited as a reason for their preference over stochastic policy gradients [87].

One of the most famous papers in reinforcement learning is [72], in which the Deep Q Network (DQN) was introduced by Mnih et al. DQN was able to match or surpass human-level performance on many Atari video games using unprocessed pixels as input. Here, a deep convolutional neural network was used to approximate the optimal action-value function $q^*(s, a)$ [72]. However, since the action-value function is used as the basis for selecting actions, DQN, in its original form, generally cannot be used in robotic tasks, which are most often characterized by high-dimensional continuous action-spaces.

The Deep Deterministic Policy Gradient (DDPG) algorithm, introduced by Lillicrap et al. [61], overcomes this limitation. DDPG is a model-free, off-policy, actor-critic algorithm which uses deep neural networks as function approximators for the policy and the action-value functions. DDPG is based on the DPG algorithm [87], and also uses concepts from DQN to stabilize the learning process.

Mathematical Preliminaries

The DDPG algorithm deals with an agent interacting with an environment E in discrete time steps $t_1, t_2, ...t_T$. At each time step t the agent receives an observation s_t, takes an action $a_t \in \mathbb{R}^N$ and receives a scalar reward r_t. A policy π defines the agent's behavior. The state, which may be stochastic, is modeled as a Markov Decision Process with a state space $\mathcal{S}$, action space $\mathcal{A} \in \mathbb{R}^N$ and reward function $r(s_t, a_t)$. A discounted return $R_t = \sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i)$ is used for judging the quality of a state ($\gamma \in [0, 1]$ is the discounting factor). The goal is to learn a policy which will maximize $J = \mathbb{E}_{r_i, s_i \sim E, a_i \sim \pi}[R_1]$, the expected return from the starting state. The probability distribution which defines the way in which states are visited, using policy π, is denoted as ρ^π [61].

The Bellman equation for action-value functions for a policy π, relates the action-value of one state-action pair to the next [61, 92]:

$$q_\pi(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}\big[r(s_t, a_t) + \gamma \mathbb{E}_{a_{t+1} \sim \pi}[q_\pi(s_{t+1}, a_{t+1})]\big] \tag{2.15}$$

As the target policy is deterministic, a function $\mu : \mathcal{S} \leftarrow \mathcal{A}$ is used to describe the policy. Due to this, the inner expectation in equation 2.15 can be avoided and it can be rewritten as [61]:

$$q_\mu(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}\big[r(s_t, a_t) + \gamma q_\mu(s_{t+1}, \mu(s_{t+1}))\big] \tag{2.16}$$

The only stochasticity which remains is due to the environment E. Hence, it is possible to learn q_μ off-policy, by using a separate stochastic behavior policy β to learn the deterministic target policy μ. The DDPG algorithm uses function approximators parameterized by θ^q and these are optimized by using the following loss function in equation 2.17. Here although the target y_t is also dependent on the parameter vector θ^q, this is ignored [61].

$$L(\theta^q) = \mathbb{E}_{s_t \sim \rho^\beta, a_t \sim \beta, r_t \sim E}\Big[\big(q(s_t, a_t \mid \theta^q) - y_t\big)^2\Big] \tag{2.17}$$
$$\text{where } y_t = r(s_t, a_t) + \gamma q(s_{t+1}, \mu(s_{t+1}) \mid \theta^q)$$

Similar to the DPG algorithm of Silver et al. [87], the DDPG algorithm also uses a parameterized actor function $\mu(s|\theta^\mu)$ for deterministically mapping states to specific actions. The actor is updated by using the policy gradient denoted in equation 2.18. The proof for this can be found in [87].

$$\nabla_{\theta^\mu} J \approx \mathbb{E}_{s_t \sim \rho^\beta}\big[\nabla_a q(s, a|\theta^q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s=s_t}\big] \tag{2.18}$$

Highlights of DDPG

The DDPG algorithm uses concepts from DPG, as well as DQN. The chief features of DDPG, necessary for understanding this algorithm, are:

Replay buffer Optimization algorithms for neural networks assume the data samples to be independently and identically distributed (i.i.d). In a reinforcement learning scenario, the data samples are generated due to the agent's sequential actions, and so the samples are highly correlated. A replay buffer, introduced in [72], is also used by the DDPG algorithm to address this issue. A replay buffer is a fixed-size, first-in-first-out (FIFO) memory cache where transitions sampled from the environment are stored. At each time step the tuple (s_t, a_t, r_t, s_{t+1}) is saved. The actor and critic networks are updated by uniformly sampling a minibatch from this replay buffer. This helps to obtain more i.i.d-like data samples for training [61].

Use of target networks Directly using equation 2.16 for updating the critic network most often leads to divergence of the network parameters. To solve this issue, copies are created of the actor and critic network and the copies (target networks) are used during the updates. The weights of a target network, θ' are updated by letting them slowly track the learned network weights θ using the relation $\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$ where $\tau \ll 1$. This constrains the target values to change slowly and improves the stability of the learning process [61].

Batch Normalization Different elements of the observation may have different units and ranges which makes the learning process difficult. Batch normalization [33] is used to normalize each dimension across the samples in a minibatch to have unit mean and variance [61].

Exploration Being an off-policy algorithm, DDPG uses a stochastic behavior policy which is separate from the learned target policy. An exploratory policy μ' was created from a deterministic policy μ by adding noise sampled from a noise process $\mathcal{N}$ to the actor policy. This was done using $\mu'(s_t) = \mu(s_t|\theta_t^\mu) + \mathcal{N}$, where θ_t^μ is the parameter vector of the actor network at time t . The noise process used here was the Ornstein-Uhlenbeck process [97] [61].

DDPG Algorithm

The DDPG algorithm is able to handle high-dimensional continuous action spaces, is computationally fast, and is relatively easy to train and implement. For these reasons, it was chosen as the learning algorithm for training the high-level controller in this thesis. The complete DDPG algorithm is listed below:

Algorithm 1 DDPG algorithm [61]

Randomly initialize critic network $q(s, a|\theta^q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^q and θ^μ.
Initialize target networks q' and μ' with weights $\theta^{q'} \leftarrow \theta^q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t, observe reward r_t and new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - q(s_i, a_i|\theta^q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a q(s, a|\theta^q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{q'} \leftarrow \tau\theta^q + (1-\tau)\theta^{q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Chapter 3

Related work

Many kinds of techniques have been used by researchers for developing algorithms for bipedal locomotion. Although it is difficult to strictly classify these techniques into groups, an approximate classification helps to organize the discussion and to understand the salient features of the different techniques. A modified form of the classification scheme presented by Al-Shuka et al. [1], is used in this chapter to discuss the different kinds of methods for bipedal locomotion as a whole. This is not an exhaustive classification and other ways of grouping methods may exist, but the presented break-up covers some of the important contributions in robotic bipedal locomotion. After a brief discussion of each class of methods, relevant studies of that class are discussed. Finally, in the last section, the relative merits and demerits of each type of approach are analyzed.

Bipedal locomotion techniques can be broadly classified into model-based, natural dynamics-based and biologically-inspired methods. Each of these categories can be further broken up into different subcategories, as shown in figure 3.1.

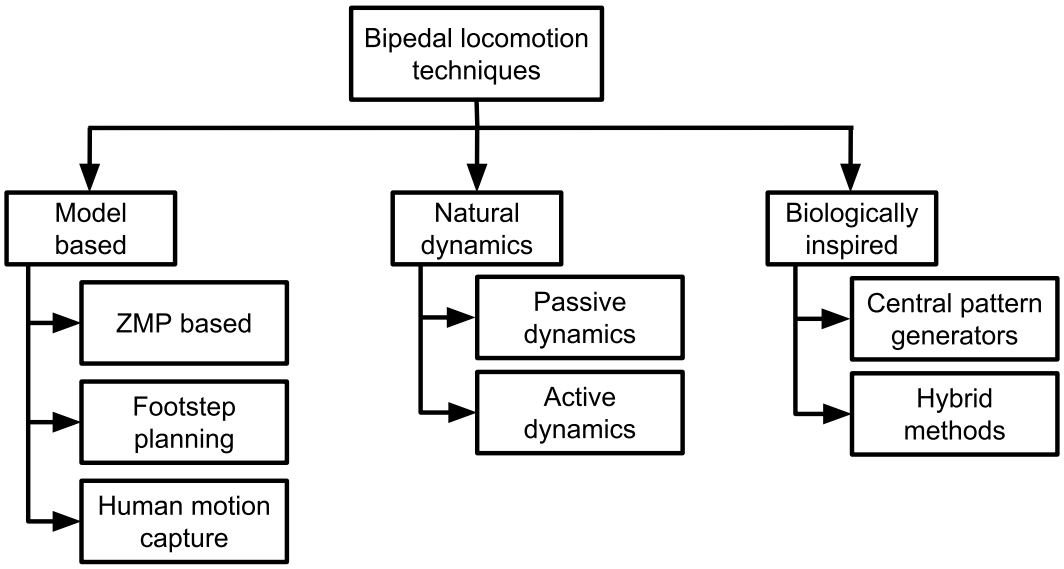


Figure 3.1: Types of bipedal locomotion techniques. Modified from [1].

3.1 Model-based Methods

Model-based techniques use analytical models in order to generate reference trajectories for the robot to follow [1]. Typically, trajectories are generated in 3-dimensional space, to be followed by some reference point on the robot's body (such as the center of mass or the center of the foot). Using inverse kinematics, joint angle trajectories are generated from the reference trajectory. A common feature of this kind of approach is the stabilizer, which makes adjustments in order to make sure that the reference trajectory is being followed as closely as feasible. The basic framework, shown in figure 3.2, consists of three main components. The pattern generator generates the reference trajectories by using the model's behavior. The robot tries to follow the trajectories, and the stabilizer closes the feedback loop by making necessary adjustments to the target trajectory [39].

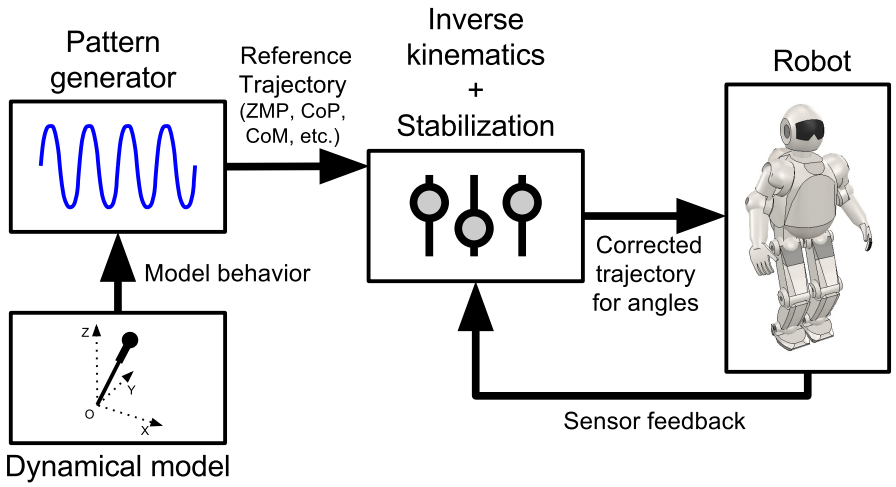


Figure 3.2: Basic control framework for model-based methods. Adapted from [39].

3.1.1 Zero-Moment-Point based Methods

The idea of zero-moment-point (ZMP) was introduced by Vukobratović and Stepanenko in 1972 [100]. The first practical application of this concept in a biped robot was made by Kato et al. on the robot WL-10RD in 1985 [34]. Since then, ZMP has been one of the most popular methods employed for bipedal locomotion [86].

One of the critical tasks for a biped robot is to maintain balance or stability. ZMP specifies the criteria for maintaining dynamic stability while walking. It is defined as the point on the ground at which the net moment of the inertial forces and gravity forces has no component along the horizontal axes. In order to be dynamically stable, the ZMP must lie within the convex hull of the polygon created by the ground contact points of the robot (support polygon) [98]. Mathematical details of the ZMP criteria for stability can be found in [98, 86, 39].

In the original paper for ZMP [100], the idea proposed was to move the robot's joints according to some predetermined angle trajectories and then to control the

position of the ZMP by using a compensating mass. The basic modern approach first defines a foot trajectory on the ground and then determines the positions of the support polygon for the duration of the walk. Then a ZMP trajectory (shown as the reference trajectory in figure 3.2) is defined such that the ZMP lies within the support polygon at all times. Finally, inverse kinematics is used to calculate the joint angles which are necessary to make the body follow the prescribed ZMP motion [86]. The stabilizer continuously monitors the position of the actual ZMP and if it deviates from the reference ZMP position, the joint angle trajectories are appropriately modified to minimize the deviation.

Thus, if the function ZMP is defined as $(p_x, p_y)^T = ZMP(q, \dot{q}, \ddot{q})$ (where $q, \dot{q}, \ddot{q}$ are vectors of the joint positions, velocities and accelerations respectively, and (p_x, p_y) defines the x and y coordinates of the ZMP), the ZMP pattern generation problem is to find a feasible $q(t)$ that will satisfy the desired foot step trajectory $p_x^d(t), p_y^d(t)$ (where (p_x^d, p_y^d) are the desired coordinates of the ZMP) [86].

To solve this problem, Sugihara et al. [91] have utilized the inverted pendulum model. They developed a motion generation method for controlling the trajectory of the center of gravity (COG) of the robot by indirectly controlling the ZMP in real time. By abstracting the robot model by an inverted pendulum, whose point of location is given approximately by the ZMP, the authors were able to utilize the dynamics equations of the pendulum to determine the trajectory of the COG. Joint angles were computed using inverse kinematics.

An important idea related to ZMP is that of preview control, proposed by Kajita et al. In [40], the authors have used a cart-table model to abstract the model of a biped robot. They used the equations of the cart-table model and a feedback controller for tracking the desired ZMP trajectory. A special feature of this technique is that future information about the ZMP location is used to move the center of mass even before the ZMP has changed position. The authors compare this process of using 'future information' (known as preview control) as driving a car on a curved road and adjusting for the turn in the road even before the car has reached it [39]. In other work, Kajita et al. have developed a method for posture and force control of a biped robot by using a linear inverted pendulum model (LIPM) and ZMP [42]. This method was used for stabilizing the walk of a 42 degree-of-freedom (DOF) robot called HRP-4C. In [41], Kajita et al. have used a 3-dimensional LIPM model for generating the motion pattern of a 12 DOF robot.

In more recent work, Yu et al. [103] have developed a controller for rejecting external disturbances in real-time by using ZMP regulation and preview control. They used a modified version of the LIPM, known as the flywheel inverted pendulum, for simplifying the dynamics of the humanoid robot. In a related work, Kamogawa et al. have developed a stability controller for a biped humanoid robot by using ZMP control and the LIPM model [44].

3.1.2 Footstep Planning

Footstep planning methods focus on finding a trajectory or path (for the robot's feet) to the goal location. Although methods such as ZMP-based control take

care of stability related aspects of walking, they do not explicitly deal with the problems of obstacles or uneven terrain. This is taken care of by methods for footstep planning. Often, however, footstep planners also utilize the ZMP criterion for trajectory generation and control.

In [58], Kuffner et al. have presented an algorithm for planning footsteps by using a forward dynamics approach. Starting from a set of possible foot locations, the algorithm computes the sequence of steps to a specified goal. Heuristic-based cost functions are used to search for optimal foot placements. A stereo vision system is utilized for obstacle detection. In [76], Perrin et al. developed a bounding box based method for footstep planning. Their method can be used with rigid body motion planners and takes advantage of the stepping-over capabilities of the robot.

In [59], footstep planning was used by the Atlas robot to walk over non-flat terrain. To plan a sequence of safe footsteps, first a 'light detection and ranging' (LIDAR) scan was used to identify obstacles. Then the feasible locations of the next footsteps were computed. In the end, a center of pressure (COP) trajectory was created by using the footstep locations, and this trajectory was passed as input to the controller responsible for following it.

Bipedal robots need to reject disturbances while maintaining the desired gait. To achieve this, Missura and Behnke proposed a method which uses a combination of footstep planning and ZMP strategies [70]. The tasks of balance control and motion generation are separated to form a hierarchical framework. The balance controller, which receives a reference trajectory as input, computes the timing and position of the next footstep along with the desired ZMP location. These are fed as input into the motion generation layer which is responsible for producing the actual joint motions. The balance controller is able to deal with strong disturbances and modifies the footstep and ZMP target locations accordingly. This work was based on the LIPM model. In [71], the same authors have presented a similar approach for online learning of foot placement for a biped robot.

3.1.3 Human Motion Capture

Another approach for model-based bipedal locomotion is to record human motion-capture data and use it to generate target trajectories for the robot. For recording human data, special markers may be attached to the limbs and joints of human subjects, whose gait is recorded as they walk on treadmills. Recent motion capture setups can even work without markers. The video recordings are then processed to extract the coordinates of the markers and from this, the velocity, acceleration and other information about the movement of the different joints can be obtained. This information is then converted into target trajectories which a robot can follow (either joint trajectories or reference point trajectories such as the hip movement) [1]. As in other model-based approaches, this method also involves the generation of reference trajectories and the use of controllers to minimize the errors between the actual and reference trajectories.

This approach was used by Hemami and Farnsworth [26] to analyze the gait stability of a 5-link planar biped. In [13], Danilov et al. have used a Kinect-based

motion capture setup to record biomechanical human gait data. From this, the ZMP and COM trajectories were computed, with the objective of using these as reference trajectories for the locomotion of a biped robot. In [2], Ames has used human motion data to create a low-dimensional representation of human walking. This was used for locomotion of two kinds of robots - an underactuated planar biped and a fully actuated 3-dimensional robot.

3.2 Natural Dynamics-based Methods

Unlike model-based methods, dynamics-based methods do not require predefined reference trajectories which the robot must follow [1]. Instead, they try to utilize the natural dynamics of the robot for achieving an energy efficient gait. Here the focus is more on the mechanical design of the robot and how this design can be exploited to the fullest possible extent. A general framework of this kind of methods is shown in figure 3.3. Here, a high-level controller monitors overall gait-related objectives, whereas, things such as low-level feedback control are taken care of by specialized mechanical structures. For purely passive dynamics-based machines, naturally, there is no controller and no active actuation of the joints.

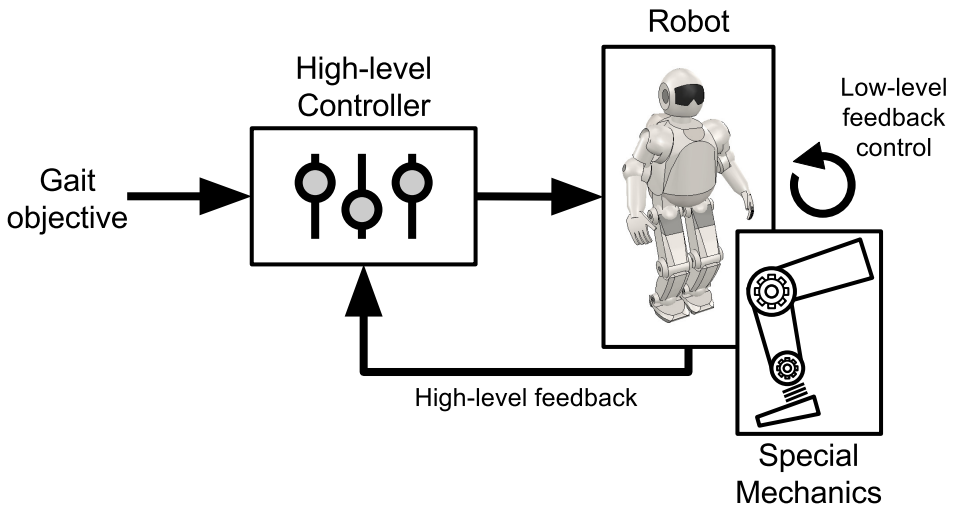


Figure 3.3: Basic control framework for natural dynamics-based methods, inferred from the survey of related literature, such as [7, 68, 8].

3.2.1 Passive Dynamics

Mainstream control paradigms, such as model and ZMP-based methods, rely to a great extent on precise joint control, requiring actuators with higher precision and frequency response than what human muscles can provide. These methods also seek to actively control every joint at all times. This is also unlike human locomotion. The energy consumption of robots using traditional control mechanisms is also much greater than that of human walking [7].

An interesting alternative approach to building walking machines was proposed by McGeer [68]. He discussed the existence of walking machines which only rely on natural dynamics, without any active control mechanism or energy input. McGeer's passive walkers could exhibit a stable walking cycle while walking down an incline under the effect of gravity. Most importantly, the gaits of these machines were very similar to the way humans walk. Although a passive walking machine resembles a toy more than it does a robot, the study of such machines is important. These machines have shown that when natural dynamics is utilized effectively, fast, agile and human-like motion is possible in robots, without having to rely on precise, active control. Once a suitable mechanism for walking has been developed, adding some form of actuation to it requires only minor modifications [68].

To show that passive dynamic walkers can be converted to actuated models and made to walk on level ground, Collins et al. developed three robots similar to McGeer's passive walkers, but with a simple form of actuation [7]. In order to measure the efficiency of these simple robots, the authors used a dimensionless parameter known as the specific cost of transport, given by the relation $c_t = (energy\ used) \div (robot\ weight \times distance\ travelled)$. The efficiency of all the three robots was found to be in the same range as of humans. The efficiency of the Asimo robot, using the same measurement parameter, was ten times less [7]. Another important outcome of this study was related to the complexity of control algorithms. The authors showed that complex algorithms, requiring a substantial amount of real-time computing may not be an absolute necessity for realizing bipedal locomotion. The robots used in this study were designed to mimic the structure and dynamics of human legs. This shows that mechanical design and control algorithm techniques are equally important when it comes to developing a bipedal robot with gait characteristics similar to humans.

3.2.2 Active Dynamics

Humanoid bipedal robots, walking around with relatively slow, careful steps are not the only kind of bipedal robots that exist today. The focus on natural dynamics and the symbiotic relationship between the mechanical and control systems have yielded robots which are capable of exhibiting dynamic, agile and energy-efficient bipedal locomotion.

The Fastrunner, a robot inspired by the ostrich, was developed by Cotton et al. [8]. The authors reported a highly efficient running gait in simulation (the simulated robot could reach a stable running speed of 35.4 kph [8]). The robot relies on a specially designed leg architecture, where each leg carries only one actuator instead of one actuator per degree-of-freedom. This greatly reduces the energy consumption of the robot. The mechanical design also contributes to the self-stabilizing capabilities of the robot. As a result, the software controller does not need to deal with low-level feedback mechanisms and is left free to deal with high-level maneuvers such as recovering from large disturbances or stepping into a special location if needed. In the simulation study, a simple sinusoidal pattern for each leg was enough to obtain a stable and fast run [8].

Another bipedal robot which utilizes its natural dynamics is ATRIAS (Assume The Robot Is A Sphere), developed by Hubicki et al. [29]. ATRIAS is a bipedal robot of human scale and uses the spring-mass model for dynamic walking and running. The spring-mass approach has helped in increasing the efficiency and agility of the robot [29].

While it is not the objective of this chapter to dwell on the aspects of mechanical design, it can certainly be pointed out that intelligent control algorithms go hand in hand with smart mechanical design towards the development of efficient bipedal robots.

3.3 Biologically Inspired Methods

Biologically inspired methods of bipedal locomotion control seek to emulate the neural locomotion control mechanisms found in vertebrates. Neural circuits, known as central pattern generators (CPG), are capable of producing coordinated patterns of rhythmic signals without the need of rhythmic inputs and can be modulated using low-dimensional control signals [31]. Artificial CPGs, discussed in section 2.2, emulate the basic behavior of natural CPGs. There exist neurobiology-inspired methods for locomotion, where recurrent or feed-forward neural networks are solely used for pattern generation (such as [78, 75]). However, in this section, the discussion will focus on only CPG-based methods and hybrid methods where concepts from model-based methods are used together with CPGs.

A basic framework for CPG-based methods can be described using figure 3.4. Here, the CPG network, using its internal parameters, generates angle trajectories for the individual robot joints. Some form of sensor feedback may be used for modulating the output of the CPGs. Typically, the CPG parameters are optimized for attaining some kind of gait-related objective such as a fast walk.

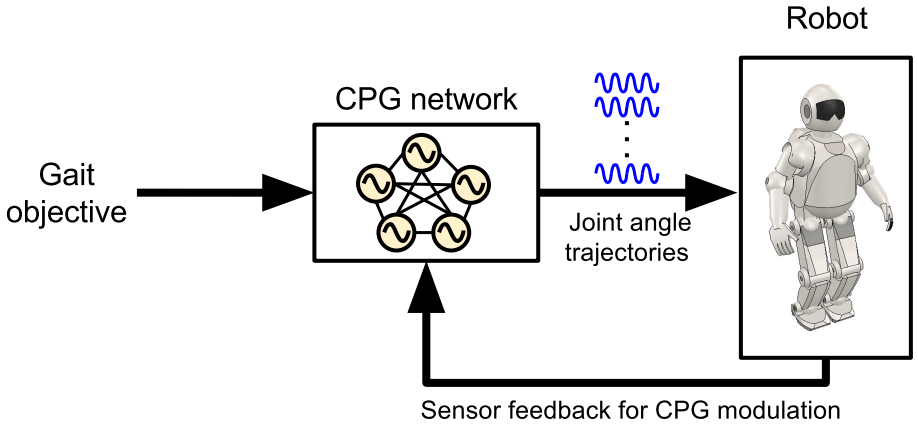


Figure 3.4: Basic control framework for CPG-based methods, inferred from the survey of related literature, such as [35, 11, 79].

3.3.1 Central Pattern Generators

The use of central pattern generators for bipedal locomotion is becoming increasingly popular [31]. The Matsuoka oscillator, which is based on Brown's half-center neuron model, has been used in many papers. The seminal work for bipedal locomotion using CPGs was done by Taga et al. [96]. They used a network of coupled Matsuoka oscillators to control the walk of a 5-link planar biped in simulation. Taga et al. showed that stable and flexible locomotion can be generated by an entrainment between the rhythmic activities of the coupled CPGs and the rhythmic movements of the mechanical structure. Sensory feedback was used for CPG modulation. The concept of entrainment, whereby the oscillations of the neural circuit and the mechanical structure reinforce each other, is the central idea presented in this study. This was responsible for the emergence of desirable gait properties such as rejection of external disturbance or adaptation to different kinds of terrain.

Aoi and Tsuchiya [3] used a simple planar biped to analyze the dynamic properties of legged locomotion using an oscillator. Feedback signals were obtained when the legs made contact with the ground and this was used to reset the oscillator phase. The authors also investigated the stability of the motion using a Poincaré map, a tool used for the stability analysis of a dynamic system.

In [35], Ishiguro et al. used a network of Matsuoka oscillators to control the movements of a 3-dimensional, 7 degrees-of-freedom biped in simulation. First, a basic CPG circuit was created by optimizing the parameters using a genetic algorithm. The authors used an artificial neuromodulation mechanism for changing the CPG parameters so that the robot can adapt to disturbances. A second stage of optimization was used to find the parameters used in the neuromodulation mechanism. For achieving entrainment, sensory feedback was introduced into the CPG network in the form of current angles of the hip joints. Using this approach the biped was able to walk on flat ground, as well as, on complex surfaces consisting of different kinds of slopes.

In [17], Endo et al. used a CPG-based controller to achieve bipedal locomotion in a physical humanoid robot. The walking motion was broken up into a stepping-in-place motion and a propulsive motion. The feedback pathways for propulsive motion were learned using a policy-gradient based method. The developed feedback controller showed stable walking behavior in a physical environment.

Cristiano et al. [11] implemented a CPG network for controlling the Nao robot's walking behavior in simulation as well as in the real world. Their CPG network was composed of Matsuoka oscillators. One of these oscillators, known as the pacemaker oscillator, was responsible for the overall behavior of the network since all other oscillators derived their output based on the pacemaker's output. A phase resetting feedback mechanism was used to reset the pacemaker when a certain condition, related to foot-ground contact, was met. The trunk angle of the robot was also used to modify a CPG parameter responsible for how much the robot leans in the sagittal plane, in order to enable walking on sloped surfaces. CPG parameters were optimized using a genetic algorithm. In [12] and [10], the same authors have investigated other kinds of CPG configurations.

Another type of artificial CPG, the Hopf oscillator, has also been used for achieving bipedal locomotion. In [79], Righetti and Ijspeert have used a network of coupled Hopf oscillators for the locomotion of the HOAP-2 humanoid robot in simulation. Since Hopf oscillators require a teaching signal, which they learn to replicate, the authors have used the joint angle trajectories generated by the default walking algorithm (provided by the robot manufacturer) as the teaching signal. Feedback pathways have been used for increasing the lateral and sagittal stability while walking. For entraining the oscillators with the body dynamics, the oscillators' phase was reset whenever the right leg touched the ground.

In [52], Kieboom has also used Hopf oscillators and the HOAP-2 robot, but here simple sine wave patterns were used as the teaching signals for the oscillators. The parameters of the sine wave generators were optimized separately using particle swarm optimization (PSO) [15]. In [31], Ijspeert has provided a comprehensive review of the use of CPGs for locomotion control.

3.3.2 Hybrid Methods

In addition to methods based purely on central pattern generators, there are also instances when CPG-based approaches have been combined with model-based approaches such as ZMP. In [74], Or has proposed a hybrid CPG-ZMP control system for bipedal locomotion of a flexible-spine humanoid robot in simulation. Here, first, a ZMP-based technique was used to generate the walking gait of the robot. The CPGs were used to generate rhythmic motion in the spine. The actual ZMP location was tracked while walking, and using the movement of the spine as a compensatory mechanism, the ZMP was kept within an acceptable area. The spinal-CPGs were controlled using a global signal.

In [102], Yang et al. have also used a combination of CPGs and ZMP to control the movements of a seven link robot in simulation. Each CPG was used to control an individual joint and the ZMP information was used as feedback for modulating the amplitude, frequency and phase of the CPGs, in order to maintain stability.

3.4 Discussion

A significant amount of work has been done by using model-based methods such as ZMP. A ZMP-based approach provides guarantees about the dynamic stability of a bipedal robot while walking - a fact that has contributed to the popularity of this method. However, a few aspects of the scope and applicability of ZMP and model-based methods in general should be put forward to understand the limitations of this approach. Model-based methods require either full knowledge of the robot's dynamics or a simplified model which can abstract the actual dynamics. Obtaining and using full-body dynamics is a complicated process. Usage of simplified dynamic models, such as LIPM, overcomes the complexity but may lead to inaccuracies in the assumed behavior of the robot. Online walking algorithms using ZMP need to perform a significant amount of computation because inverse kinematics is required

to find the joint angle trajectories from the COM or ZMP trajectory [1]. The ZMP method is applicable only for fully-actuated position controlled robots, and so torque controlled robots are not suitable for this method [86]. A big limitation of this approach is that ZMP computations do not consider the natural dynamics of the robot and so are not able to take advantage of it. This leads to excessive energy consumption for maintaining balance and may result in an unnatural gait. State-of-the-art humanoids such as ASIMO have been found to use 20 times the energy used by a typical human for walking [7] (after performing necessary scaling operations due to the difference in size and weight). Pre-computation of COM or ZMP trajectories also relies on knowledge about the terrain, which may not be always available or accurate in a real-world situation [1].

Dynamics-based methods are specially designed to exploit the natural dynamics of the robot. This approach relies on specially designed physical mechanisms and has led to the development of very agile and energy efficient bipedal robots [8, 29]. However, there is no unified general strategy for achieving the desired results [1]. Active dynamics-based approaches are also more suitable for fast moving robots and cannot be generalized for use on any kind of bipedal robot.

CPG-based methods offer a nice balance between model-based and dynamics-based methods. They can be quite easily used in most bipedal robots because they provide the option of controlling different joints separately. CPG-based methods can also take advantage of the natural body dynamics by utilizing the property of entrainment. They display stable limit-cycle behavior (where a CPG changes its oscillation pattern when disturbed but returns to the original pattern after the disturbance is removed) and can efficiently take care of unexpected disturbances which occur during walking [31]. The computational requirements of CPGs are also not significant, which makes them suitable for online and real-time applications. Additionally, there is no dependence on any kind of full-body or simplified model [1]. CPGs also provide the option of modulating their output using only a few parameters [31], and this can be utilized for realizing higher forms of control built on top of the CPG network.

However, CPG-based methods have some limitations as well. A solid theoretical foundation relating to the stability and behavior of complete CPG-based robotic systems is missing [31], although theoretical work about the behavior of individual CPGs exists [80, 66, 65, 67]. Some CPGs, such as the Matsuoka oscillators, rely on optimization algorithms for finding good parameter values and this results in a high dependence on simulation experiments.

Most of the studies dealing with CPGs use some form of feedback control [35, 11], but it is done in a static manner. For example, a particular CPG parameter is varied linearly with some feedback value (such as body incline). Introducing a trainable high-level controller, which can use feedback information to modulate CPGs, will lead to greater generalization abilities and a more effective form of control. Hence, in this thesis, instead of statically depending on feedback values, a neural network is trained, using reinforcement learning, to modulate the CPG parameters.

Chapter 4

System Architecture

Central pattern generators provide an elegant way of solving the problem of bipedal locomotion. In chapter 3, a number of studies were discussed, in which CPGs have been used to make robots walk. The common approach in most of these studies is to use a network of coupled CPGs to control the joints for walking. The CPG network is generally augmented by feedback pathways to allow entrainment between the CPG dynamics and the body dynamics of the robot. The output of the CPG network, and therefore the movement of the individual joints, can be controlled by modulating only a few CPG parameters. This is also similar to the control mechanism found in animals (figure 2.1a). This kind of a setup opens up the possibility of introducing a trainable high-level controller, which can learn how to modulate the CPG network to achieve some global objective. By utilizing existing knowledge about CPGs and recent advances in reinforcement learning, a hierarchical walking controller for bipedal walking is proposed in this thesis.

In this chapter, the architectural details of the proposed system are discussed. Starting with an overview of the system, each component is described, including the structure of the low-level CPG controller, the feedback pathways, and the high-level neural network controller. Details of parameter optimization, training and experiments are presented later, in chapters 5 and 6.

4.1 Overview

Using the vertebrate locomotor system discussed in section 2.1 as an inspiration, the task of controlling joint motions for walking has been broken up into two parts. At the bottom of this hierarchy lies the CPG network which forms the low-level controller. It is responsible for sending target angles to the robot's joints. The high-level controller is made up of a feed-forward neural network, and is used to help the robot maintain a straight walking trajectory. The controller monitors the state of the robot's torso and modulates a couple of parameters in the CPG network, which in turn, effects the CPG network's output pattern. It is important to note that the high-level controller does not need to control the robot's joints directly. As a result, the dimension of the high-level control signal is much lower than the

dimension of the low-level control signal (which consists of the target joint angles) which the CPG network sends to the robot. The robot accepts the target angles and attempts to reach the angles it has been given. A simplified overview of the system architecture is shown in figure 4.1.

A simulated Neuro-Inspired COmpanion (NICO) robot has been used in this work. The NICO robot is 101cm tall, weighs 7kg and has 30 DoF (degrees-of-freedom) in total. Each leg has 3 DoF in the hip, 1 DoF in the knee and 2 DoF in the ankle [51]. Force sensors were added to the feet of the simulated robot for detecting foot-ground contact. The controller proposed in this work controls 10 out of the 12 joints in the two legs and 2 joints in the shoulders (since arm swing increases stability while walking). Only joints moving in the sagittal[1] or frontal[2] planes are used. Although the NICO has been used in this work, any position-controlled bipedal robot with a similar morphology can be used in its stead.

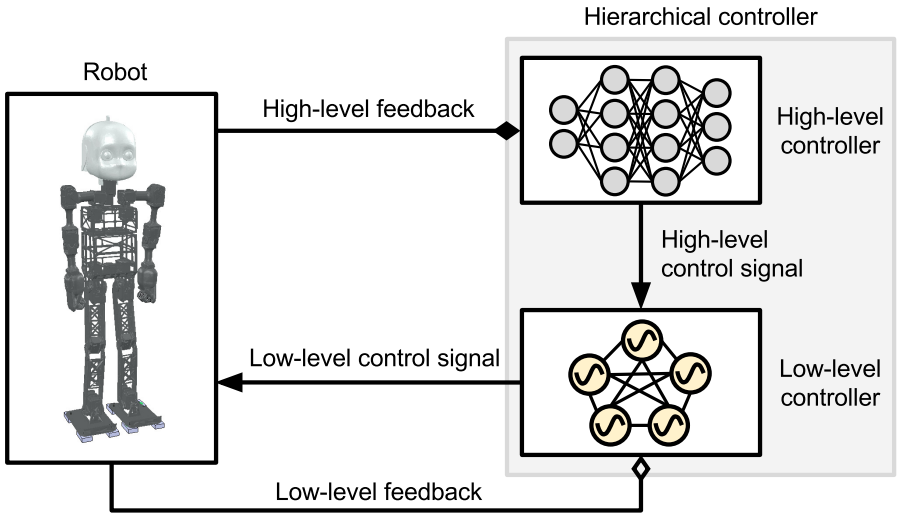


Figure 4.1: Overview of the system, showing the layout of the different components.

4.2 Structure of the Low-level Controller

The low-level controller is made up of a network of coupled Matsuoka oscillators (discussed in section 2.2). The Matsuoka oscillators autonomously produce rhythmic signals which are directly used to drive the individual joints responsible for walking. The CPG network is modulated by using low-level feedback loops which help to entrain the CPG network's output with the natural dynamics of the robot. The CPG network configuration, that is used in this work, was proposed by Cristiano et al. [11], who used it for controlling the Nao robot. Since the NICO robot has a similar structure, the network could easily be used for it. However, changes have been made to the way feedback is handled. Additional parameters have been introduced to accommodate the novel high-level control mechanism.

[1]The sagittal or median plane divides the body into left and right.
[2]The frontal or coronal plane divides the body into front and back.

4.2.1 CPG Network Configuration

The CPG network is created by interconnecting 13 Matsuoka oscillators, as shown in figure 4.2. Each oscillator behaves according to equations 2.3-2.7. The arrows between oscillators denote coupling connections. The direction of an arrow is from oscillator j (master) to oscillator i (slave). Dashed arrows represent inhibitory connections (weight $w_{ij} = -1$). Solid arrows represent excitatory connections (weight $w_{ij} = +1$). For inhibitory connections, the slave oscillates in opposite phase to the master and for excitatory connections, both the master and slave oscillate in phase with each other. The central oscillator in this configuration only has outgoing connections and hence, is known as the pacemaker oscillator. In the absence of any feedback connections into the pacemaker, it oscillates according to its internal parameters [11]. The pacemaker is responsible for generating the master signal that is propagated across the network.

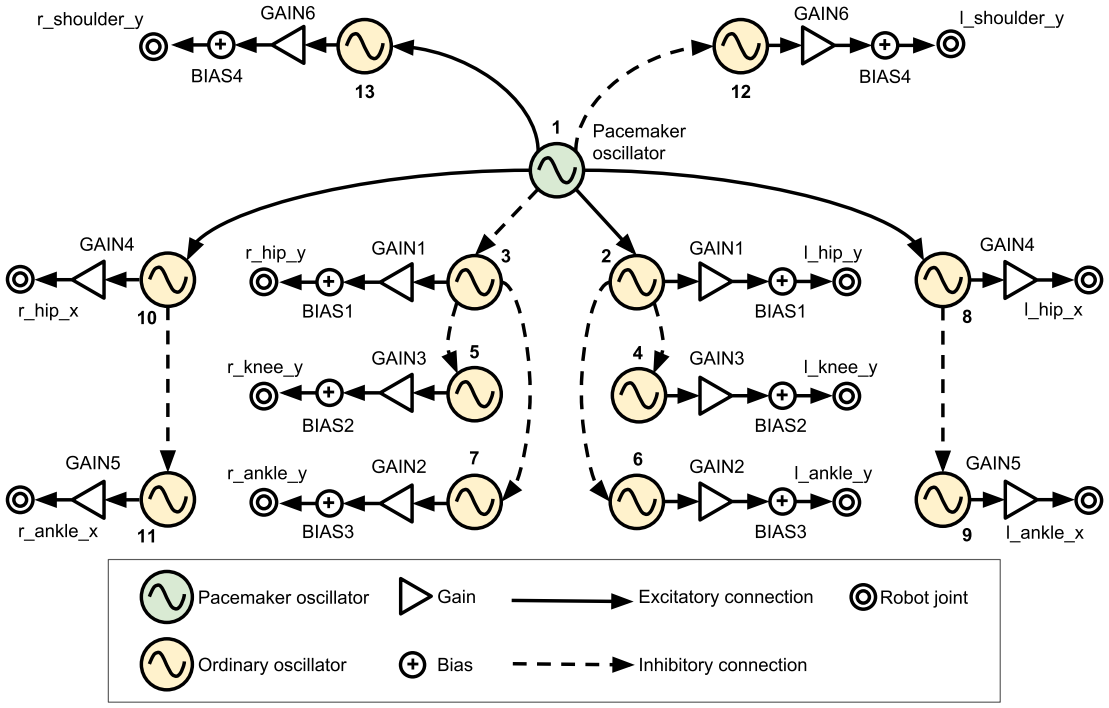


Figure 4.2: CPG Network [11]. Names of joints are indicated near the symbols for joints.

The topology of the connections in the CPG network is designed to approximately reproduce the relative phase differences between different joints in humans during walking. In humans, the left and right hips move in anti-phase in the sagittal plane. Similarly, the right and left shoulders also move in anti-phase in the sagittal plane. The arms and legs on the opposite (contralateral) side move in-phase. This relationship is enforced through the connections of the pacemaker to the hip and shoulder oscillators for the sagittal joints (oscillator numbers 2, 3, 12 and 13) [11]. The frontal hip joints should move in-phase and so both the corresponding oscillators have excitatory connections with the pacemaker. Ipsilateral

frontal joints should move in opposite phase so that the foot stays parallel to the ground. This is enforced by having inhibitory connections between the ipsilateral frontal oscillators, that is, between oscillators 8 and 9, and between oscillators 10 and 11. Connections between the oscillators for the knee and ankle joints in the sagittal plane follow a similar logic.

Apart from the pacemaker, all other oscillators are used to generate target angles for the robot's joints. There are 12 such oscillators corresponding to 12 joints. The output o_i of oscillator i is converted into the target angle θ_i for the i^{th} joint using equation 4.1.

$$\theta_i = o_i \times gain_i + bias_i \tag{4.1}$$

Here $gain_i$ is a factor which is used to alter the amplitude of the oscillator's output and $bias_i$ is an additive term that shifts the mean position of oscillation. Using these two factors for each oscillator, target angles of arbitrary nature can be produced. However, the frequency of all the oscillators will be the same as that of the pacemaker. This frequency depends on the pacemaker time constants τ and τ', and the constant k_f (equation 2.7). Thus if the pacemaker's parameters can be chosen so that the conditions for stable oscillation are satisfied [66], then it is possible to generate target joint angle trajectories of arbitrary frequency, amplitude, and mean-position by varying the values of k_f and the different gains and biases within some range. The different trajectories will have a phase difference of either 0 or π radians with the output of the pacemaker oscillator. The low-level control signal emitted by the CPG network (as shown in figure 4.1) at each time step is the vector $[\theta_2, \theta_3, \theta_4, ..., \theta_{13}]$ consisting of the target joint angles.

Osc. i	Joint (θ_i)	$gain_i$	$bias_i$
1	-	-	-
2	l_hip_y (θ_2)	GAIN1	BIAS1
3	r_hip_y (θ_3)	GAIN1	BIAS1
4	l_knee_y (θ_4)	GAIN3	BIAS2
5	r_knee_y (θ_5)	GAIN3	BIAS2
6	l_ankle_y (θ_6)	GAIN2	BIAS3
7	r_ankle_y (θ_7)	GAIN2	BIAS3
8	l_hip_x (θ_8)	GAIN4	0
9	l_ankle_x (θ_9)	GAIN5	0
10	r_hip_x (θ_{10})	GAIN4	0
11	r_ankle_x (θ_{11})	GAIN5	0
12	l_shoulder_y (θ_{12})	GAIN6	BIAS4
13	r_shoulder_y (θ_{13})	GAIN6	BIAS4

Table 4.1: Oscillators and their joints, gains and biases. Values of gains and biases are symmetrical across the left and right sides of the robot. Frontal joint names end in _x, sagittal joint names end in _y.

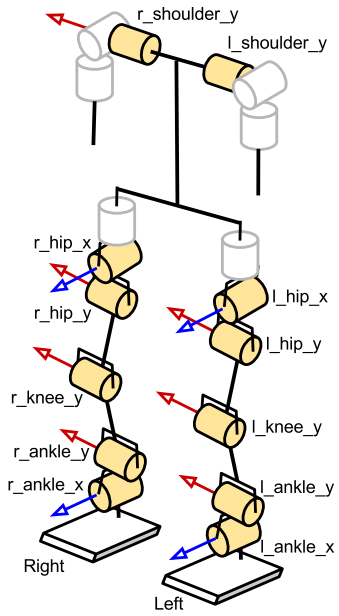


Figure 4.3: Joints driven by oscillators (shown in color). Sagittal joints' axes are shown in red, frontal joints' axes are shown in blue.

For the joints in the sagittal plane, both gains and biases are used to allow for arbitrary angle trajectories. However, for the frontal joints, only gains are used. Biases are omitted in this case because the frontal joints must oscillate about the zero position. Since they control the left-to-right swaying motion, any other mean position will cause the robot to sway more in one direction. In order to maintain symmetry between the movement of the left and right joints, the gains and biases on one side are equal to the gains and biases of the corresponding oscillator on the other side of the body. This results in four different bias parameters and six different gain parameters. The oscillators and their respective joints, gains, and biases are listed in table 4.1. The location of the different joints controlled by the CPG network, and their respective axes of rotation are shown in figure 4.3.

The pacemaker's internal parameters (table 4.3) are determined empirically so that stable oscillations can be produced. Then these internal parameter values are replicated in the other oscillators. This avoids the need of determining the values of the large number of internal oscillator parameters through optimization. Thus, for the entire CPG network, the parameters which must be optimized are: k_f for controlling the frequency, and the four biases and six gains for controlling the mean positions and amplitudes of the produced signals. These are shown in table 4.2.

Table 4.2: CPG network parameters to be optimized.

k_f	GAIN1	GAIN2	GAIN3	GAIN4	GAIN5	GAIN6	BIAS1	BIAS2	BIAS3	BIAS4

4.2.2 Pacemaker Oscillator

The design of the CPG network in figure 4.2 is such that once suitable internal parameters for the pacemaker oscillator are found, rhythmically varying target joint angle trajectories can be generated by the network autonomously. Determining suitable parameters by hand requires some experimentation but it greatly reduces the search space of the optimization algorithm, which can instead search for proper values for k_f and the gains and biases that are used in the CPG network. Table 4.3 lists the constant parameters for the pacemaker which have been used by Cristiano et al. in [11]. The same values have been used for this work because they

Table 4.3: Internal parameters of the pacemaker oscillator [11].

Parameter	Description	Value
τ_0	Time constant for rate of discharge	0.2800
τ_0'	Time constant for adaptation	0.4977
β	Self-inhibition constant	2.5000
w_0	Constant of mutual inhibition	2.2829
u_e	Tonic input	0.4111
m_1, m_2	Output scaling constants	1.0

result in stable oscillations and support a large range of the variable parameter k_f. Using these values, the outputs of the pacemaker oscillator for three values of the frequency controlling parameter, k_f, are shown in figure 4.4. The oscillation frequency decreases as k_f is increased from 0.2 to 0.3 and then to 0.4. Oscillations are still produced when $k_f \approx 0.01$ (very high frequency) and also when $k_f \approx 3.0$ (very low frequency). But values of k_f outside the range $[0.2, 1.0]$ produce frequencies which are not usable on a walking robot. It is to be noted that in the present scenario, the output signal of the oscillator is taken to be a joint angle trajectory. The original definition of the Matsuoka oscillator defines the output as a torque and this interpretation has been used in studies like [96]. However, since there are no constraints regarding this, the rhythmic output of the oscillator can be interpreted as the scenario demands.

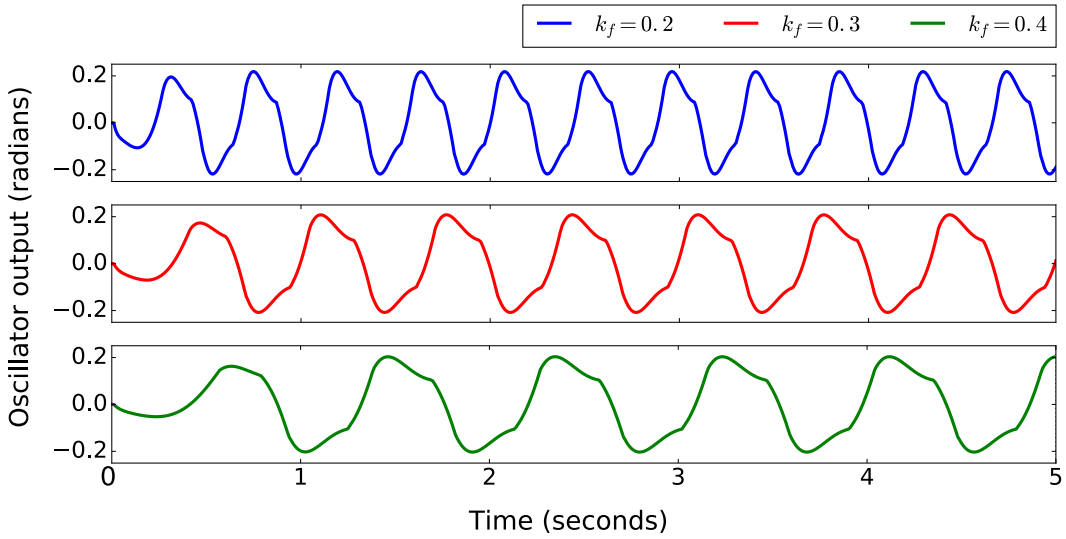


Figure 4.4: Pacemaker output for three different values of k_f.

4.2.3 Feedback Pathways

In the absence of feedback pathways, the CPG network behaves like an isolated system whose oscillations are only dependent on the internal parameters of the CPGs. In such an open loop configuration, there is no way for the CPG network to alter its behavior in order to adapt to situations the robot may be facing. The introduction of feedback changes this, by allowing the CPG network to change its output based on proprioceptive information provided by the robot. Feedback pathways provide a way for the CPG network to entrain itself with the body dynamics of the robot and hence, is of utmost importance.

Whenever CPGs are used for joint control, some form of feedback is employed. However, an investigation into the manner in which this feedback improves the robot's gait is lacking in the current literature. A comparison between an open loop CPG network and networks with two different kinds of low-level feedback is

taken up in this thesis. This will help to identify the relative merits and demerits of the feedback types. The two kinds of feedback investigated here are called *angle feedback* and *phase reset*.

Angle Feedback

The output of a CPG is used to drive a joint, but this angle cannot be reached instantaneously. Every joint will have some lag between the target and the actual angle. This provides a way of introducing feedback into the CPG network. Differential equations 2.3 and 2.4 of the extensor and flexor neurons of a Matsuoka oscillator contain the feedback variables f_{1i} (extensor) and f_{2i} (flexor). The actual angle values, after multiplying with a constant, can be fed into these variables. The feedback being provided to the flexor is the negative of the feedback being fed into the extensor as the two neurons are antagonistic.

$$\begin{aligned} f_{1i} &= k \times \theta_i^{actual} \qquad \text{(extensor)} \\ f_{2i} &= -k \times \theta_i^{actual} \qquad \text{(flexor)} \end{aligned} \tag{4.2}$$

Here k is the weight of the feedback and θ_i^{actual} is the current angle of the i^{th} joint.

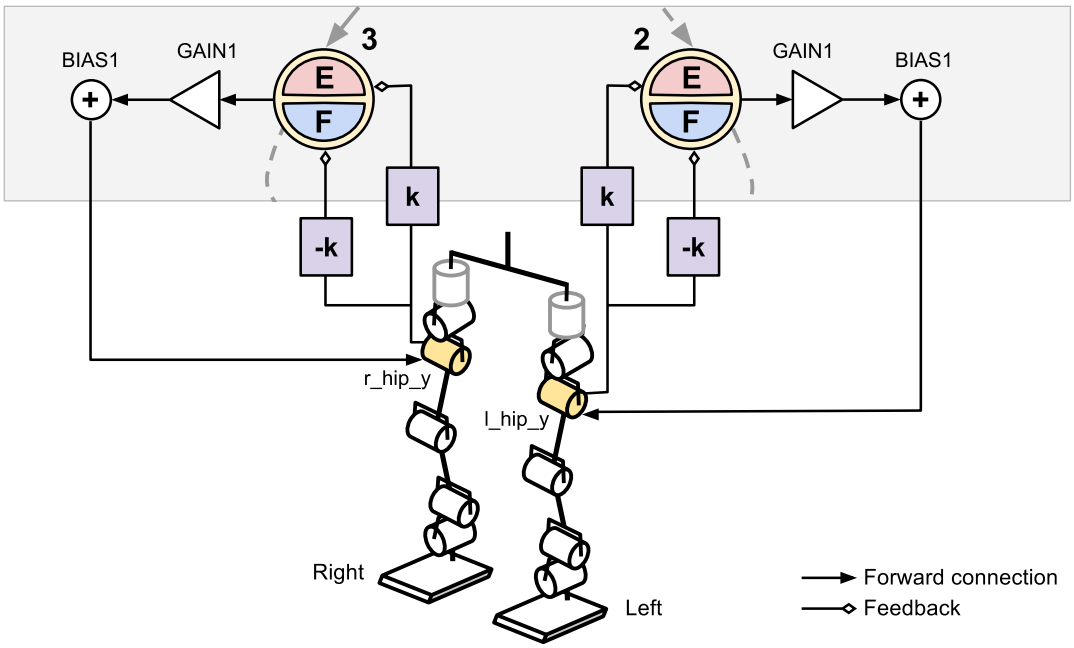


Figure 4.5: Angle feedback configuration. The grey box shows a portion of the CPG network containing oscillators 2 and 3. E and F denote the extensor and flexor neurons respectively.

In the simplest case, only one set of joints is used for providing feedback. Oscillators 2 and 3 act as masters to the other oscillators controlling sagittal joints, in their respective legs. Since forward motion is caused by the movement in the sagittal plane only, the role of oscillators 2 and 3 is the most important. Hence, feedback was only introduced into these two oscillators.

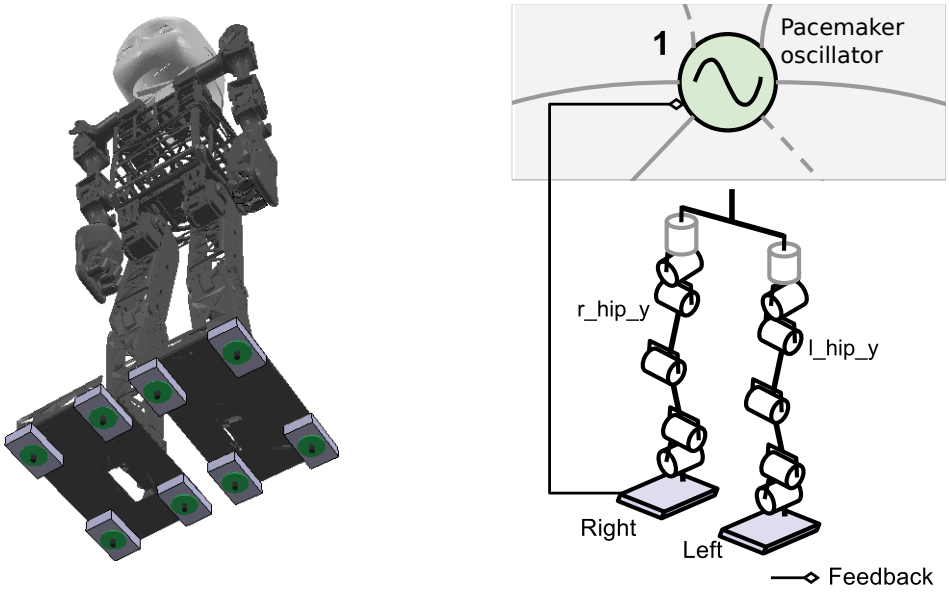
The feedback connections are depicted in figure 4.5. This feedback mechanism is inspired by the one used by Ishiguro et al. [35]. However, in this thesis, a different CPG network configuration is used to control a full humanoid robot in simulation, whereas Ishiguro et al. used a simplified 3D model. Additionally, here the feedback weight parameter k is optimized along with the frequency parameter k_f, and the gains and biases of the CPG network, as shown in table 4.4.

Table 4.4: CPG network parameters to be optimized for angle feedback.

k_f	GAIN1	GAIN2	GAIN3	GAIN4	GAIN5	GAIN6	BIAS1	BIAS2	BIAS3	BIAS4	k

Phase Reset

When the CPG network is used to make the robot walk, there are two kinds of dynamics at play - the output of the CPG network and the movement of the robot's legs. Due to a delay in reaching target angles, the rhythms of the legs and the CPG network may not be synchronized. Phase resetting is a way to achieve this synchronization. The idea is to keep track of the time instants when the feet strike the ground and to reset the phase of the pacemaker (and through it, the other oscillators) when needed so that the CPG cycle is coordinated with the leg movements.



(a) Force sensors (b) Connection for phase reset

Figure 4.6: (a) Force sensors at the heel are used to detect ground contact. (b) The grey box shows a portion of the CPG network with the pacemaker. The feedback signal from the right leg resets the pacemaker's state variables to 0.

For implementing phase resetting, Cristiano et al. [11] kept track of the time intervals between successive ground contacts by the feet and when this time interval

was outside a permissible range, the pacemaker was reset. However, determining what the permissible range should be, is not a straightforward task. It requires careful tracking of the time differences between successive peaks in the pacemaker's output signal and comparing this to the time differences between successive foot-ground contacts. Then a tolerance range needs to be set and when the difference between the two time intervals goes beyond this range, the phase reset mechanism needs to be triggered. A simpler way, used by Righetti et al. [79], is to start the walking cycle with the left leg and to reset the pacemaker whenever the right foot hits the ground. As this method does not introduce any additional parameters, it has been used in this work. Thus, the parameter set of the original CPG network, shown in table 4.2, remains unchanged.

For detecting foot-ground contact, force sensors were added to the four corners of each foot of the simulated robot (figure 4.6a). When the average force of the two sensors in the right leg's heel went above a threshold, the pacemaker oscillator was reset by setting its state variables $(u_{11}, v_{11}, u_{21}, v_{21})$ to 0. The threshold for detecting ground contact was set to 10N based on experimental observations. The walking cycle begins with the left foot moving forward (the joint *l_hip_y* rotates in the negative direction) and the right foot hitting the ground indicates that the left foot should start moving forward again. Phase resetting is especially useful in scenarios where the foot strikes the ground unexpectedly in the middle of a cycle. At that moment the other foot immediately starts moving forward again, similar to the way humans recover after a sudden trip while walking.

The effect of resetting the pacemaker oscillator is shown in figure 4.7. The pacemaker oscillator's parameters were set according to table 4.3 and $k_f = 0.2$ was used. The phase of the oscillator was reset at 1.5, 2.5 and 3.7 seconds. As shown in the plot, at the end, the signal with phase resetting (solid line) can vary a lot from the one without any phase resetting (dashed line).

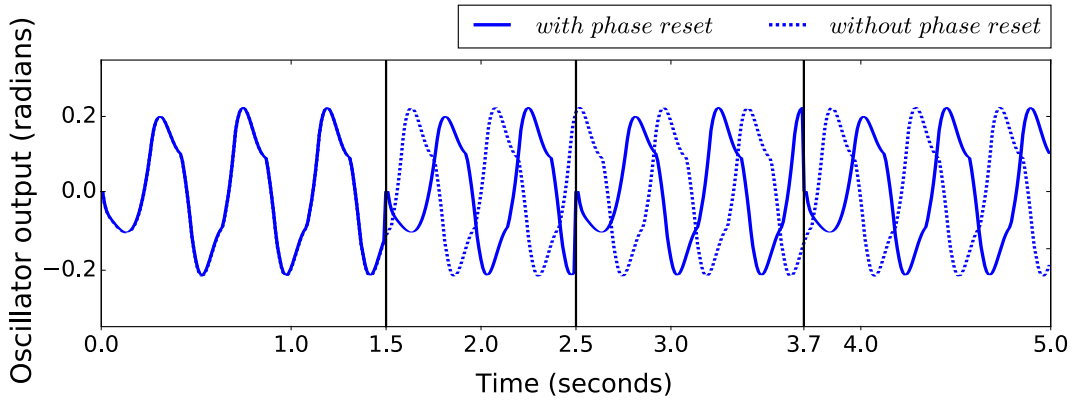


Figure 4.7: Effect of phase resetting on the pacemaker's output. $k_f = 0.2$ was used to generate the output. Vertical lines indicate the time instants when phase reset was triggered.

4.3 Structure of the High-level Controller

A robot, controlled by a CPG network alone, may be capable of robust bipedal locomotion, but even with feedback mechanisms, there is a limit up to which such a low-level controller can react to environmental changes. A feedback-enabled CPG network only uses proprioceptive information such as foot-ground contact or actual joint angles. If additional sensory information can be utilized, the controller may become even more robust and adaptable. For example, information from the camera can be used to determine how to change direction, or information from the inertial sensors can be used to maintain balance on a sloped surface. A modular, high-level controller can accept such sensory information and then make appropriate changes in the CPG network parameters to achieve the desired behavior.

A high-level controller also provides a way of dealing with structural errors in the model and with environmental disturbances. Due to possible minor inconsistencies in the simulation model, and slippage between the feet and the ground, the NICO tends to deviate from a straight line trajectory when it walks for an extended duration of time. Building a perfect model would eliminate the deviation in simulation, but in the real world, the robot would still be subject to errors. Thus, it is crucial for the controller to function even in the presence of systematic and non-systematic errors. Hence, to show the usefulness of a high-level controller, a neural network is used in this thesis to make the robot walk in a straight line. For this, the neural network monitors the state of the robot's torso and then modulates some parameters in the CPG network so that any lateral deviation is minimized.

4.3.1 Control Mechanism

One way to control deviation is to interrupt the robot's walk, take corrective steps by turning in a particular direction to minimize the deviation, and then to resume walking again. This strategy unnecessarily breaks up the gait into alternate phases of walking and turning, and is not the ideal way to attain the desired objective. A better strategy is to make continuous small adjustments while the forward walk is in progress. This has two advantages: the walk is uninterrupted and smooth, and drastic corrections do not have to be made. Moreover, deviation control can be achieved by using the same joints that are used for the forward walk.

The joints *l_hip_y* and *r_hip_y* control the hip joints responsible for leg movements in the sagittal plane, which cause the forward motion of the robot. These joints are driven by oscillators 2 and 3 respectively. The walking motion starts when the robot lifts the left leg and moves it forward. To be able to lift the left leg, the robot must transfer its weight onto the right leg. This is done by swaying the hips to the right concurrently with the forward motion of the left leg. Similarly, when the right leg moves forward, the weight is transferred onto the left leg by swaying towards the left. The swaying motion of the hips is controlled by the joints *l_hip_x*, *r_hip_x*, *l_ankle_x* and *r_ankle_x* which are driven by oscillators 8, 10, 9 and 11 respectively (see table 4.1 for a mapping between oscillators and joints, and figure 4.3 for the location of the different joints in the robot).

The oscillator output o_i is related to the target joint angle θ_i according to equation 4.1. Thus the target angle for l_hip_y is obtained by multiplying the output of oscillator 2 with the gain applied to this oscillator and then adding a bias to the result ($\theta_2 = l\_hip\_y = o_2 \times gain_2 + bias_2$). Similarly the target angle for r_hip_y is obtained by using the equation $\theta_3 = r\_hip\_y = o_3 \times gain_3 + bias_3$. To maintain symmetry between the movements of the left and right joints, the same gain and bias values are applied to oscillators 2 and 3 ($gain_2 = gain_3$ and $bias_2 = bias_3$). Under ideal conditions, this should have been sufficient for walking in a straight line, but since this is not the case, a deviation can occur.

To counter the deviation, $gain_i$ in equation 4.1 is multiplied by a factor Ψ. This is done only for oscillators 2 and 3, which drive the forward motion. By applying different values of Ψ to the oscillators 2 and 3, the target angles and hence the forward motion for the two legs can be made different. The value of $gain_i$ is not changed directly, since its value is obtained by optimizing the CPG network parameters, and once found, it should be frozen. The factors Ψ_l and Ψ_r, for oscillators 2 and 3, are derived directly from the outputs of the high-level neural network controller. Thus, the equations for the target angles for forward motion are modified as:

$$\begin{aligned} \theta_2 &= l\_hip\_y = (o_2 \times \Psi_l \times gain_2) + bias_2 \\ \theta_3 &= r\_hip\_y = (o_3 \times \Psi_r \times gain_3) + bias_3 \end{aligned} \tag{4.3}$$

The forward stride of each leg is a combination of two movements. For the left leg, the left sagittal hip joint causes the forward motion. During this forward motion, the left frontal hip joint causes a sideways motion to the right. The same logic is applicable for the right leg as well, where the sagittal joint causes the forward motion and the frontal joint causes a simultaneous movement to the left. As a result, each individual step follows a curved trajectory, as shown in figure 4.8.

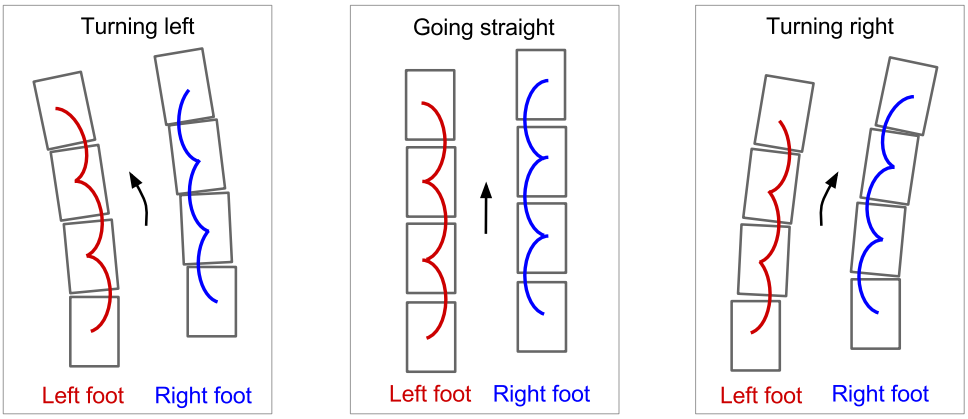


Figure 4.8: Turning mechanism. The curves show the trajectory of the individual steps. **Center**: the forward and sideways motion of both legs are unchanged and the robot walks straight. **Left**: the forward motion of the right foot is suppressed, but the sideways motion remains the same. So, the robot turns to the left. **Right**: The forward motion of the left leg is suppressed, but the sideways motion remains the same. Hence, the robot turns to the right. Reason for the turn is explained in the text.

To understand how Ψ_l and Ψ_r cause the robot to turn while walking, consider the following scenario. Suppose the robot has deviated to the left of its path and needs to turn right to compensate for the deviation. In this situation, if Ψ_l is assigned a lower value than Ψ_r, then the amplitude of motion of the sagittal left hip joint (target angle θ_2) will be less than the amplitude of motion of the sagittal right hip joint (target angle θ_3). The movements of both the frontal hip joints remain unchanged.

When the forward movement is reduced and the sideways movement is kept the same, the left leg ends up in a position which is slightly to the right of the position it would reach if the forward motion was not suppressed (see figure 4.8 (right)). This results in the robot taking a small step sideways to the right. When the left leg is planted in the new position and the right leg moves forward, the next position of planting the right foot also moves slightly to the right. Over a few successive steps, this causes the robot to make a counter-deviation to the right. Once the robot regains the correct trajectory and heading direction, the values of Ψ_l and Ψ_r are again made equal. The decision of when and how much to change the values of Ψ_l and Ψ_r are taken by the high-level neural network controller, based on the state of the robot's torso.

However, there is no labeled dataset containing the correct outputs for every possible observation of the torso's state, which can be used to train the neural network. Hence, the network is trained using reinforcement learning, which relies on a trial-and-error approach involving repeated interactions with the environment.

4.3.2 High-level Neural Network Controller

The neural network, shown in figure 4.9, forms the high-level controller and is trained using the Deep Deterministic Policy Gradient algorithm (DDPG), which was discussed earlier in section 2.4.3. DDPG treats the neural network as a function approximator for the policy function. After successful training, this network may be directly queried using the state of the robot's torso to find out what the appropriate action should be. DDPG, being an actor-critic algorithm, uses another neural network (critic) to critique the actions determined by the policy network (actor) during training. Details of the critic network, training process and other aspects related to reinforcement learning are discussed later in chapter 6. In this section, the focus is on the structure of the policy network and how this high-level neural network controller fits into the overall system.

The high-level controller is a fully-connected, feed-forward neural network consisting of 2 hidden layers containing 400 and 300 units respectively. Units of all the hidden layers use the rectified non-linearity activation function (ReLu). The output layer consists of two neurons with sigmoid activations to bound the output in the range $[0.0, 1.0]$. The outputs of the network are labeled Φ_l and Φ_r and they are used to derive the parameters Ψ_l and Ψ_r respectively, discussed in equation 4.3. The input to the network is a 12-element vector $[\alpha, \beta, \gamma, \dot{\alpha}, \dot{\beta}, \dot{\gamma}, x, y, z, \dot{x}, \dot{y}, \dot{z}]$, consisting of the angular position (α, β, γ) and velocity $(\dot{\alpha}, \dot{\beta}, \dot{\gamma})$, and the cartesian position (x, y, z) and velocity $(\dot{x}, \dot{y}, \dot{z})$ of the robot's torso in the three dimensions.

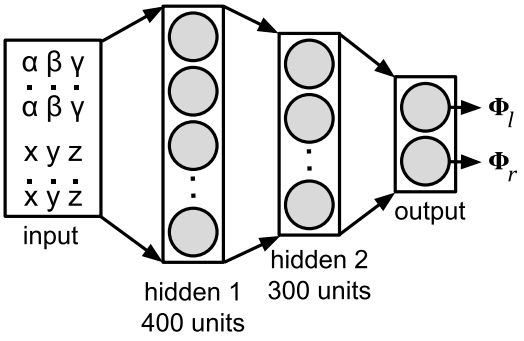


Figure 4.9: High-level neural network controller.

The outputs Φ_l and Φ_r make up the high-level control signal that is sent to the CPG network to modulate its output. However, these values are not used directly as factors for the gains in equation 4.3. The initialization of the final layer weights and biases, by the DDPG algorithm, is such that the policy outputs are near zero in the initial stages of training. In such a scenario, if Φ_l and Φ_r (both having near-zero values) are used as gain factors, the angles *l_hip_y* and *r_hip_y*, computed using the relation $(oscillator\_output \times gain\_factor \times gain) + bias$, would not show any oscillatory behavior about the bias position (equation 4.3). Hence the robot would not be exhibiting any forward motion at all. To avoid this problem, Φ_l and Φ_r are instead, interpreted as *gain suppression factors*, which determine how much the value of a gain is to be reduced. Since the values of Φ_l and Φ_r are restricted to be within [0.0,1.0], a value of 0.0 will denote no reduction in the gain and 1.0 will indicate that the gain is to be maximally reduced. The reciprocal relationship between the *gain factor* Ψ and the *gain suppression factor* Φ is represented in equation 4.4.

$$\begin{aligned} \Psi_l &= 1.0 - (1.0 - \xi) \times \Phi_l \\ \Psi_r &= 1.0 - (1.0 - \xi) \times \Phi_r \end{aligned} \tag{4.4}$$

The parameter ξ, called the *lowest gain factor*, is the lowest value which may be assigned to the *gain factors* Ψ_l and Ψ_r. For instance, when the *gain suppression factor* Φ_l has a value of 1.0, it implies maximal suppression of the gain. The *gain factor* Ψ_l, in this case, is assigned the value ξ. When Φ_l has a value of 0.0, it means that the gain should not be changed and hence the corresponding gain factor, Ψ_l, should have a value of 1.0. The same logic is applicable for Φ_r and Ψ_r as well. The value for ξ is also bound to be in the range [0.0,1.0].

Another interpretation of ξ is that it controls how much influence the high-level controller can exert over the low-level controller. When the value of ξ is close to 1.0, Ψ_l and Ψ_r evaluate to approximately 1.0, irrespective of the neural network's output (Φ_l and Φ_r). In this case, the high-level controller has minimal influence over the robot's behavior. When the value of ξ is near 0.0, Ψ_l and Ψ_r will be heavily dependent on Φ_l and Φ_r respectively, and hence, the high-level controller can influence the CPG network to a great extent.

Figure 4.10 depicts a detailed view of the system architecture. The different levels of the hierarchical controller are shown, along with their interconnections and internal components. The low-level controller, modulated by the low-level feedback, drives 12 joints for walking. While the robot walks, the state of its torso is fed into the high-level controller, which modulates the CPG network's output by modifying the gains of the sagittal hip joints and thereby helps the robot to walk straight.

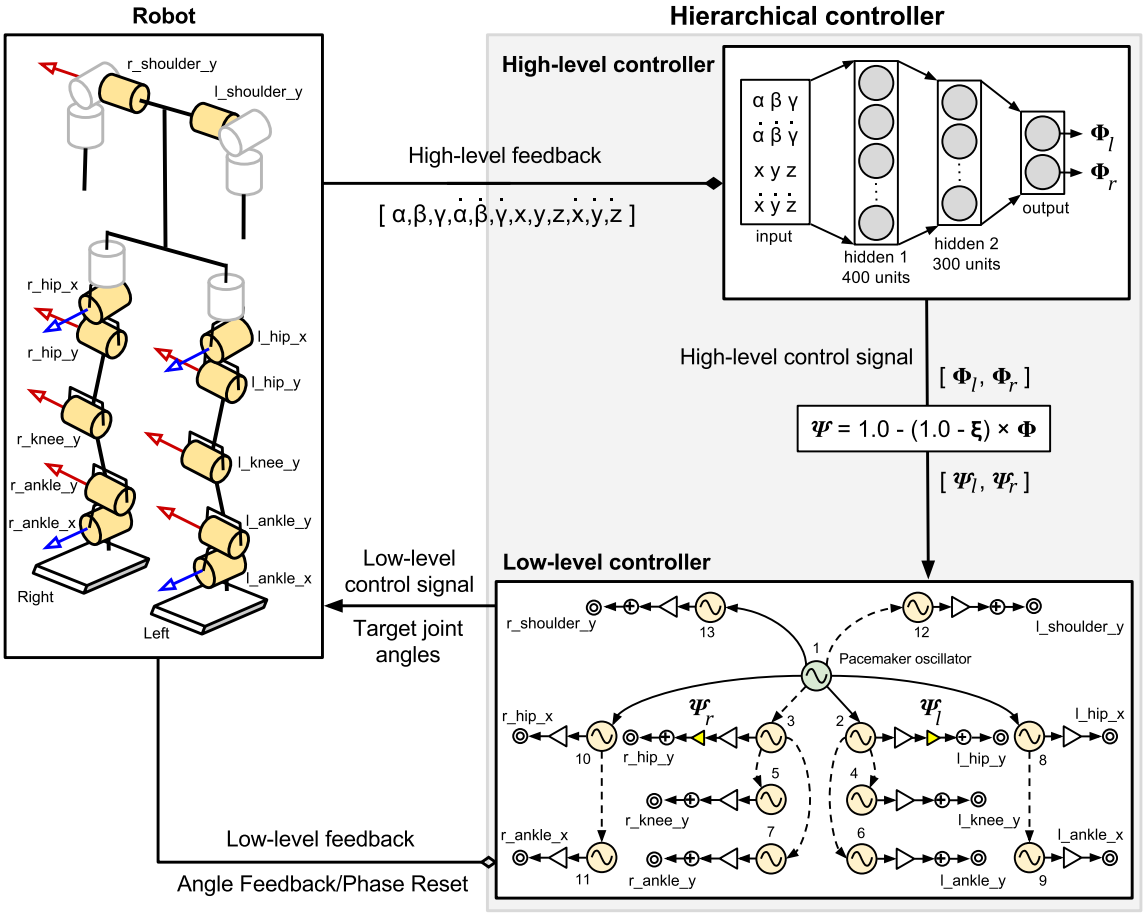


Figure 4.10: A detailed view of the system. Symbols are explained in the text.

Chapter 5

Low-level Control

Once the structure of the low-level CPG network has been decided, the next step is to find suitable values of the variable parameters that are used in the network. This leads to the emergence of a basic gait. When no feedback mechanisms are used, the basic gait is the result of the CPG network producing oscillations according to its internal parameters. To convert the CPG network from an open loop system to a reactive system, capable of adapting its behavior to the environment, low-level feedback mechanisms need to be added. The two kinds of feedback that are used here are called *angle feedback* and *phase reset* (these have been discussed in chapter 4). At this point, the system is still without any form of high-level control but is capable of exhibiting a stable walk.

This chapter deals with the optimization process that is used to find the values of the parameters used in the low-level controller. A genetic algorithm is employed for this task, using which, the parameters of three kinds of setups are evolved - a CPG network with no feedback (*open loop*), with *angle feedback* and with *phase reset* mechanisms. The results of the evolution process are discussed. The three types of gait that are obtained using these setups are then analyzed and compared to show the effects of introducing feedback into an open-loop configuration. The comparison is made on the basis of measurable quantities. This comparison also helps in choosing the feedback mechanism which performs the best. The optimized CPG network, with the chosen feedback mechanism, acts as the base on which the high-level control structure is added (discussed in chapter 6).

Although the configuration of the low-level CPG network used in this work is the same as the CPG structure proposed by Cristiano et al. [11], there are a few differences. Apart from minor changes in the implementation of the genetic algorithm, the fitness function used here is new. The fitness function used in [11] was based on the velocity and deviation of the walk, whereas, here it is based on the period of time the robot is upright and the forward distance covered during the walk. Cristiano et al. used only one kind of feedback - *phase resetting*. A simpler form of this feedback, based on [79], is used here. Additionally, *angle feedback*, which is based on [35], is also used. The objective comparison of the gaits, obtained by using the system in open loop, and with the two kinds of feedback, is a novel contribution.

5.1 Evolution of a Basic Gait

Evolutionary optimization techniques, such as genetic algorithms, are a popular choice for high-dimensional optimization problems where the fitness surface is unknown, irregular or not continuous. A genetic algorithm does not need any explicit knowledge about the problem being solved. Instead, it relies on a representation of possible solutions in the form of *chromosomes*, and on the *fitness function*, which is a way of evaluating the 'goodness' of a chromosome. General concepts related to genetic algorithms have been discussed in chapter 2. In this work, a genetic algorithm is used for optimizing the CPG network, which controls the robot in a simulated environment. Before going into the details of the optimization process, a description of the setup and simulation environment is presented.

5.1.1 Setup

The behavior of the robot in simulation depends to a certain extent on the frame rate of the simulator, which in turn, is dependent on the rendering capability of the computer. The behavior of the simulated robot on a faster machine, with a better frame-rate, is expected to be closer to what can be expected in the real world. The genetic algorithm and the gait analysis experiments were carried out on two machines: a faster machine ($M1$), having an Intel Core i5-6500 processor (3.20GHz×4) with 16GB RAM, and a machine with a lower configuration ($M2$), having an Intel Core i3-4005U processor (1.70GHz×4) with 8GB RAM. On both machines, the genetic algorithm was able to find good solutions. The robot was able to display a stable walk in the majority of cases. However, the solutions and the evolved gaits were different on the two machines due to the different configurations.

One way to ensure that the same solutions are produced across different machines would be to make the control loop of the CPG controller synchronous with that of the simulator. For every iteration of the CPG control loop, a signal can be sent to the simulator control loop to advance the simulation by one step. This would make sure that the timing of the simulator is exactly the same as that of the CPG controller, and thus the final behavior of the robot will be the same across different machine configurations.

However, doing this presents a big problem when the CPG controller is to be transferred to the real robot. In the real robot, there is no way of controlling the physical environment by sending synchronizing signals. Hence, this approach is avoided and an asynchronous mode of control is adopted. Although this leads to different solutions on different machines, the setup needs minimal modification before it can be transferred to the real robot. Although the solutions on $M1$ and $M2$ are quantitatively different, they share some common trends. Since $M1$ has a better configuration, the behavior of the simulated robot running on it is expected to be more realistic. Hence only the results from $M1$ are discussed in this chapter. Complete results of the genetic algorithm related experiments, which were performed on $M2$, are included in appendix A.

The Virtual Robot Experimentation Platform (V-REP) simulator [14] from Coppelia Robotics was used in this work. The Bullet 2.78 physics engine was used for the simulation dynamics. Apart from the simulation frame rate (determined by the machine configuration), all other aspects of the setup were identical for all the experiments on $M1$ and $M2$.

5.1.2 Genetic Algorithm Parameters

A modified form of the simple genetic algorithm (SGA) described in [22] has been used in this work. Instead of a binary chromosome, a real-valued chromosome is used, where the genes contain real numbers representing the values of the CPG network parameters. A two-point crossover mechanism is used for recombination. The points of crossing over are chosen randomly. The probability that crossover occurs is 0.8. For parent and survivor selection, a tournament selection strategy is employed, with a tournament size of 3. Chromosomes are mutated by adding a small real number (may also be a negative number) to the value of each gene. The probability that a chromosome will be chosen for mutation is 0.1. Once a chromosome is chosen, each gene in it has a probability of 0.05 of being mutated. The small real number for mutating a gene is randomly chosen from a Gaussian distribution with mean $\mu = 0.0$ and standard deviation $\sigma = 0.01$. A fixed population size of 200 was used and the genetic algorithm was run for 30 generations. The values of the different hyperparameters of the genetic algorithm, summarized in table 5.1, were chosen based on the observations from preliminary trials and on the values used in [11]. Since these values resulted in a stable walk in most cases, they were not tuned or modified any further.

Table 5.1: Hyperparameters of the genetic algorithm.

Population size	200
Max. generations	30
Chromosome-type	Real-valued
Selection	Tournament selection with tournament size 3.
Crossover	2-point crossover with random crossover points. Probability of crossover=0.8.
Mutation	Addition of a number drawn from a Gaussian distribution ($\mu = 0.0$, $\sigma = 0.01$). Chromosome mutation probability=0.1. Gene mutation probability=0.05.

5.1.3 Chromosome Structure and Parameter Bounds

The parameters, which are optimized by the genetic algorithm, include the frequency controlling parameter k_f, the angle feedback weight k, and the biases and gains used in the CPG network. The internal parameters of the Matsuoka oscillators, used in the low-level CPG network, are empirically determined (table 4.3)

and so they are not a part of the parameter search. Refer to section 4.2 for more information about the different parameters which are used in the CPG network.

Table 5.2: Limits of the parameters used in the chromosome

Setup	Variable	Description	Lower bound	Upper bound
open loop *angle feedback* *phase reset*	k_f	CPG frequency controller	0.2	1.0
	GAIN1	Sagittal hip gains	0.01	1.0
	GAIN2	Sagittal ankle gains		
	GAIN3	Sagittal knee gains		
	GAIN4	Frontal hip gains		
	GAIN5	Frontal ankle gains		
	GAIN6	Sagittal shoulder gains		
	BIAS1	Sagittal hip biases	-0.6	0.0
	BIAS2	Sagittal knee biases	0.0	0.5
	BIAS3	Sagittal ankle biases	-0.5	0.0
	BIAS4	Sagittal shoulder biases	0.0	1.0
angle feedback	k	Weight of *angle feedback*	-2.5	2.5

The chromosome structure used by the genetic algorithm consists of the CPG network parameters mentioned above. While the *open loop* and *phase reset* setups use the chromosome [k_f, GAIN1, ..., GAIN6, BIAS1, ..., BIAS4] (table 4.2), the *angle feedback* setup uses an additional parameter k and so its chromosome structure is [k_f, GAIN1, ..., GAIN6, BIAS1, ..., BIAS4, k] (table 4.4). The upper and lower limits of these parameters are listed in table 5.2.

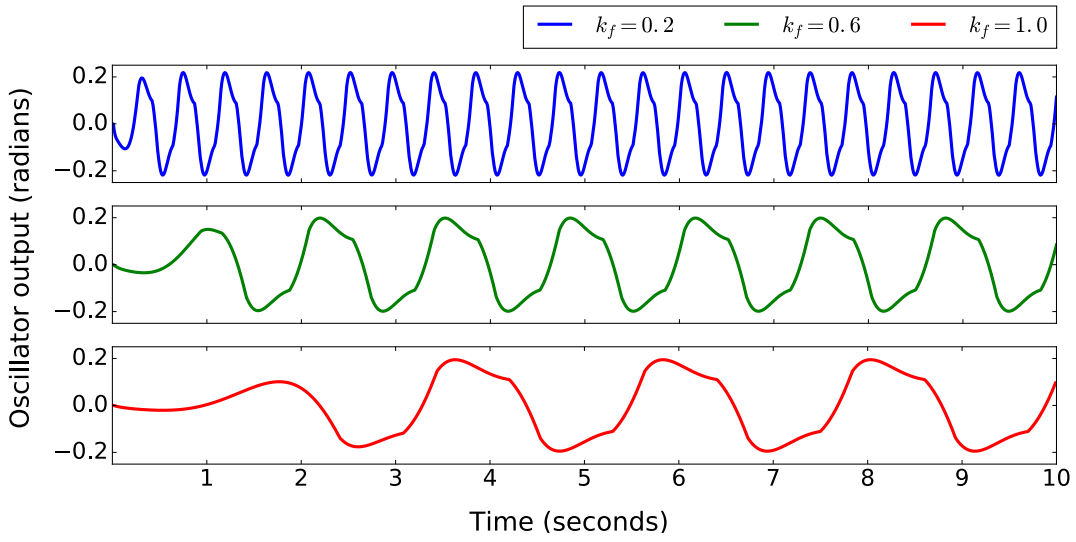


Figure 5.1: Patterns for deciding the limits of parameter k_f.

For setting the limits of k_f, the frequencies of the patterns generated by the pacemaker oscillator for different values of k_f were investigated. The upper plot in

figure 5.1 shows the output of a Matsuoka oscillator when $k_f = 0.2$ (internal parameters were set according to table 4.3). The frequency of the pattern is more than 2Hz (2 complete cycles per second). Frequencies higher than this are practically not attainable with the motors that are used on the robot. The lower plot shows the pattern generated when $k_f = 1.0$. Here, the frequency is approximately 0.4Hz. If k_f is made higher, the frequency will reduce further and the very low speed of joint motion will not be suitable for walking. Values of k_f within the range [0.2, 1.0] produce patterns whose frequencies are suitable for use on a walking robot (an example pattern is shown in the middle plot).

The parameters for gain in the chromosome determine the amplitude of the target angles that are sent to the joints. As can be seen from figure 5.1, the range (difference of highest and lowest value) of the output signal is approximately 0.4 radians (23°). This range is sufficient for a walking motion. Since, here, the oscillator's amplitude was not modified, the gain can be considered to be 1.0. The lower limit of the gain (0.01) results in a very low range of angles (≈ 0 radians). Lower values of gain do not result in any joint motion at all. Hence the limit for the gain parameters was set at [0.01, 1.0].

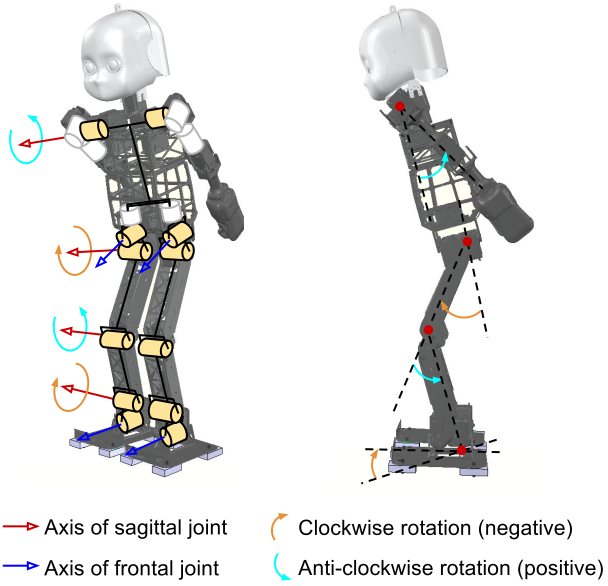


Figure 5.2: Bias Position

The biases, which are used in the CPG network, set the mean position of oscillation of the sagittal joints (whose rotation axes are shown with red arrows in figure 5.2). Biases are not set for the frontal joints, as discussed in section 4.2.1. When all the sagittal joints are set to their respective bias angles, the robot assumes a starting pose similar to that shown in figure 5.2. Only after reaching this start pose, the joints start oscillating according to the output of the corresponding oscillator, and the robot starts to walk. The initial pose of the robot is very important for a stable walk, since, if this pose is unstable, even the best CPG parameters will not be able to prevent a fall as soon as the robot starts. In order to ascertain a stable starting

position, the arms are moved back, the hip bent forward and the knees and ankles bent slightly to move the center of gravity of the robot a little forward and lower. To assume this pose, the mean position of the shoulder and knee joints should be positive (anti-clockwise rotation), and that of the hip and ankle joints should be negative (clockwise rotation), as shown in the right-hand side of figure 5.2. The ranges for the biases in table 5.2 were set to follow this convention. This approach for setting the gains and biases is based on [11]. However, the exact ranges are different as they have been tuned to suit the NICO robot. 5.2.

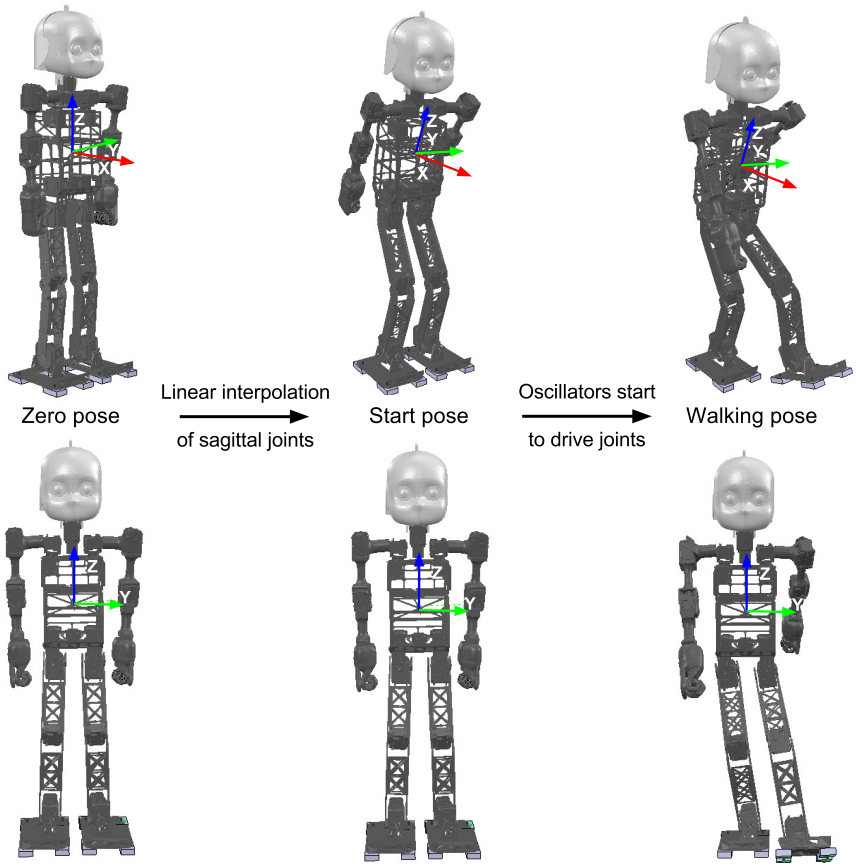


Figure 5.3: Sequence of poses before walking starts. Top: side-on view, bottom: frontal view.

The bias values in the chromosome determine the mean position of all the controlled joints and the oscillations occur about these positions. Since the robot starts from the zero pose (all angles=0 rad.), it first has to reach the start pose (angles=bias values) before it can start walking. The zero and start poses are shown in figure 5.3. Changing angles too fast, in order to assume the start pose, can result in unwanted instability that adversely affects the gait. To prevent this, all the sagittal angles were linearly interpolated from their zero-positions to their bias-positions over a period of 5 seconds. Then, a further 2 seconds were spent to allow any oscillatory movement of the body to subside. After this, the CPG network was turned on, and the individual oscillators were allowed to drive the robot's joints.

The range of the feedback parameter k was set in an empirical fashion. The only guiding principle here was that k should be able to assume both positive and negative values, so that the effects of introducing both positive and negative feedback into the extensor and flexor neurons alternatively (figure 4.5), may be evaluated by the genetic algorithm.

5.1.4 Fitness Function

The fitness function is perhaps the most crucial element of a genetic algorithm. It defines the fitness surface of the problem which the genetic algorithm is trying to solve, and forms the link between the genotype (coded representation of the chromosome) and the phenotype (physical expression in the form of features, produced by the code in the chromosome). In the context of the current problem, the genotype is represented by the configurable parameters of the low-level network. The phenotype is the observable characteristics of the bipedal locomotion that is produced when a particular set of values is assigned to those parameters. It is the fitness function which judges the characteristics of the gait and produces a numerical measure of the quality of the produced motion.

In [11], the fitness of a chromosome was calculated as $f = \alpha(vel) - \gamma(dev)$, where vel is the velocity of the robot in the forward direction, dev is the lateral deviation, and α and γ are weighting coefficients. Although this fitness function would rate highly the chromosomes that result in a fast, straight walk, it ignores the stability of the walk. In the initial generations, there may be chromosomes which result in a very balanced walk but are unable to result in a fast motion that covers much distance. It is important not to reject these chromosomes, but to give them the chance of improving themselves with respect to the distance that can be covered. To do this, the following fitness function has been used in this thesis:

$$f = distance_x + (\lambda \times t_{up}) \tag{5.1}$$

The robot starts at the location (0,0) of the world coordinate frame, facing in the direction given by the x-axis. In equation 5.1, $distance_x$ is the distance (in meters) walked by the robot in the x direction, t_{up} is the time (in seconds) till which it does not fall and λ is a weighting factor (set to 0.5). Since each chromosome is tested for 20 seconds (at the most), the distance term implicitly encourages higher velocities. The term for time contributes to a high fitness score if the robot stays upright for a long time. The robot is considered to be upright as long as the center of its torso is at or above 0.3m from the ground. In the standing position (zero pose in figure 5.3) the center of the torso is approximately 0.66m above the ground.

Since the distance is in meters and the time is in seconds, in the initial generations, the fitness score is dominated by chromosomes which result in higher stability, even if the distance is not significant. As the generations progress and the average fitness of the population increases, most of the chromosomes result in stable gaits. At this point, chromosomes which can also display a longer walk start coming into prominence. Towards the end, the best chromosomes are able to exhibit a stable walk covering a good distance (as shown in the next section).

Given the configuration of the CPG network, which is designed to always produce symmetrical oscillations across the left and right side joints, it is not useful to include a penalty for the lateral deviation in the fitness function. Minimization of lateral deviation is taken care of by the high-level controller.

5.1.5 Results

A basic gait for 3 types of setup (*open loop*, *angle feedback* and *phase reset*) was evolved. For each type, the genetic algorithm was executed 3 times each on 2 computers ($M1$ and $M2$), resulting in a total of (3 $setups \times 3$ $runs \times 2$ $machines$) 18 runs. Only the results from $M1$ are discussed here (results from $M2$ can be found in appendix A).

For each run, the genetic algorithm was allowed to execute for 30 generations. The population of the first generation was initialized randomly, according to the limits described in table 5.2. For evaluating the fitness of a chromosome, first, a CPG network was constructed by setting its parameters according to the genes of the chromosome. The robot was positioned at position ($x = 0, y = 0$) of the world frame, with the robot's x-axis having the same direction as the x-axis of the world frame. After allowing the robot to reach its start pose (figure 5.3), the CPG network was allowed to drive the individual joints, which resulted in a walking motion. The robot was allowed to walk for a maximum time limit of 20 seconds or till it fell down. The fitness of the chromosome was then calculated by using equation 5.1. For each generation, the fitness score of the best chromosome, the distance walked by the robot with that chromosome's values and the average fitness of the generation were tracked. The results of the genetic algorithm for *open loop*, *angle feedback* and *phase reset* are shown in figures 5.4, 5.5 and 5.6 respectively.

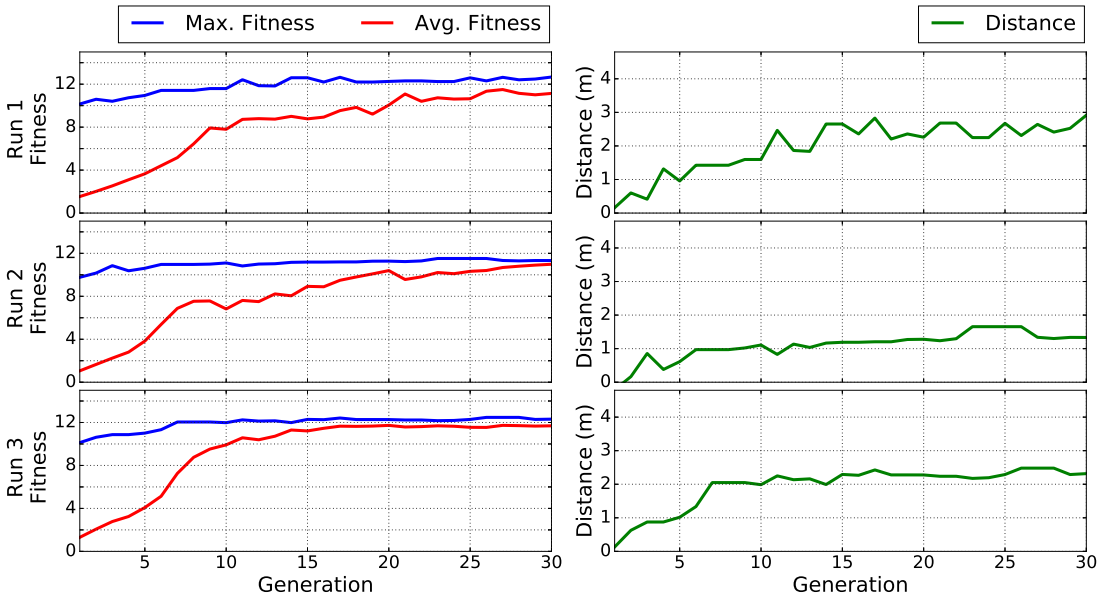


Figure 5.4: **Open loop** results. Left column: Fitness vs. generation, right column: distance vs. generation. Top to bottom: Three separate runs of the genetic algorithm.

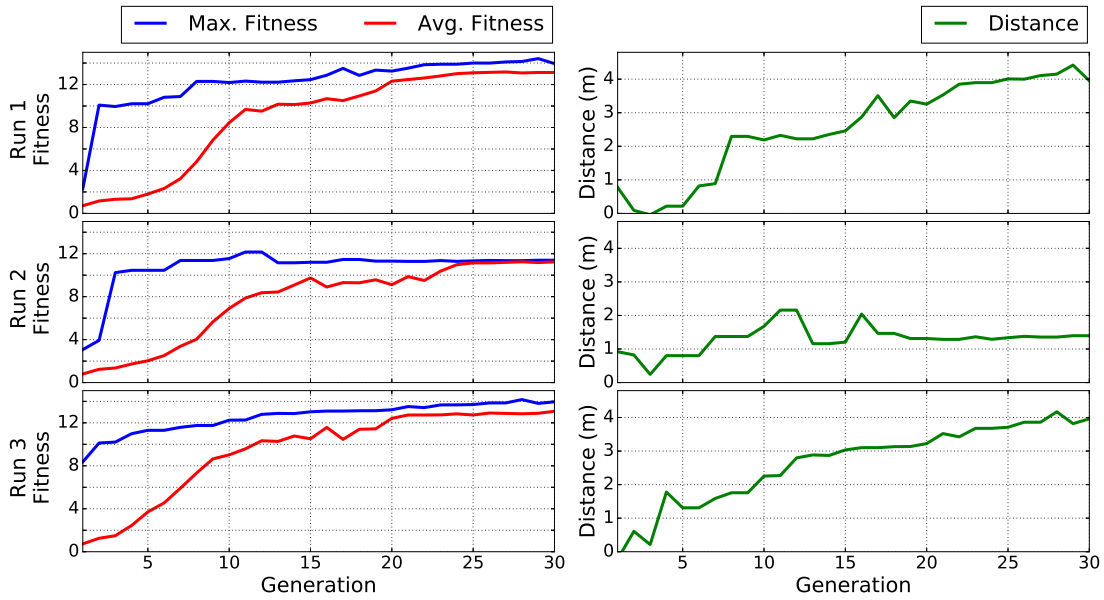


Figure 5.5: **Angle feedback** results. Left column: Fitness vs. generation, right column: distance vs. generation. Top to bottom: Three separate runs of the genetic algorithm.

It can be seen from the plots that the maximum fitness (blue line) of the initial and final generations differs by a small amount in most cases. This is because the fitness score f is dominated by t_{up} ($f = distance_x + 0.5 \times t_{up}$), which attains its best possible value of 20s very early on in the evolution process. The maximum possible contribution of t_{up} to f is $0.5 \times 20 = 10$. Any fitness score above 10 is surely due to $distance_x$. Since, in 20s the robot walks around 4m at the most, the increase in fitness is small.

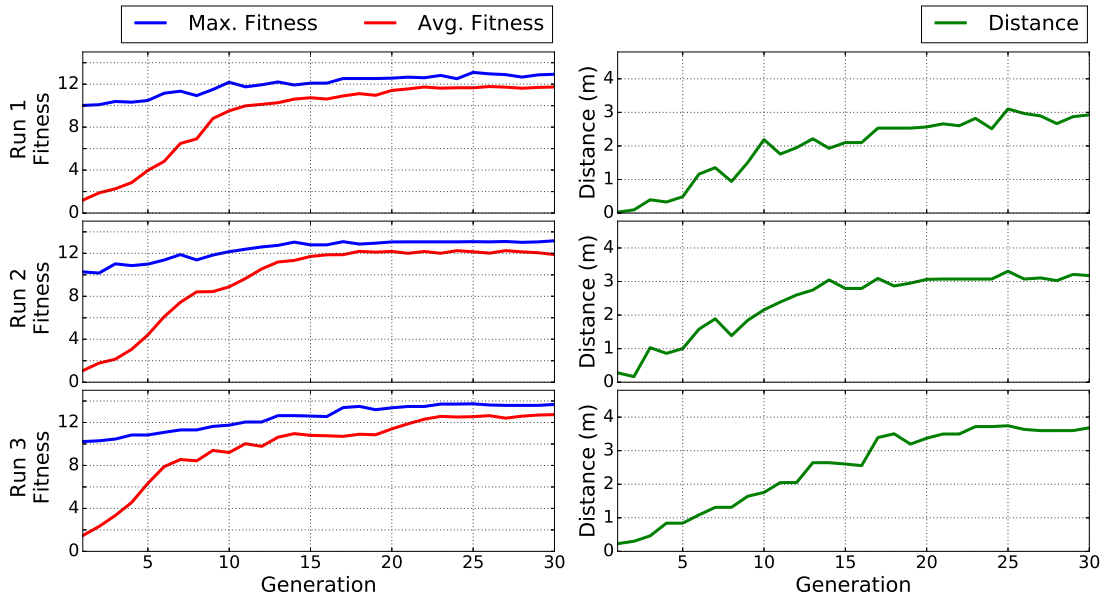


Figure 5.6: **Phase reset** results. Left column: Fitness vs. generation, right column: distance vs. generation. Top to bottom: Three separate runs of the genetic algorithm.

However, the fact that the genetic algorithm is able to gradually improve the fitness of the population, is proven by the sharp increase in the average fitness of the population (red line) over the generations. In the initial generations, the average fitness is near 0, but towards the end, it is very close to the best fitness, as indicated by the proximity of the red and blue lines. The plots for distance (green line) also show that the best chromosomes of the latter generations achieve a much better forward distance ($distance_x$) than the initial generations. The chromosomes in the initial generations have a high fitness score but achieve very low distances whereas, in the later generations the fitness scores increase slightly but there is a sharp increase in the forward distance. This shows that ultimately, it is the distance that differentiates the best chromosomes from the good ones.

While running the genetic algorithms, an interesting observation was made about the effect of the different genes. The first action of the robot is to reach the start pose (figure 5.3) and this is determined by the bias values in the chromosome. In the initial generations, the biases of most chromosomes could not result in a stable start pose and the robot fell even before it started its walking movement. As the generations progressed, chromosomes with better values of bias were evolved and the start pose of the robot was stable for many chromosomes. In the final generations, almost every chromosome resulted in a stable start pose due to good bias values. Thus, before good values for bias were found, the other genes of the chromosome did not even come into play. Only after stable biases were found, did the evolution of the other genes in the chromosome start. As a result, in the initial generations, most often the robot could neither stand nor walk, in the middle generations it could stand but not walk and in the final generations, it could stand as well as walk.

Table 5.3: Best performance in the 30^{th} generation.

	Open loop			Angle feedback			Phase reset		
	run 1	run 2	run 3	run 1	run 2	run 3	run 1	run 2	run 3
Best distance (m)	2.92	1.33	2.31	3.95	1.40	3.96	2.93	3.18	3.68
Best time (s)	20	20	20	20	20	20	20	20	20
Best fitness	12.92	11.33	12.31	13.95	11.40	13.96	12.93	13.18	13.68

For the three different setups, the performance of the genetic algorithm in the last generation is listed in table 5.3. From the information in this table and the plots of the fitness and distance, an important difference in the performance of the setups can be observed. For *open loop* (figure 5.4) and *phase reset* (figure 5.6), the best fitness score of the very first generation is around 10 for all the three runs. This indicates that the best chromosome in the first generation kept the robot upright for nearly the whole of 20s (since $fitness = distance_x + 0.5 \times t_{up}$, and $distance_x$ is nearly 0 for the first generation, which implies $t_{up} \approx 20$s). For *angle feedback* (figure 5.5), the first generation's best fitness score is significantly lower than 10 in all the three runs. This suggests that until suitable values were found for the feedback weights, the feedback mechanism made the system less stable.

However, once suitable values were found, the same feedback mechanism enabled the robot to cover much higher distances. In the final generation of 2 out of 3 runs, *angle feedback* achieved a best distance of 3.95 and 3.96 meters, whereas the best distances in the final generations of the other setups were much lower (table 5.3).

The performance of *angle feedback* in run 2 was much worse than its other two runs. The best distance was only 1.40m (figure 5.5). Here, it seems that the optimization process was stuck at a local optimum. A look at the solutions found by the genetic algorithm (table 5.4), reveals that for run 2, the value of the feedback weight is negative (-0.6130), while for the other two runs it is positive (0.7654 and 1.5364). This suggests that a large negative feedback weight is not beneficial. The value of k_f for run 2 was also much higher, resulting in a lower frequency and lesser distance covered. Similarly, for run 2 of *open loop*, where the performance was comparatively very poor, k_f has a higher value.

It can also be seen from table 5.3, that when some form of low-level feedback was used (*angle feedback* or *phase reset*), the best distance in most cases was higher than when no feedback was used (*open loop*). This fact points toward the beneficial effects of using a low-level feedback loop.

Table 5.4: CPG network parameters found by the genetic algorithm (genotype of the chromosomes); BIASi is in radians; k_f, GAINi and k do not have any units.

	Open loop			Angle feedback			Phase reset		
	run 1	run 2	run 3	run 1	run 2	run 3	run 1	run 2	run 3
k_f	0.4488	0.8144	0.4470	0.2258	0.7594	0.3178	0.3158	0.2840	0.2496
GAIN1	0.4167	0.7825	0.6213	0.5178	0.7708	0.3777	0.6792	0.4846	0.5626
GAIN2	0.1692	0.3372	0.0866	0.0101	0.0813	0.0234	0.0134	0.0469	0.0174
GAIN3	0.1861	0.0489	0.2071	0.0245	0.1381	0.0132	0.1478	0.0236	0.0140
GAIN4	0.6941	0.6607	0.7099	0.4555	0.8223	0.4567	0.4343	0.3094	0.3939
GAIN5	0.1372	0.0636	0.0818	0.2146	0.0223	0.2019	0.2060	0.2936	0.1845
GAIN6	0.5818	0.7723	0.4866	0.8918	0.2309	0.3309	0.3598	0.0883	0.9518
BIAS1	-0.1634	-0.0666	-0.1714	-0.1538	-0.0046	-0.0519	-0.1332	-0.2375	-0.3392
BIAS2	0.0356	0.3010	0.4983	0.1841	0.0895	0.0963	0.4287	0.2509	0.2576
BIAS3	-0.0147	-0.1895	-0.2999	-0.0973	-0.1217	-0.1156	-0.2780	-0.1445	-0.1045
BIAS4	0.0175	0.1774	0.0575	0.0107	0.5806	0.4814	0.5847	0.2630	0.0775
k	NA	NA	NA	0.7654	-0.6130	1.5364	NA	NA	NA

5.2 Gait Analysis

Although the results obtained from the evolution process indicate that feedback is beneficial and that *angle feedback* performs better than the other setups, they do not provide conclusive evidence to support this suggestion. In order to have more evidence of the comparative benefits of the different setups, each solution found by the genetic algorithm was analyzed by putting it through an extensive

performance test. The solutions of the genetic algorithm consist of 9 different chromosomes, shown in the columns of table 5.4 (one chromosome for each of the three runs of *open loop*, *angle feedback* and *phase reset*). These chromosomes, which evolved after 30 generations, result in gaits with different characteristics. Some of these gaits exhibit the ability to cover a long forward distance, while others exhibit greater stability during walking. In order to run a performance test of the evolved chromosomes, first, a set of characteristics needs to be chosen which can be used to objectively differentiate the different setups which the gaits are associated with.

5.2.1 Evaluation Parameters and Setup

The fitness score, f (obtained by equation 5.1), is an obvious indicator of the merit of a chromosome since it was used to choose fitter chromosomes during the evolution process. The fitness score is calculated by using the values of the forward distance ($distance_x$) and the time the robot was upright (t_{up}). Hence, these two parameters are also evaluated independently in order to judge a chromosome. Although the low-level control mechanism does not control the lateral deviation of the robot, the absolute value of the lateral deviation (dev_y^{abs}) is chosen as a parameter to determine if the different setups result in the same kind of deviation or not. The length of the stride is another important characteristic feature that can be used to compare gaits. Here, the length of a stride is calculated as the difference between the successive locations of the left foot and the right foot as the robot walks. For the whole duration of the walk, the length of each stride is calculated, and then the average stride length $stride^{avg}$ is calculated by dividing the sum of the stride lengths by the number of steps taken. Apart from this, the stability of the robot's torso during the walk can be used to differentiate gaits. Stable gaits will show comparatively small variance in the angular position of the torso. Hence the variances in the average torso orientations, with respect to the world frame, are also used in this analysis. The three orientation parameters which are used are: $torso_\alpha^{var}$, $torso_\beta^{var}$ and $torso_\gamma^{var}$ - the variance in the torso's orientation about the world's x, y and z-axis respectively. These evaluation parameters are summarized in table 5.5.

Table 5.5: Parameters used for evaluating gaits.

Parameter	Description
f	Fitness score calculated according to equation 5.1
$distance_x$	Distance covered by the robot along the world frame's x-axis
dev_y^{abs}	Absolute value of the lateral deviation along the world frame's y-direction
$stride^{avg}$	Average length of a stride during the walk
t_{up}	Time for which the robot did not fall during the walk
$torso_\alpha^{var}$	Variance of the torso's orientation about the world's x-axis during the walk
$torso_\beta^{var}$	Variance of the torso's orientation about the world's y-axis during the walk
$torso_\gamma^{var}$	Variance of the torso's orientation about the world's z-axis during the walk

At the end of 30 generations, 9 chromosomes are obtained from the 3 runs of the 3 setups. These are named OL_1, OL_2 and OL_3 (for runs 1, 2 and 3 of *open loop*), AF_1, AF_2 and AF_3 (for runs 1, 2 and 3 of *angle feedback*), and PR_1, PR_2 and PR_3 (for runs 1, 2 and 3 of *phase reset*). For each chromosome, the CPG network was constructed by using the values in the chromosome's genes, and the robot was allowed to walk for a maximum time of 20s unless it lost balance and fell before that time. During the walk, the different parameters discussed above were computed. This process was repeated 100 times for each chromosome. Similar to the genetic algorithm runs, the robot started at the (0,0) position of the world frame, with it's x-axis in the direction of the world frame's x-axis. Starting from the zero pose, the sagittal joints were then linearly interpolated over 5s to assume the start pose (figure 5.3). After waiting another 2s for stabilizing, the robot started walking.

5.2.2 Results

Median values of the gait evaluation parameters, obtained after evaluating the 9 chromosomes 100 times each, are summarized in table 5.6. Since means are strongly affected by outliers in the data, in this analysis, medians have been used instead. The scores of the different parameters are plotted using boxplots in figures 5.7-5.12, where results of *open loop*, *angle feedback* and *phase reset* are denoted with red, blue and green boxes respectively. Medians are indicated by vertical black lines within the boxes. The left and right edges of the boxes denote the lower ($Q1$) and upper quartiles ($Q3$) of the data. The whiskers of the boxplots show the data range given by $[(Q1 - 1.5 \times IQR), (Q3 + 1.5 \times IQR)]$ where IQR is the inter-quartile range. Outliers are shown with grey circles.

Table 5.6: Median values of the gait evaluation parameters over 100 trials.

Parameter	Open loop			Angle feedback			Phase reset		
	OL_1	OL_2	OL_3	AF_1	AF_2	AF_3	PR_1	PR_2	PR_3
f	11.984	11.174	11.852	13.390	11.288	13.519	11.979	11.478	13.049
t_{up} (s)	19.105	20.000	19.570	20.000	20.000	20.000	20.000	17.990	20.000
$distance_x$ (m)	2.345	1.178	2.038	3.394	1.293	3.544	2.009	2.091	3.066
dev_y^{abs} (m)	0.209	0.391	0.524	1.164	0.107	0.889	0.961	0.312	0.451
$stride^{avg}$ (m)	0.060	0.054	0.121	0.059	0.080	0.100	0.057	0.110	0.096
$torso_\alpha^{var}$ (rad^2)	0.021	0.008	0.011	0.004	0.011	0.007	0.003	0.013	0.005
$torso_\beta^{var}$ (rad^2)	0.012	0.003	0.006	0.002	0.002	0.001	0.002	0.008	0.001
$torso_\gamma^{var}$ (rad^2)	0.025	0.084	0.059	0.084	0.013	0.051	0.222	0.063	0.055

The fitness score f over 100 trials of the 9 chromosomes are summarized in figure 5.7. The median scores of the 3 *angle feedback* chromosomes (AF_1, AF_2 and AF_3) are 13.39, 11.288 and 13.519 respectively. Apart from the second chromosome, where the optimization was stuck at a local maximum, these scores are higher than the median scores of the other setups. The only one which comes close is the third chromosome (PR_3) of *phase reset* with a median score of 13.049. The fitnesses for

phase reset are marginally better than those of *open loop*. Also, over a 100 trials, *angle feedback* shows the least amount of variability in the fitness, as shown by the length of the blue boxes (the inter-quartile range) in figure 5.7.

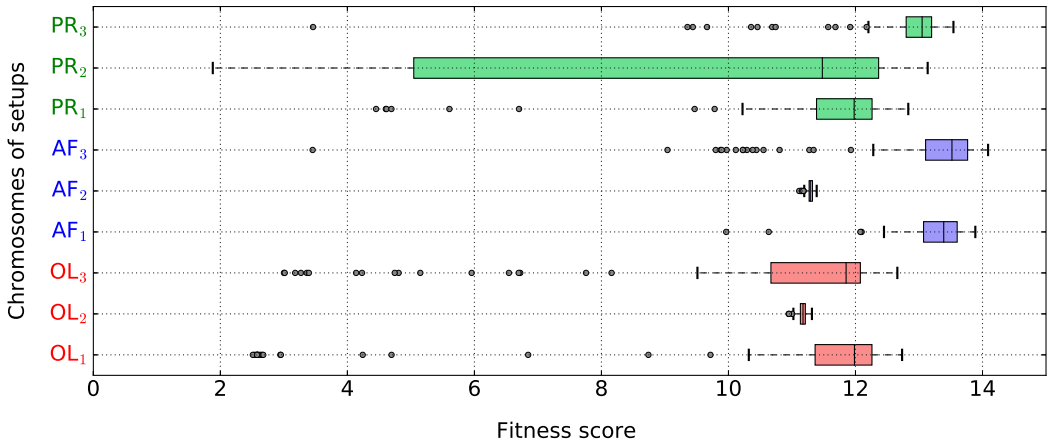


Figure 5.7: Fitness f for 100 trials.

When the forward distance ($distance_x$) was measured for the 3 setups, here also, 2 of the 3 *angle feedback* chromosomes (AF_1 and AF_3) show a better performance than the chromosomes of the other setups (left-side plot in figure 5.8). The median distance achieved by AF_1 and AF_3 was 3.394m and 3.544m respectively, whereas the next best median distance was only 3.066m, achieved by PR_3 (table 5.6). The median distances of AF_1 and AF_3 are also close to the distances (3.95m and 3.96m respectively) these chromosomes achieved in the 30^{th} generation of the evolution process (table 5.3). The stochastic nature of the environment results in variability of the distance covered when the same chromosome is used multiple times. However, as can be seen in the left plot in figure 5.8, the variability in forward distance for *angle feedback* is not very large.

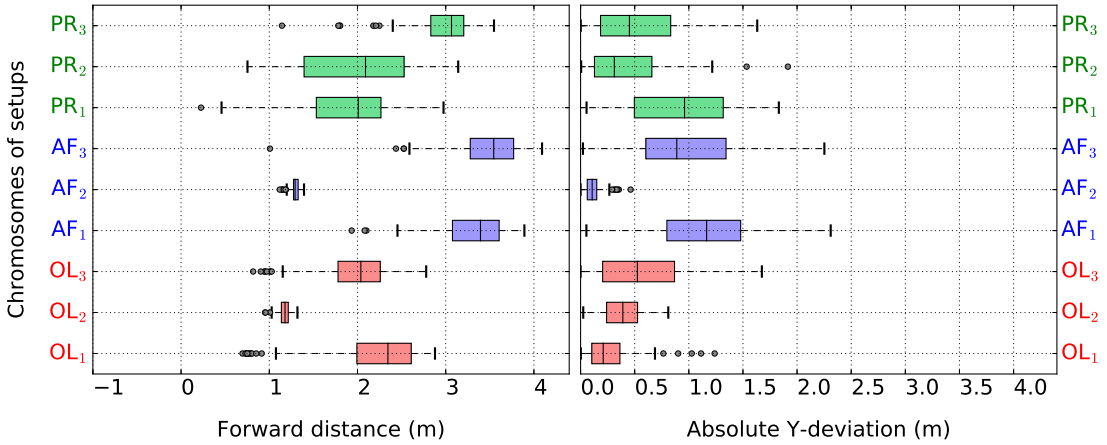


Figure 5.8: (left) The forward distance $distance_x$ and (right) the absolute lateral deviation dev_y^{abs} for 100 trials.

Since the low-level CPG-based controller does not explicitly control the robot's deviation, it is expected that if the robot walks further, its deviation will also be greater. In the right-side plot in figure 5.8, it can be seen that the median values of dev_y^{abs} for AF_1 and AF_3 are relatively high. However, there does not appear to be a direct relationship between distance and deviation for all the cases. PR_1, which has a comparatively low median distance, has a high median value for deviation.

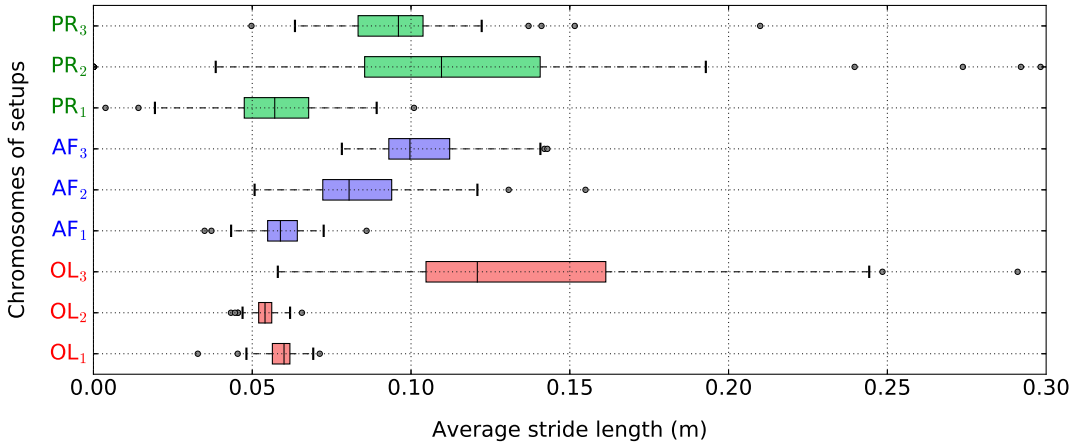


Figure 5.9: Average length of stride $stride^{avg}$ for 100 trials.

The average length of a stride ($stride^{avg}$) is a good descriptor of a gait. A long stride can result in a longer distance, but it can also make the robot less stable. A shorter stride can result in more stability but less forward distance. As shown in figure 5.9, all the chromosomes result in relatively short strides (given the NICO is 101cm tall). The variability in stride length for *angle feedback* is lower than for *phase reset*. The *open loop* chromosome OL_3 and the *phase reset* chromosome PR_2 show a high variability in stride length. The *open loop* chromosome OL_2 has a low median score for average stride length, which is perhaps a reason why this chromosome has a very low median score for the forward distance as well (left-side plot of figure 5.8). PR_2, which has one of the largest median stride lengths, achieved the lowest median time of 17.99s for staying upright (figure 5.10).

The stability of the robot is an important criterion for bipedal locomotion. If the low-level controller, which does not control stability explicitly, can result in a stable walk, then any high-level balance controller will have a much easier job. A direct indicator of the gait stability of the different chromosomes is the time (t_{up}) for which the robot stayed upright during the walk. It can be seen in figure 5.10, that for all the 3 *angle feedback* chromosomes, the median score for t_{up} is 20.0s. Since the maximum time allowed for walking was 20s, this means that the robot fell down very few times in the all the trials for AF_1, AF_2 and AF_3. The *phase reset* setup is the second-best performer, where PR_1 and PR_3 have a median score of 20s. However, PR_2's median t_{up} is only 17.99s and its variability is also quite high. The setup without any feedback (*open loop*) has a median score of 20s for only one of its three chromosomes.

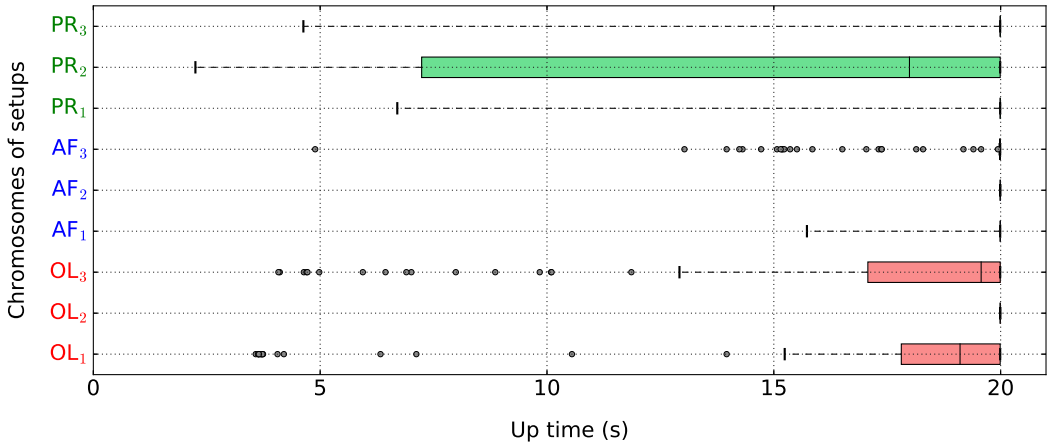


Figure 5.10: Up time t_{up} for 100 trials. When Q1=Q3=median, the boxes have zero width.

In addition to t_{up}, the change in orientation of the robot's torso is also a good indicator of the stability of motion. When the torso orientation does not change much, it is expected that the robot's motion will be smoother. Maintaining balance would also be easier in such a case. The variances in the torso orientations ($torso_{\alpha}^{var}$, $torso_{\beta}^{var}$ and $torso_{\gamma}^{var}$), about the three coordinate axes of the world frame, are plotted in figures 5.11 and 5.12. The first noticeable thing in these plots is the difference in the x-axis scales between the figures. The angles $torso_{\alpha}$ and $torso_{\beta}$ are the orientations of the robot's torso about the world x and y axes respectively. Thus, $torso_{\alpha}$ is related to the left-to-right swaying motion in the frontal plane and $torso_{\beta}$ is related to the forward and backward swaying motion in the sagittal plane. These two types of motion are rather restricted for all the three setups. However, as the robot shows quite a bit of lateral deviation, it's heading direction also changes significantly. This change is captured by $torso_{\gamma}$, which denotes the orientation of the torso about the world z-axis. Hence $torso_{\gamma}$ assumes much greater values than $torso_{\alpha}$ and $torso_{\beta}$ in all the cases, and so its variance $torso_{\gamma}^{var}$ has a bigger range.

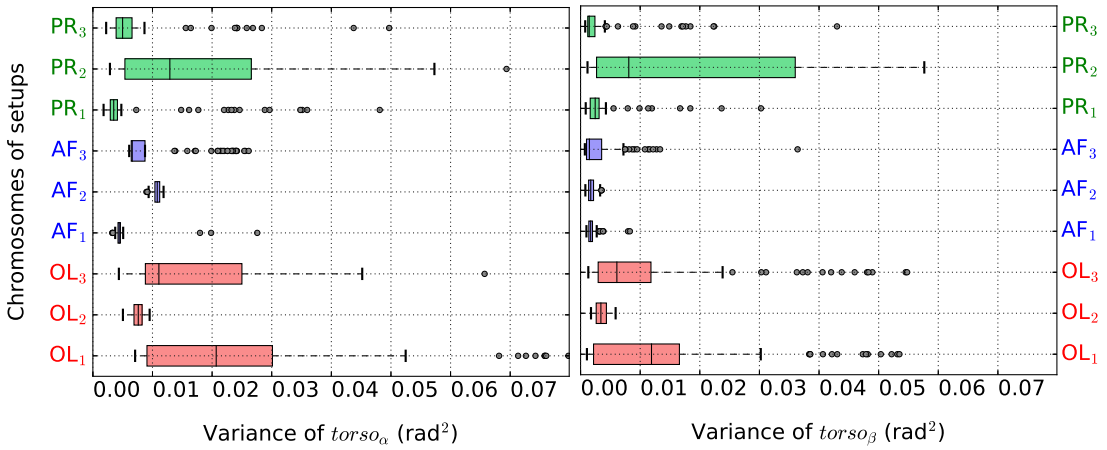


Figure 5.11: (left) $torso_{\alpha}^{var}$ - variance in the torso's orientation about the world x-axis, (right) $torso_{\beta}^{var}$ - variance in the torso's orientation about the world y-axis, for 100 trials.

From the left and right plots in figure 5.11, it can be seen that the *angle feedback* chromosomes, in general, have the lowest median scores for the variance in torso orientation. The variability in the scores for *angle feedback* is also significantly lower than the other two setups (judged by the distance between the whiskers of the boxplots). The angle $torso_\gamma$ is related to the deviation of the robot, which in turn, is related to the distance walked by it. Hence it would be expected that the chromosomes AF_3 and AF_1, which achieved the best distances would have the highest scores for $torso_\gamma$ as well. However, as can be seen in figure 5.12, these two chromosomes do not result in the worst variance in orientation about the z-axis. Overall, *phase reset*'s median scores are the worst in this respect. The variability in the variance over 100 trials is also quite high for *phase reset*.

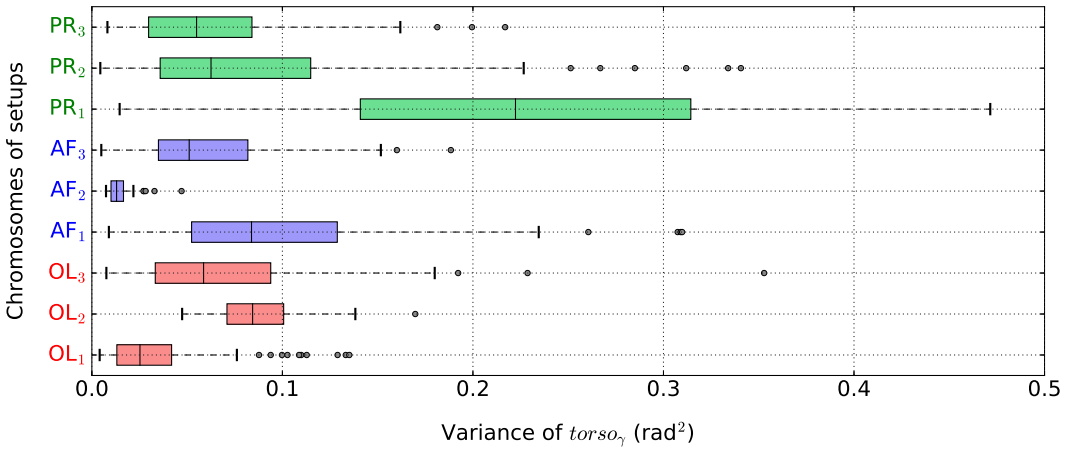


Figure 5.12: $torso_\gamma^{var}$ - variance in the torso's z-axis orientation for 100 trials.

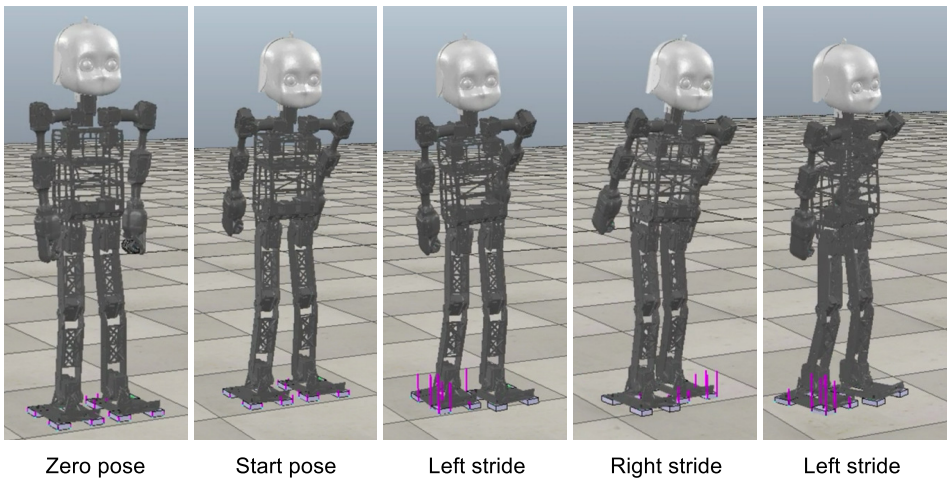


Figure 5.13: Snapshots of the walking motion from one of the gait evaluation trials using *angle feedback*. The purple lines on the support foot (the foot which is on the ground) indicate the vertical ground reaction forces acting on the pressure sensors on the foot.

5.3 Discussion

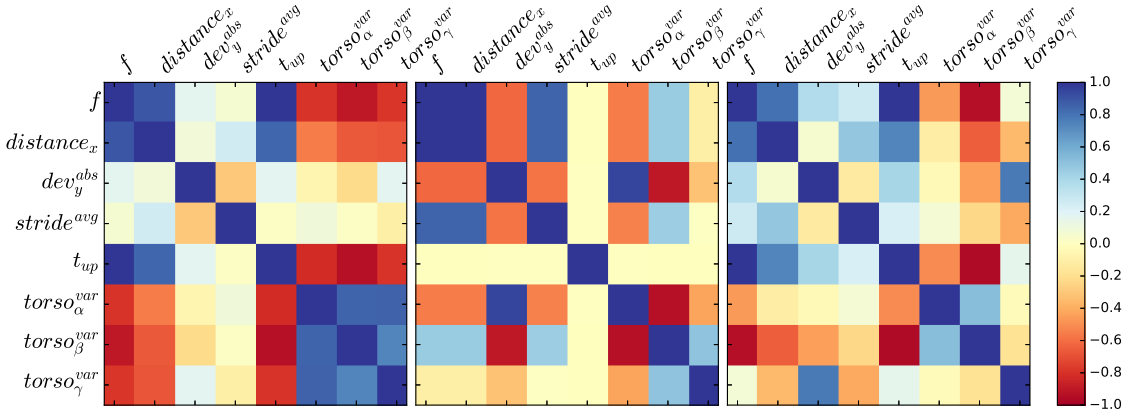
The gait evaluation parameters were chosen so that they would numerically represent some characteristics of the gait. Thus, for a good chromosome, the metrics related to desirable characteristics such as stability should have a high value, and those for undesirable characteristics such as lateral deviation should have a low value. However, in order to show that these evaluation metrics are actually useful in judging a gait, the relationship of these parameters with the fitness score parameter needs to be analyzed. When the scores for the different metrics are compared with the fitness score and with each other, by using Pearson's correlation coefficient r [19] (equation 5.2), some relationships can be seen.

$$r = \frac{n(\sum xy) - (\sum x)(\sum y)}{\sqrt{[n\sum x^2 - (\sum x)^2][n\sum y^2 - (\sum y)^2]}} \tag{5.2}$$

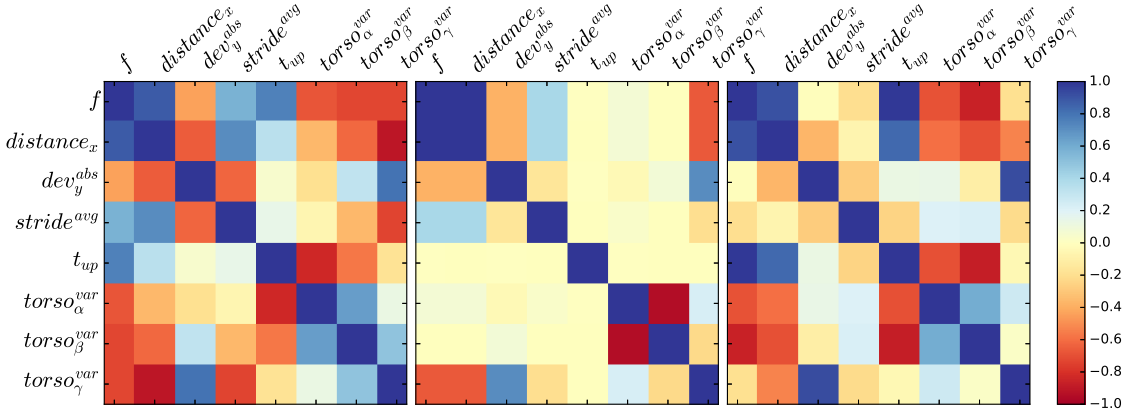
Here, $n = 100$ is the number of data points, and x and y are the values of the two parameters being compared. Correlation matrices were created for each chromosome by using the values of the respective evaluation parameters (table 5.5). Although the correlation coefficient does not indicate causality, it does provide insights into how different parameters are related. Figure 5.14 shows the correlation matrices for the 9 chromosomes used during the gait evaluation process.

In the figure, shades of blue indicate positive correlation and shades of red indicate negative correlation. Darker shades and lighter shades indicate high and low correlation respectively. In most of the correlation matrices, the fitness score f has a high positive correlation with the forward distance $distance_x$ and the up-time t_{up}. This is expected, since $distance_x$ and t_{up} are used to calculate f. However, it is interesting to see the relationship of the other parameters with f. For example, the parameters for variance in torso orientation, $torso_{\alpha}^{var}$ and $torso_{\beta}^{var}$, show a high negative correlation with f, $distance_x$ and t_{up} for almost all the chromosomes. This means that whenever $torso_{\alpha}^{var}$ and $torso_{\beta}^{var}$ have low values, $distance_x$ and t_{up} have high values (and vice-versa). This supports the idea that low variance in orientation is good for the gait. The correlation of $torso_{\gamma}^{var}$ with f, $distance_x$ and t_{up} also shows a similar, but less strong, correlation.

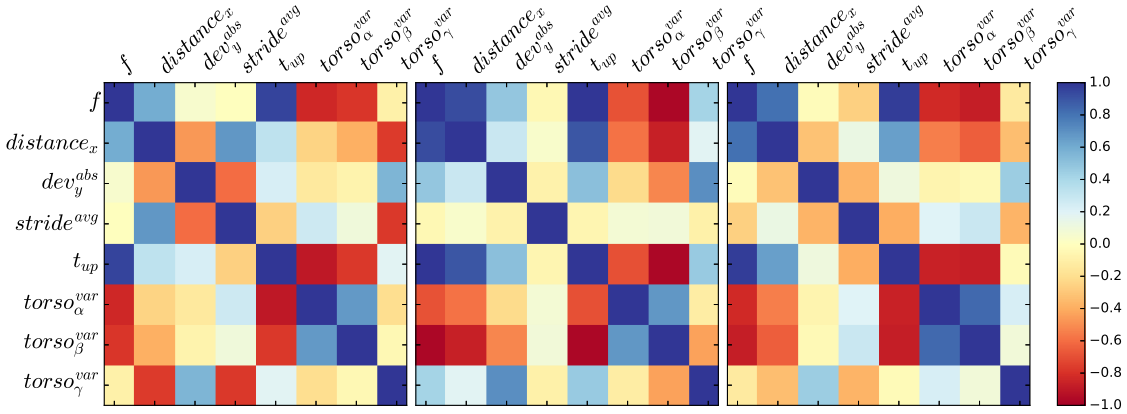
The correlation of the average stride length $stride^{avg}$ with f, $distance_x$ and t_{up} shows different trends for the different setups. For *open loop* (plots in the first row of figure 5.14), $stride^{avg}$ has a weak positive correlation with f, $distance_x$ and t_{up}. This suggests that, when no feedback is used, the robot takes long steps whenever it is able to walk for a long distance (or vice-versa). However, for *angle feedback* and *phase reset* (plots in the second and third rows of figure 5.14), the relationship of $stride^{avg}$ with f, $distance_x$ and t_{up} is less clear because both positive and negative correlations occur. The parameter for lateral deviation, dev_y^{abs}, shows a moderate amount of negative correlation with f, $distance_x$ and t_{up} for *angle feedback* (plots in the second row of figure 5.14). However, for the two other setups, there is no definite pattern. For AF_2 (middle plot in the second row of figure 5.14), few strong correlations exist. This chromosome did not exhibit the high performance of the other *angle feedback* chromosomes.



Correlation matrices for *open loop* chromosomes. (left) OL_1, (middle) OL_2, (right) OL_3.



Correlation matrices for *angle feedback* chromosomes. (left) AF_1, (middle) AF_2, (right) AF_3.



Correlation matrices for *phase reset* chromosomes. (left) PR_1, (middle) PR_2, (right) PR_3.

Figure 5.14: Correlation between the gait evaluation parameters.

The results presented in section 5.1.5 demonstrate the feasibility of the low-level, CPG-based controller for walking. All the 9 chromosomes which were evolved, could make the robot walk. However, as can be seen from the forward distance achieved in the 30^{th} generation (table 5.3), *angle feedback* performed the best, followed by *phase reset* and *open loop*. When the chromosomes of the different setups were put through a performance test, the same general trends in the results were repeated. Since during this test, each chromosome was tested 100 times, the results in section 5.2.2 provide more evidence about the superiority of the *angle feedback* setup. The chromosomes for *angle feedback* resulted in gaits in which the robot covered more forward distance and stayed upright for a longer time, compared to the other setups. The performance of the *phase reset* chromosomes was worse than *angle feedback* but marginally better than *open loop*, in terms of forward distance and up time. They also showed a lot of variability in the results (represented by the spread of the data in the plots in section 5.2.2).

Thus, from the results presented in this chapter, it may be concluded that the *angle feedback* mechanism does improve the overall quality of the gait produced by the CPG network. Hence it was chosen as the feedback mechanism of the low-level controller, on top of which the high-level controller was built.

Chapter 6

High-level Control

In the hierarchical bipedal controller developed in this thesis, the low-level controller's role is to produce oscillatory motion in the robot's joints, while also adapting these oscillations to the way the robot interacts with the environment. The high-level controller, on the other hand, is concerned with meeting a higher overall objective, which in this case, is to minimize the lateral deviation while walking. A high-level controller gives the robot the ability to modify its movements by using only a few parameters which ultimately influences the overall behavior of the low-level CPG controller. The use of a few control parameters simplifies the training process of the high-level controller, while also maintaining a close similarity to the locomotion control mechanism found in animals.

The structure of the high-level neural network controller and the way it achieves deviation control by modifying CPG network parameters were discussed in detail in chapter 4. This chapter covers the details of how the high-level controller was created and trained using reinforcement learning. This chapter also discusses how hyperparameters were set and how the reward function, that is used in the reinforcement learning setup, was designed. Also included, are the results of experiments which demonstrate the effectiveness of this controller.

6.1 Problem Formulation

Recall from section 2.4.1, that the reinforcement learning problem is concerned with learning the correct behavior of an agent interacting with an environment so that a long-term numerical reward can be maximized. At each step, the agent takes an action, and as a result of that action, it gets in return an observation of the current state and a reward from the environment. In an episodic reinforcement learning setup, the agent's overall experience is broken up into a series of episodes. Each episode begins with the environment and the agent in an initial state and continues until some termination criterion is met. After an episode is terminated, the environment and agent are reinitialized and the next episode begins. The goal, in the episodic setup, is to maximize the expected total reward per episode and also to be able to achieve a high level of performance in as few episodes as possible.

6.1.1 Agent and Environment

Since the high-level controller is trained using an episodic reinforcement learning setup, the first step is to formulate the problem of deviation control as a reinforcement learning problem. To do so, exact definitions need to be provided for what constitutes an agent, environment, action, reward and state.

Since the basic task here is concerned with learning the correct behavior of a robot functioning in an environment, it is tempting to consider the robot to be the agent in the reinforcement learning setup. However, the boundary of separation between the agent and the environment need not be based on the boundary between the robot and its physical surrounding. During the training process, the setup for the training algorithm, including the high-level controller, acts as the agent since it receives the state of the environment and performs actions which lead to a new state and a new reward. Once the training is complete, reward signals are no longer needed, but the high-level controller continues to act as the agent by observing the states and generating actions accordingly. The training process of the high-level controller does not modify the structure of the low-level CPG network in any way. Hence, everything outside the purview of the high-level controller is considered to be the environment. This includes the low-level controller, the robot and its three-dimensional surrounding.

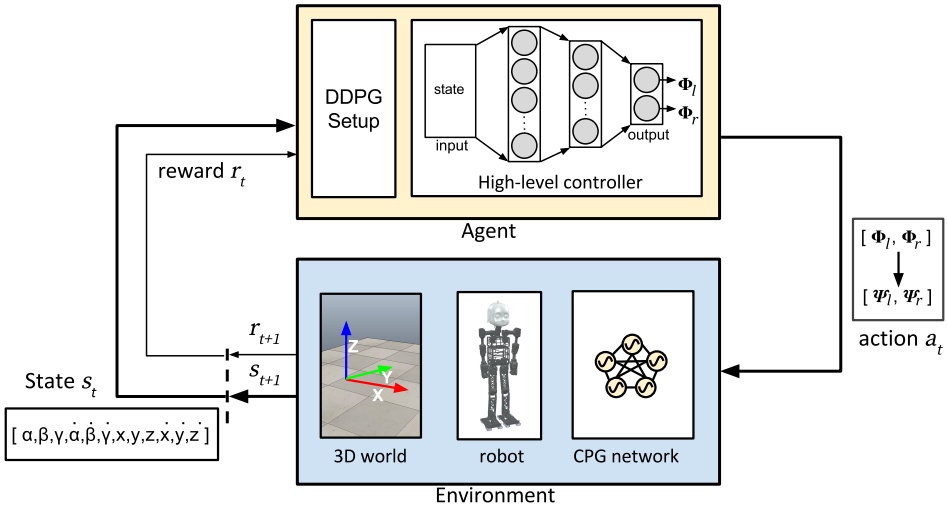


Figure 6.1: Reinforcement learning setup.

Once the agent and environment are defined, it is easy to visualize what constitutes the action and the state. The action is the output of the high-level neural network, consisting of the *gain suppression factors* Φ_l and Φ_r (when the value of Φ_l or Φ_r is close to 1.0, the forward motion of the corresponding leg is maximally suppressed). These are converted into the *gain factors* Ψ_l and Ψ_r using equation 4.4 (section 4.3.2 contains detailed explanation of these factors and the conversion). The *gain factors* are then used to modulate the CPG network's output. The state is the 12-element vector consisting of the angular position (α, β, γ) and velocity $(\dot{\alpha}, \dot{\beta}, \dot{\gamma})$, and the cartesian position (x, y, z) and velocity $(\dot{x}, \dot{y}, \dot{z})$ of the robot's

torso in the three dimensions. During training, the reward is a numerical measure which tells the agent how good its performance is. Figure 6.1 depicts the different components of the reinforcement learning setup, as applicable to the current context.

6.1.2 Design of Reward Function

In a reinforcement learning setup, there are no 'labeled' correct actions for every observation, as in supervised learning. Instead, for every action taken by the agent, it receives a numerical reward signal telling it whether the action taken was beneficial to the overall objective or not. Over the course of time, the agent tries to maximize the accumulated reward and thereby learns the behavior that it was being trained to exhibit. Given the crucial role of the reward signal, the logic for calculating it must be designed to accurately quantify the agent's performance.

Rewards can be calculated in a number of different ways. In goal-directed tasks, such as solving a maze or playing a game against an opponent, the common approach is to provide a large positive reward when the goal is reached or the game is won. If not, then at every timestep, the agent receives either a zero reward or a small negative reward. Negative rewards are particularly helpful in training the agent to achieve the desired behavior in the shortest possible timeframe because if the agent can achieve its goal in a short time, the sum of negative rewards is less and the overall sum of rewards is maximized.

In situations where the agent has to deal with continuous environments, such as in robotics, the reward at each timestep is usually chosen to be a real number. This number can be a combination of positive values representing desirable characteristics resulting from the agent's actions, and negative values representing undesirable characteristics. Since the purpose of the high-level controller is to minimize the lateral deviation of the robot, the obvious negative characteristic that can be included in the reward at each timestep is the amount of lateral deviation dev_y. As lateral deviation is undesirable in either direction, the negative of the absolute lateral deviation (in meters) was included in the reward ($reward = -dev_y^{abs} = -abs(dev_y)$). In addition to this, it is also imperative that the agent learns that a fall during the walk must be avoided. Hence a large negative reward of -100 was included in the reward if the robot fell down. However, when this reward signal was used for training, the trained agent prevented any kind of forward movement by keeping the *gain suppression factors* Φ_l and Φ_r close to 1.0 irrespective of the environment's state. Since the forward movement was minimal, the lateral deviation was also minimal and the robot did not fall. As a result, the accumulated reward was maximized, but the correct behavior did not emerge.

In addition to penalizing the agent for lateral deviation and instability (when the robot falls), it is equally important to reward it for the forward distance covered by the robot. This was done by modifying the reward function to $reward = -dev_y^{abs} + distance_x$. However, in doing so, another kind of undesirable behavior was exhibited by the trained agent. In this case, the agent kept its output (Φ_l and Φ_r) close to 0.0 all the time, thereby almost nullifying the effect of the high-

level controller. As a result, the robot's walk was similar to the case when only the low-level controller was used, resulting in quite a large lateral deviation along with moderate forward distance. This happened because, compared to the lateral deviation, the forward distance was much larger. At every timestep, the difference of the forward distance and lateral deviation always resulted in a positive reward, and so the agent completely ignored the lateral deviation.

To rectify this issue, the forward distance and lateral deviation were multiplied by weights to control their contributions to the reward value. When the reward was calculated as $reward = \zeta_{dev}(-dev_y^{abs}) + \zeta_{dist}(distance_x)$, with $\zeta_{dev} = 1.0$ and $\zeta_{dist} = 0.5$, the agent was able to change Φ_l and Φ_r according to the state and minimize the deviation. However, yet another kind of unwanted behavior emerged. In the initial seconds of its gait, the robot walked more or less straight. After a few seconds, when it started to deviate, the high-level neural network (agent) would immediately suppress the forward movement completely by setting Φ_l and Φ_r to 1.0. This resulted in the robot facing in a wrong direction and taking only sideways steps after advancing a little distance in the forward direction.

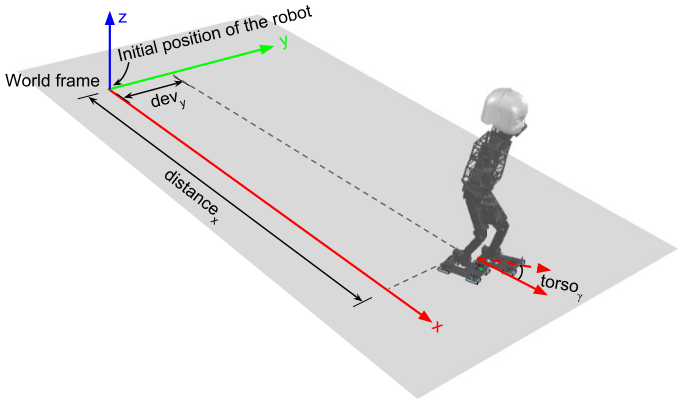


Figure 6.2: Components used for calculating the reward at any particular timestep. The robot is initially positioned at the origin of the world frame, facing in the direction of the x-axis. In the image, the robot has deviated to the left. Symbols are explained in the text.

Finally, to make sure that the robot also faces the correct direction, while at the same time covering the maximum possible forward distance and minimizing the lateral deviation, a negative term was included in the reward for the orientation of the robot's torso ($torso_\gamma$). Thus, the reward at each timestep was calculated as

$$reward = \begin{cases} -100 & \text{if the robot falls} \\ \zeta_{dev}(-dev_y^{abs}) + \zeta_{dist}(distance_x) + \zeta_\gamma(-torso_\gamma^{abs}) & \text{otherwise} \end{cases} \tag{6.1}$$

Here, $torso_\gamma^{abs} = abs(torso_\gamma)$ and $\zeta_{dev}, \zeta_{dist}$ and ζ_γ are weights of the respective terms. The terms for distance and deviation are in meters and the torso angle is in radians. Since $distance_x$ would have the highest contribution, ζ_{dev} and ζ_γ were set to 1.0 and different values for ζ_{dist} were tried out during the training experiments. Figure 6.2 shows the different components used in the reward calculation.

6.1.3 High-Level Controller Training

At each timestep t, the high-level neural network controller needs to produce an action $a_t \in \mathbb{R}^2$, consisting of the *gain suppression factors* Φ_l and Φ_r. As explained in section 4.3.2, both Φ_l and Φ_r are constrained to be in the range $[0.0, 1.0]$. The state $s_t \in \mathbb{R}^{12}$, at each timestep, is the 12-element vector $[\alpha, \beta, \gamma, \dot{\alpha}, \dot{\beta}, \dot{\gamma}, x, y, z, \dot{x}, \dot{y}, \dot{z}]$, as discussed in section 6.1.1. Thus, both the state and action spaces for this problem are continuous and multi-dimensional. The Deep Deterministic Policy Gradient (DDPG) algorithm, discussed in section 2.4.3, was specially designed to handle continuous, multi-dimensional state and action spaces, and hence it was chosen for training the high-level controller.

Critic Network

As DDPG is an actor-critic algorithm, it uses a critic function as an estimator of the action-value function. The *q-values* produced by the critic are used to derive the policy gradient for updating the actor network's parameters. The mathematical equations related to the calculation of the policy gradient can be found in section 2.4.3. In the context of the current problem, the actor is the high-level neural network whose structure has been described in section 4.3.2. Similarly, the critic function is also implemented as a neural network, as shown in figure 6.3.

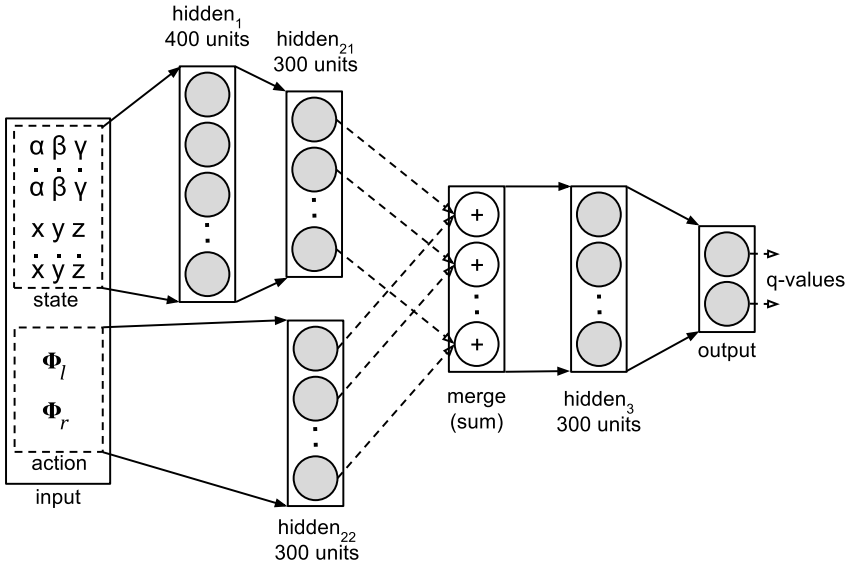


Figure 6.3: Critic neural network. Grey circles represent neurons with non-linear activations. White circles represent the summed output from 2 neurons. Solid arrows represent full connections between adjacent layers. Dashed arrows represent connections with weight=1.

The input to the critic network consists of the state and action retrieved from the replay buffer (see section 2.4.3). The state vector forms the input to the upper part of the network, consisting of two consecutive hidden layers of 400 and 300 units respectively ($hidden_1$ and $hidden_{21}$). The action vector forms the input to a separate hidden layer $hidden_{22}$, made up of 300 units. The outputs of $hidden_{21}$

and $hidden_{22}$ are merged by summing in a pairwise manner and the result is fed into another hidden layer $hidden_3$, consisting of 300 units. The final output layer consists of 2 neurons which produce the *q-values*. The structures of both the actor (high-level controller) and the critic networks were based on the network structures used by Lillicrap et al. [61] for evaluating the DDPG algorithm. As the evaluation in [61] was carried out on a large number of tasks, including classic reinforcement learning problems and continuous control problems, the network structures were considered to be generic enough for the deviation control task in this thesis as well.

Training Process

The DDPG algorithm is listed in section 2.4.3, along with details of the special features of the algorithm. A simplified pictorial representation of the steps (①-⑦) that occur in each iteration, during the training process, is shown in figure 6.4.

In step ①, the actor network receives the current state s_t of the robot's torso and produces an action a_t, consisting of $[\Phi_l, \Phi_r]$ (*gain suppression factors)*, that is to be used to modulate the CPG network. For introducing an exploratory effect during training, the action a_t is modified by adding noise terms generated by an Ornstein-Uhlenbeck process [97]. In this process, a noise term $\mathcal{N}$ for an input Φ is calculated as $\mathcal{N} = \theta(\mu - \Phi) + \sigma \times r$ where θ, μ and σ are parameters of the noise process and r is a random number. The term Φ is then modified as $\Phi = \Phi + \mathcal{N}$.

In step ②, the action a_t (with noise) is performed, by converting Φ_l and Φ_l into the *gain factors* Ψ_l and Ψ_r (using equation 4.4), and then using these values to modulate the CPG network's output. As a result, a new state s_{t+1} and a reward r_t are produced. The scalar reward is calculated according to equation 6.1.

The state transition, represented by the tuple (s_t, a_t, r_t, s_{t+1}), is stored in the replay buffer in step ③. The replay buffer is a first-in-first-out (FIFO) cache which is used to break the correlation between data points in a minibatch while updating the critic network (more details can be found in section 2.4.3).

Next, in step ④, a random minibatch of N transitions is fetched from the replay buffer. Since the selection of transitions present in the minibatch is randomly made, they are not correlated.

The DDPG algorithm makes use of target actor and critic networks for improving the stability of the training process. The target actor and critic networks are created by copying the actor and critic networks respectively. Their outputs are used for producing the target values, which together with the data in the minibatch, is used to calculate the loss value. The parameters of the critic network are then updated by minimizing this loss. These steps are carried out in step ⑤.

In step ⑥, the actor network is updated, by using the sampled policy gradient calculated based on the parameters of the critic and actor networks (equation 2.18).

Although the target networks initially have the same weights as the actor and critic networks, their weights are updated in step ⑦, by having them slowly track the weights of their parent networks using the equation $\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$ where $\tau \ll 1$. Here, θ and θ' represent the parameters of the parent and target networks respectively (e.g, the actor network is the *parent* of the target actor network).

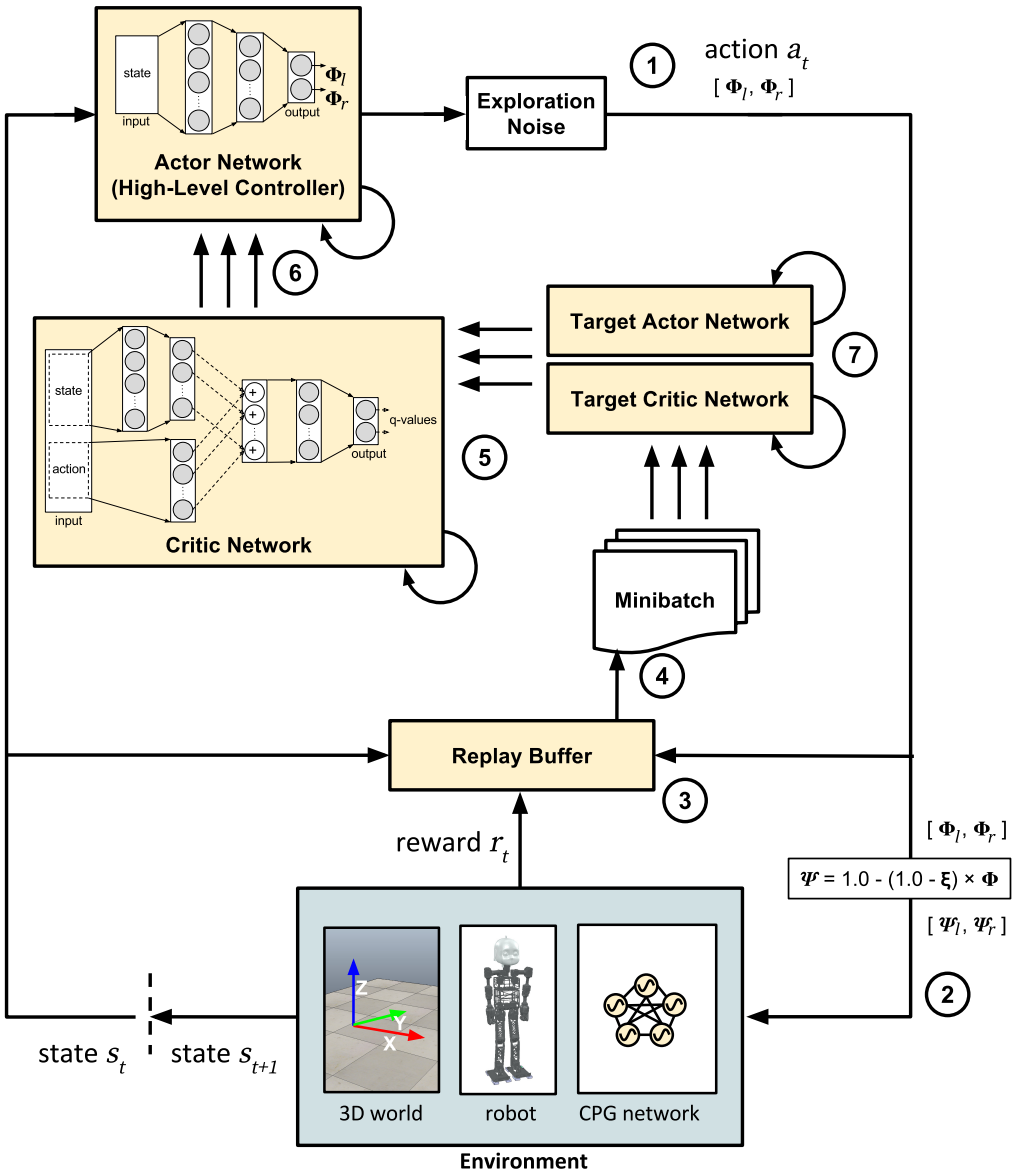


Figure 6.4: Pictorial description of the DDPG training process in the current context. The individual steps are explained in the text. The critic network does not receive the current state and action, but is updated by using the minibatch and the target networks.

6.2 Experiments

For developing the high-level controller, first, the low-level CPG network was constructed by using the *angle feedback* mechanism, because the performance of this feedback was the best (as discussed in chapter 5). The parameters of the CPG network were initialized from the chromosome obtained in run 3 (AF_3) of the *angle feedback* optimization process. The values of the different parameters, that were used to initialize the CPG network, are shown in table 6.1 (reproduced from table 5.4). The internal parameters of all the CPGs were set according to table 4.3. The

initialization of connections between the oscillators was done in the same way as described in chapter 5 (refer to figure 4.2). The reinforcement learning experiments were conducted on machine $M1$ (refer to section 5.1.1 for descriptions of machines).

Table 6.1: Chromosome AF_3, which is used for initializing the CPG network.

k_f	GAIN1	GAIN2	GAIN3	GAIN4	GAIN5	GAIN6	BIAS1	BIAS2	BIAS3	BIAS4	k
0.3178	0.3777	0.0234	0.0132	0.4567	0.2019	0.3309	-0.0519	0.0963	-0.1156	0.4814	1.5364

6.2.1 DDPG Hyperparameters

The hyperparameters of the DDPG setup, used for training the high-level controller, were almost identical to those used in [61]. An Adam optimizer [55] was used for learning the parameters of both the actor and critic networks. The learning rates were 10^{-4} and 10^{-3} for the actor and critic respectively. For the critic network, L_2 regularization was used with a weight decay of 10^{-2}. The ReLu activation function was used in all the hidden layer neurons. The neurons in the final output layer of the actor network had sigmoid activations to bound the output between 0.0 and 1.0. The neurons in the final layer of the critic had linear activations. The final layer weights and biases of the actor and critic networks were initialized from a uniform distribution $[-3 \times 10^{-3}, 3 \times 10^{-3}]$, so that the initial outputs of both the networks would be close to zero. The weights and biases of all the other layers were initialized from the uniform distribution $[-\frac{1}{\sqrt{f}}, \frac{1}{\sqrt{f}}]$, where f was the fan-in of the layer. A minibatch size of 64 was used for the training.

For calculating the loss, which was used for updating the parameters of the critic network, the expected long-term reward needed to be calculated (equation 2.17). In this calculation, a discount factor of $\gamma = 0.99$ was used. The size of the replay buffer was set to a maximum of 10^5 entries. For updating the target networks, a rate of $\tau = 10^{-3}$ was used. The parameters of the Ornstein-Uhlenbeck process [97] were set as $\mu = 0.0$, $\theta = 0.15$ and $\sigma = 0.2$.

6.2.2 Setup

Apart from the hyperparameters which are specific to the DDPG algorithm, a few extra hyperparameters have been used in the setup described in figure 6.4. The first three hyperparameters, ζ_{dev}, ζ_{dist} and ζ_{γ}, are used in the calculation of the reward signal (equation 6.1). The fourth hyperparameter is the *lowest gain factor* ξ, which controls the amount of influence the high-level controller has on the low-level controller. As discussed in section 4.3.2, when $\xi \approx 1.0$, the high-level controller has minimal influence and when $\xi \approx 0.0$, the high-level controller has maximal influence on the low-level controller. To train the high-level neural network, different sets of values of these hyperparameters were tried out. This resulted in four different trained models from four different setups, $Setup_1 - Setup_4$. The values of the hyperparameters for the different setups are shown in table 6.2.

The values of these hyperparameters were decided empirically. The performance of the different trained models was then objectively compared in a separate testing phase. In addition to this, a control setup, $Setup_{control}$, consisting of only the low-level CPG network without any high-level controller, was also tested to see the difference that the high-level controller makes to the robot's performance.

Table 6.2: Reinforcement learning setups with different hyperparameters.

Setup	ζ_{dev}	ζ_{dist}	ζ_{γ}	ξ
$Setup_1$	1.0	0.5	1.0	0.1
$Setup_2$	1.0	0.5	1.0	0.4
$Setup_3$	1.0	0.3	1.0	0.1
$Setup_4$	1.0	0.3	1.0	0.4
$Setup_{control}$	No high-level control			

6.2.3 Training Results

For each of the setups with high-level control ($Setup_1 - Setup_4$), the high-level neural network was trained for 1000 episodes using the DDPG algorithm, as described in figure 6.4. At the start of each episode, the robot was placed at the origin of the world frame, facing the direction of the world's x-axis. The start of the walk was done in the same way as in the experiments in chapter 5. Each episode lasted for 40 seconds. If the robot fell down before this time limit, the episode was terminated. Reward calculation and weight updates were done once every second to allow the actor's output to take effect. In these training experiments, a higher time limit of 40s was used (compared to 20s in chapter 5), to allow the high-level controller enough time to change the walking direction.

During the training process, the progress of each trained model was evaluated periodically. This was done by using the high-level neural network (actor) to control the deviation for one episode after every 10^{th} training episode. During this test episode, the exploratory noise was switched off and parameter updates were not made. At the end of the episode, the reward was calculated, along with the individual components which were used to calculate the reward (forward distance, lateral deviation and torso-γ angle).

Figure 6.5 shows the training progress for $Setup_1$. The reward plot shows a large number of troughs (large negative values) in the initial episodes. Since a reward of -100 was given when the robot fell down, the reward plot indicates that initially, the system was quite unstable. However, in later episodes, it can be seen that the high-level controller learned to be more stable, since large negative rewards do not occur in the later episodes. The plot for distance and deviation shows that in the initial episodes (till about episode 450), there was a lot of variability, but most of the time the robot could reach a distance of around 4m. However, from episode 450 to 750 (approximately), both the distance and deviation graphs show very less variability.

The distance during these episodes is quite low (below 2m) and so the deviation is also low. After around the 750$^{\text{th}}$ episode, the model's behavior changed again, and the robot's forward motion was less constrained. The distance improved while the deviation stayed around the 1m mark. The plot for torso orientation shows more or less stable behavior from around episode 500 onwards.

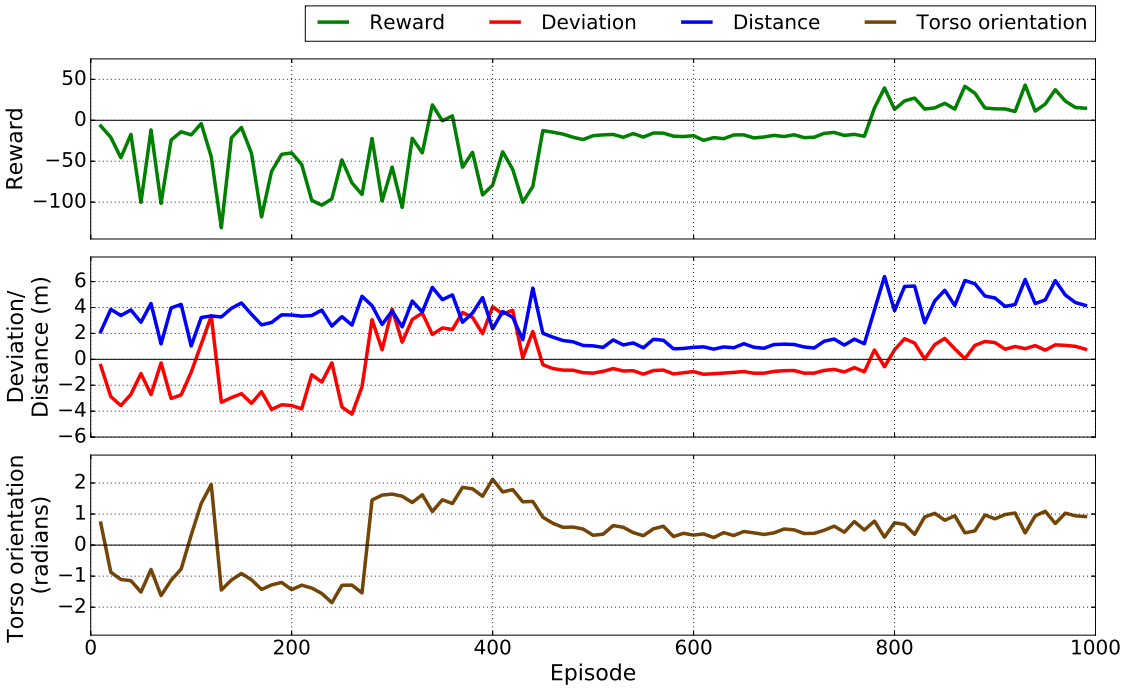


Figure 6.5: Training progress for $\mathbf{Setup_1}$ in 1000 episodes. (Top) Reward, (middle) distance and deviation, (bottom) torso orientation angle about the world z-axis.

The training progress of the high-level controller for $Setup_2$ is shown in figure 6.6. The plot for reward shows that the model for $Setup_2$ was more stable than the model for $Setup_1$ since rewards lesser or equal to -100 occur less frequently. Unlike the previous setup, there is no long period where the distance covered shows extremely low values. In all the three plots (reward, distance-deviation, and torso orientation), the performance shows variability. However, compared to the distance and deviation achieved by the trained model in the initial episodes, in the final few episodes, the performance is better, because the distance stays between the 4m and 6m mark and the deviation stays between 2m and -2m. The deviation, relative to the forward distance, is still quite significant, even near the 1000$^{\text{th}}$ episode.

The training progress of the model using $Setup_3$ is demonstrated in figure 6.7. This model shows much more promising results with respect to all the four parameters that were measured during training (reward, distance, deviation and torso orientation). The plot for reward shows a stable behavior from episode 600 onwards. Every time the trained model was tested after episode 600, the distance achieved was around 6m while the deviation stayed close to 0. The torso orientation also remained near the ideal position of 0 radians, from episode 600 till the end.

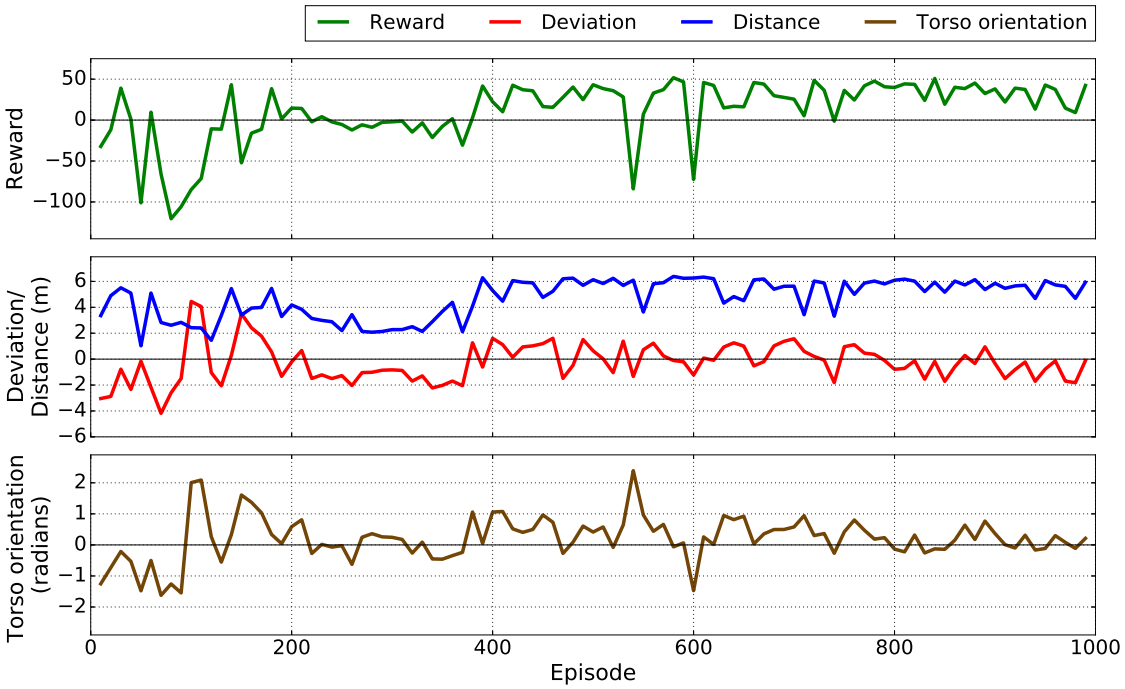


Figure 6.6: Training progress for $\mathbf{Setup_2}$ in 1000 episodes. (Top) Reward, (middle) distance and deviation, (bottom) torso orientation angle about the world z-axis.

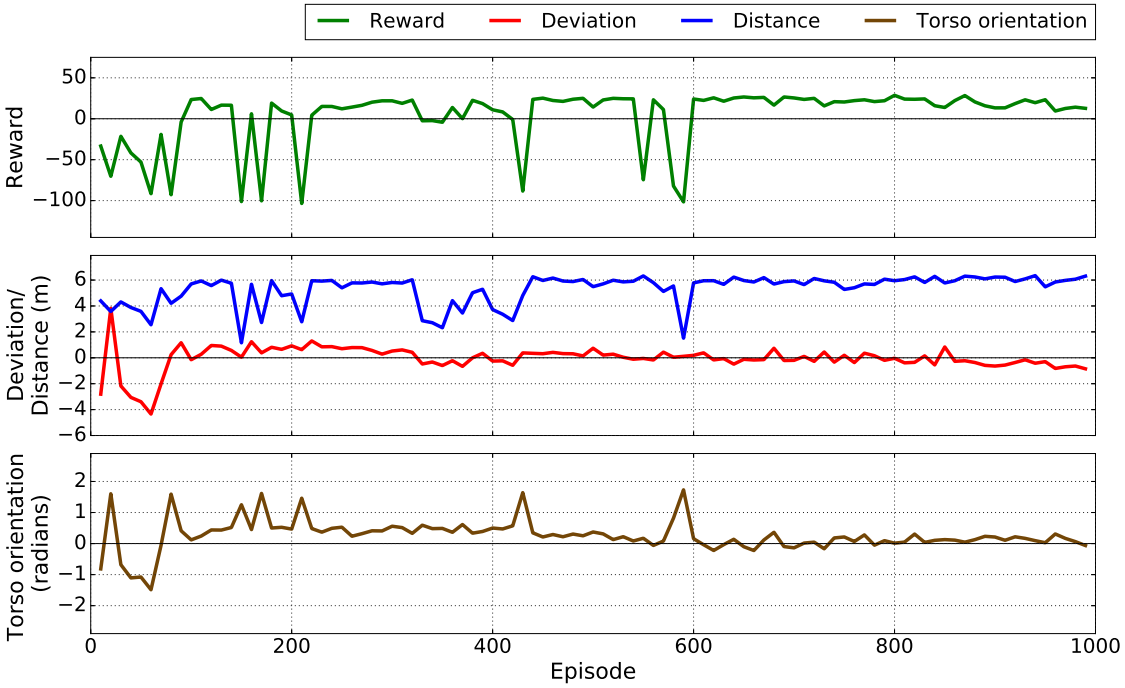


Figure 6.7: Training progress for $\mathbf{Setup_3}$ in 1000 episodes. (Top) Reward, (middle) distance and deviation, (bottom) torso orientation angle about the world z-axis.

Figure 6.8 shows the performance of the model using $Setup_4$ during training. Towards the 1000^{th} episode, this model was also able to achieve forward distances of around 6m. However, there was a lot of variability in the model's performance, as can be seen from the plots. Even in the final 200 episodes, the robot fell down twice, as can be inferred from the large negative rewards. The models of none of the previous setups show this kind of instability towards the end of the training. The plot for deviation varies around 0 but does not settle down. The torso orientation also shows significant variations, though it does reach near 0 in quite a few episodes.

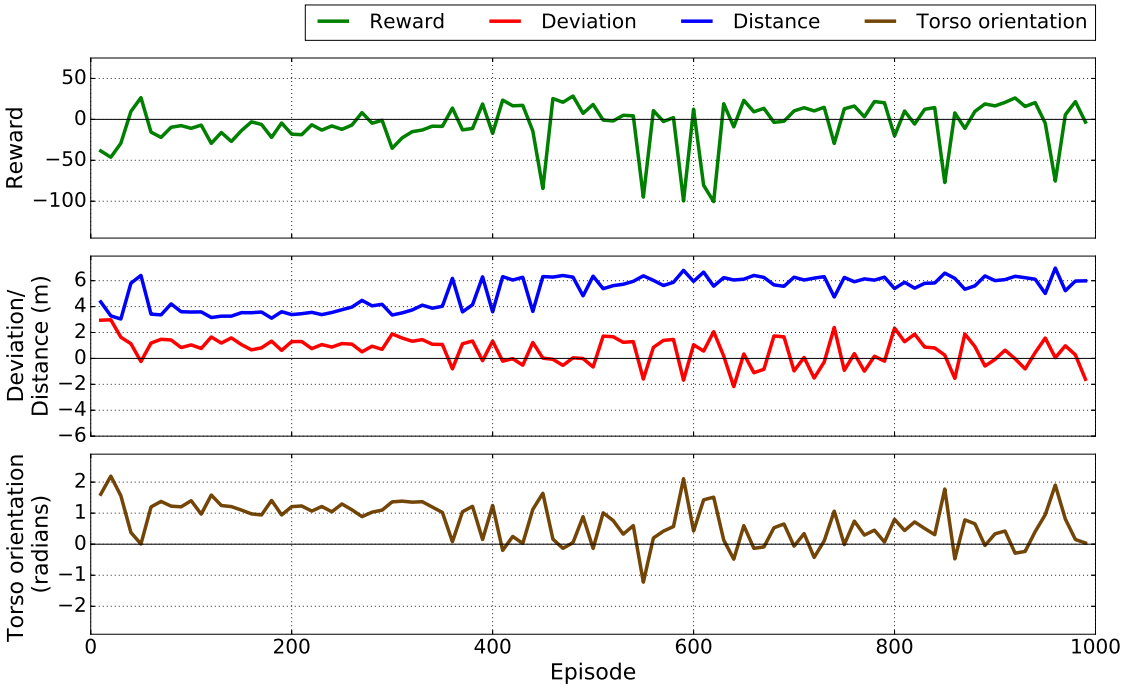


Figure 6.8: Training progress for $\mathbf{Setup_4}$ in 1000 episodes. (Top) Reward, (middle) distance and deviation, (bottom) torso orientation angle about the world z-axis.

6.2.4 Test Results

Once the training was complete for the four setups, the trained high-level neural network for each setup was tested for 100 episodes, each of 40s duration. The distance, deviation and torso-γ orientation were measured at the end of each episode. The control setup was also put through the same test. During the test phase, the high-level network's behavior was completely deterministic, since the exploratory noise was only used during the training phase.

The performance of the four trained models and the control setup are shown using boxplots in figures 6.9-6.11. The trained models are depicted with blue boxes and the control setup is shown in red. The other symbols in the plots have the same meanings as the boxplots used in section 5.2.2. The median values and interquartile range (IQR) for each measurement of each setup are shown in table 6.3.

Instead of the mean and standard deviation, the median and IQR were used to analyze the performance of the models because these measures are resistant to outliers.

Table 6.3: Median and inter-quartile Range (IQR) values for distance (m), deviation (m) and torso orientation (rad) for the four trained models and the control setup in 100 test episodes.

	Median			IQR		
Setup	Distance	Deviation	Torso orientation	Distance	Deviation	Torso orientation
$Setup_{control}$	4.313	3.560	1.605	1.117	0.694	0.318
$Setup_1$	4.689	1.022	0.934	1.354	0.438	0.148
$Setup_2$	4.252	1.878	1.326	0.717	0.258	0.180
$Setup_3$	6.044	-0.774	0.011	0.331	0.586	0.183
$Setup_4$	6.071	-0.045	0.225	0.614	1.484	0.607

It can be seen from figure 6.9, that the distance walked by the robot, when the trained models for $Setup_3$ and $Setup_4$ were used, is much higher than all the other setups. For both these setups, the median distance covered in 40s for 100 test episodes is above 6m. Although the median distance for $Setup_4$ (6.071m) is slightly higher than the distance for $Setup_3$ (6.044m), $Setup_4$ shows more variability in the results, as can be judged from its IQR value. This variability in $Setup_4$'s performance was also visible during the training process (figure 6.8). The performance of the trained models of the other setups is slightly better than the control setup. For $Setup_1$, the median distance is higher than $Setup_{control}$, but the variability of the results is higher. The median distance for $Setup_2$ is slightly less than that for $Setup_{control}$, but the IQR is lower.

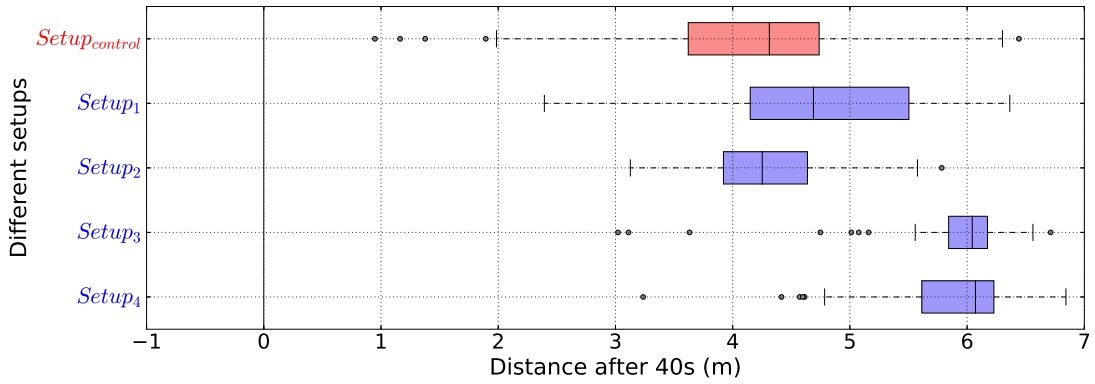


Figure 6.9: Results for **distance** in 100 test episodes for each setup.

Figure 6.10 shows the results for deviation. Since the primary task of the high-level controller is to minimize the deviation of the robot, the effect of the high-level controller can be judged from this figure. The deviation for the trained models of all the setups is significantly lower than the deviation for $Setup_{control}$. Without any form of high-level control, the robot tends to turn strongly towards its left. This can be seen from the median deviation of 3.56m for $Setup_{control}$ (table 6.3). Thus,

at the end of 40s, the robot walking without high-level control deviates almost as much as it walks forward. When high-level control is used, this deviation is significantly lowered and consequently, the forward distance also improves. The best performance is achieved by $Setup_3$ and $Setup_4$, for which the median deviations are -0.774m and -0.045m respectively. The negative median deviations indicate that for these two setups, the high-level controller tries to compensate for the tendency of the robot to turn left, by pushing it towards the right. The median deviation for $Setup_4$ is very close to 0, but as with the distance, the variability in deviation is also high, with an IQR of 1.484m. $Setup_1$ and $Setup_2$'s performance is much better than $Setup_{control}$ but the deviation control is not as strong as for $Setup_3$ and $Setup_4$.

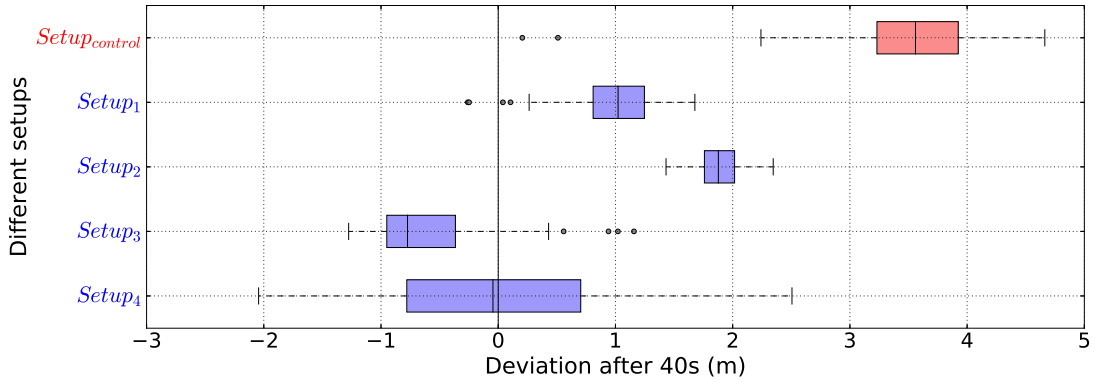


Figure 6.10: Results for **deviation** in 100 test episodes for each setup.

The orientation of the robot's torso at the end of the walk (about the world z-axis) indicates how much the robot has turned. The robot starts its walk in an orientation where its x-axis coincides with the x-axis of the world frame, and hence the orientation is 0 radians. For a perfectly straight walk, the robot should still be facing in exactly the same direction at each step. Hence, in the ideal case, the final orientation should also be 0 radians.

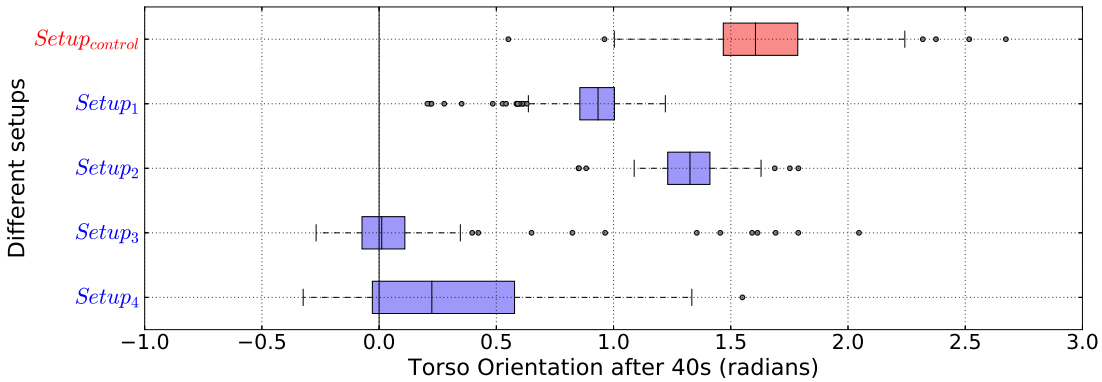


Figure 6.11: Results for **torso orientation** in 100 test episodes for each setup.

Figure 6.11 shows the torso orientations achieved by the different setups. Here too, the effect of high-level control is evident. The median orientations for the trained models of all the setups are better than that for $Setup_{control}$. Among the different setups, $Setup_3$'s median orientation is nearly 0 (0.011 radians) while the variability is also not very high (IQR=0.183 radians). The next best performance is by $Setup_4$ with a median orientation of 0.225 radians. However, the IQR is very high (0.607 radians). Similar to the previous results, for torso orientation, the performance of $Setup_1$ and $Setup_2$ is also inferior to that of $Setup_3$ and $Setup_4$.

Figure 6.12 shows the robot's path after 40s, for the different setups (3 examples for each setup). The best deviation control is achieved by $Setup_3$ and $Setup_4$.

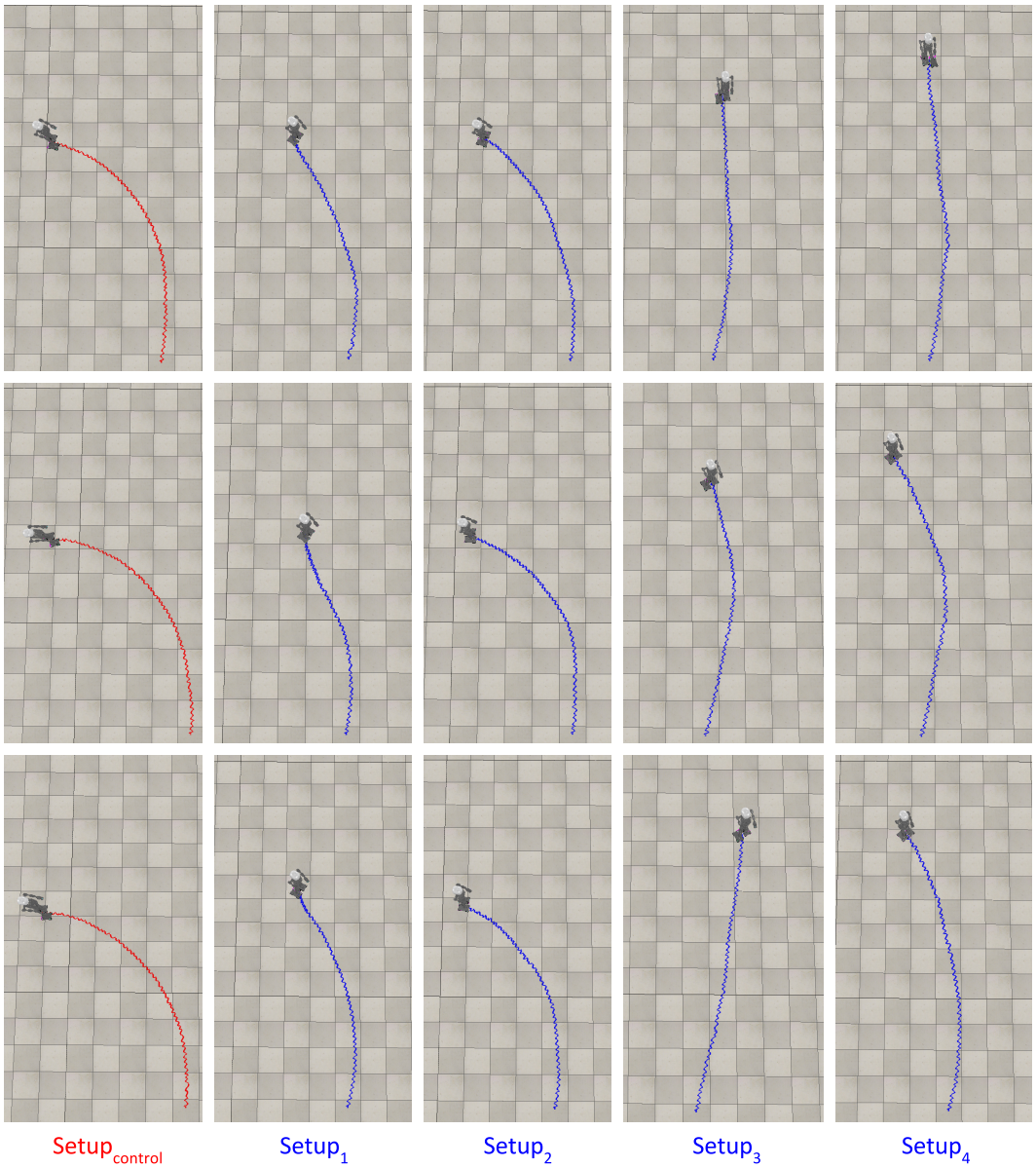


Figure 6.12: Top-down view of the robot's path. Each column contains images of three examples for a particular setup.

6.3 Discussion

The results presented in section 6.2.4 show that the models for $Setup_3$ and $Setup_4$ performed the best in the evaluation tests. These models consistently achieved the best scores for all the parameters which were measured (distance, deviation and torso orientation). The difference with the results of the other setups is also visible in the boxplots. From figure 6.12, it can be seen that the paths taken by the robot for $Setup_3$ and $Setup_4$ (images in the two right-most columns) were much straighter than the other setups. When the median values of the parameters are considered, the performances of $Setup_3$ and $Setup_4$ are quite close. However, in all the tests, $Setup_4$ showed much greater variability in the results. This variability was also present during the training process, as shown in figure 6.8. Hence, overall the best high-level control was achieved by the model for $Setup_3$.

Among the different setups, there is a clear distinction between the performance of ($Setup_1$, $Setup_2$) and ($Setup_3$, $Setup_4$). For all the parameters, the results for $Setup_1$ and $Setup_2$ are inferior to that of $Setup_3$ and $Setup_4$. The main difference between these sets of setups is the value of the hyperparameter ζ_{dist} (weight of the distance term in the reward calculation in equation 6.1). For $Setup_1$ and $Setup_2$ ζ_{dist} was set to 0.5, while for $Setup_3$ and $Setup_4$, ζ_{dist} was set to 0.3. The values of ζ_{dev} and ζ_{γ} were set to 1.0 for all the setups. A higher value of ζ_{dist}=0.5 in the reward calculation during training gave more importance to the distance term. Thus, to a certain extent, the negative contributions of the deviation and orientation terms were overridden by the distance term. When a lower value of ζ_{dist}=0.3 was used for $Setup_3$ and $Setup_4$, the negative effects of deviation and orientation were more prominent in the reward calculation and the high-level controller learned to give more importance to these terms while deciding its actions.

The difference between the two best setups was the value of the *lowest gain factor* ξ. For $Setup_3$, ξ was set to 0.1 and for $Setup_4$, it was set to 0.4. Thus for $Setup_3$, the high-level controller could exert more influence in selectively suppressing the forward motion of a leg. This factor, combined with a nice balance between the different terms in the reward calculation, possibly led to the high-level controller for $Setup_3$ producing good results with less variability.

However, even for $Setup_3$, the high-level controller could not produce completely straight paths. The mechanism for controlling the deviation by selectively suppressing the forward motion of a leg is possibly not adequate for achieving perfect control. This mechanism may be improved by letting the high-level controller also affect additional joints such as those responsible for the sideways swaying motion of the robot. The promising aspect of the results presented in this chapter is that, even with a limited form of high-level control, the neural network could be trained to perform fairly well at minimizing the lateral deviation. Most importantly, these results indicate that it is possible to achieve stable bipedal locomotion by using a high-level neural network controller to improve the performance of a low-level CPG network. Using the same approach, more complex forms of high-level control, such as maintaining balance, can also be attempted in the future.

Chapter 7

Conclusion

This thesis presented a biologically-inspired, hierarchical controller for bipedal locomotion. Using this controller, stable bipedal locomotion was achieved for the NICO robot in a simulated environment. This concluding chapter begins with a brief summary of the individual steps that were taken to develop the walking controller. An overview, of the results that were obtained, is presented in the context of the research questions proposed in the beginning of the thesis. The contributions of this work are also discussed. Finally, some suggestions are presented for improving the proposed controller and for implementing it on a physical robot.

7.1 Summary

The proposed bipedal locomotion controller consists of two parts arranged in a hierarchy. At the lower level, a network of central pattern generators (CPGs) creates the rhythmic patterns that are required to control the individual joints for walking. This CPG network was originally proposed by Cristiano et al. [11]. Different feedback mechanisms for modulating the output of the CPG network have been tested and the resulting robot gaits have been analyzed. In addition to the *phase reset* feedback mechanism proposed in [11], another kind of feedback called *angle feedback* was also tested. The parameters of the CPG network were optimized using a genetic algorithm.

After testing the CPG network with different feedback mechanisms, the feedback type, with the best relative performance, was chosen and a high-level neural network controller was added on top of it. The neural network's role was to monitor the position and orientation of the robot's torso and to adjust certain parameters in the CPG network so that the robot walks in a straight line. The high-level controller was trained using reinforcement learning. Different sets of values for its hyperparameters were tested to find the best combination.

The primary research question (section 1.2) that this thesis aimed to answer was whether stable bipedal locomotion could be achieved by using a hierarchy of controllers, where a CPG network would directly control the robot's joints and a neural network would modulate the CPG network's output. The results presented

in this thesis show that this is possible. In chapter 6 it was shown that the high-level controller was able to modulate the CPG network's output, and was able to minimize the lateral deviation during walking. The robot was also able to walk a longer distance. To achieve the desired objective, the design of the reward function in the reinforcement learning setup was crucial, as it provided the means of objectively judging the robot's walk. With a properly designed reward function and carefully selected hyperparameters, it was possible to train the neural network to achieve its stated objective of minimizing the lateral deviation. However, for the high-level controller to function properly, the role of the CPG network, augmented by a proper feedback mechanism, was also equally important.

The secondary research question (section 1.2), investigated in this thesis, was related to the effectiveness of the low-level CPG feedback mechanisms. The analysis of different feedback mechanisms in chapter 5 revealed that the *angle feedback* mechanism definitely enabled the CPG network to improve the robot's gait. On machine $M1$, the performance of the *phase reset* mechanism was marginally better than the *open loop* configuration (with no feedback pathways) and the performance of the *angle feedback* mechanism was found to be the best.

7.2 Contributions

The concept of using central pattern generators to achieve bipedal locomotion in robots is not new. In the past, many studies [11, 12, 79] have used CPGs, such as the Matsuoka oscillator or the Hopf oscillator, to generate the rhythmic patterns for walking. However, most of the existing approaches for CPG-based bipedal locomotion concentrate on the mechanism and behavior of the CPGs only. The main contribution of this thesis is the development of a controller for bipedal locomotion by combining existing knowledge about CPGs with a novel high-level mechanism for further improving the robot's gait. A recent advancement in reinforcement learning - the Deep Deterministic Policy Gradient (DDPG) algorithm - has been used to train the high-level neural network controller.

The usage of a neural network as a high-level controller makes it relatively easy to extend the functionality of the controller to achieve other objectives. This can be done by modifying the design of the neural network in order to use different inputs and outputs and then training it by using a reward function designed to suit the new objective. The hierarchical, modular structure of the walking controller also makes it possible to try out other types, or configurations of CPGs.

Although the NICO robot has been used in this thesis, the developed controller is independent of any particular robot model, since it does not rely on information about the kinematics or dynamics of a particular robot. Hence, conceptually, the same controller can be trained to control and optimize the gait of other position-controlled humanoid robots with a similar morphological structure. However, for both the optimization of the low-level CPG network and the training of the high-level neural network controller, a large number of walking trials need to be performed. This is only possible if a simulated model of the robot is available.

Another contribution of this thesis is the analysis of the effects of different low-level feedback mechanisms that are used with the CPG network. Feedback mechanisms enable the CPGs to adapt their rhythmic patterns to the natural dynamics of the robot. However, in the literature, most CPG-based methods use only a single type of feedback. In this thesis, the same CPG network has been tested with two kinds of feedback and without any feedback, and the resulting gaits have been compared based on measurable parameters.

7.3 Future Work

The proposed hierarchical controller has been able to produce a stable gait in simulation and has also been able to control the lateral deviation in spite of the presence of systematic errors and slippage which make the robot turn. However, the effectiveness of this controller can be further improved.

Currently, the mechanism for making the robot change its direction relies on selectively suppressing the forward motion of the legs, by altering the gains of the sagittal hip joints. While this strategy does result in the robot turning in the desired direction, the effect is not instantaneous. A few seconds are needed for the turning to occur. This mechanism may be improved by also utilizing the joints which produce movement in the frontal or transverse[1] planes. However, this would increase the complexity of the learning process and may lead to lowering the robot's stability in the initial phase of training. Hence, a lot more episodes may be necessary for teaching the high-level controller to behave in the desired manner.

The set of gains and biases, that constitute the parameters of the low-level CPG network, have been determined through an optimization process. Similarly, the weights associated with the *angle feedback* pathways have also been determined through optimization. Instead of using fixed values of these feedback weights, it would be interesting to employ a neural network to alter these weights based on sensory information. The *phase reset* mechanism is also static in the sense that the phase of the pacemaker oscillator is reset whenever the robot's right foot strikes the ground. Here too, there is a scope of using a trainable neural network which would determine whether resetting the phase is necessary based on how synchronized the robot's movements are with the rhythms of the CPG network. The feedback mechanisms' purpose is to adapt the CPG network's oscillatory output to the dynamics of the robot. In situations where the robot walks on undulating surfaces or transitions from a flat surface to a sloping one, a trainable element within the CPG feedback mechanism would encourage better adaptation. Instead of using *angle feedback* and *phase reset* separately, the two feedback mechanisms may also be combined together in the same CPG network to test whether multiple types of feedback working together provide any additional benefits.

In this thesis, the high-level controller has only been used for controlling the lateral deviation of the robot. However, using the same general framework, more

[1]The transverse or axial plane divides the body into the head and tail portions.

diverse forms of high-level control can be achieved. The scope of this work was limited to walking on flat surfaces and so balance control was not handled explicitly. If the controller is to be trained to control walking on sloped surfaces, then it needs to be given a way of altering the tilt of the robot's body to adjust to a slope. At present, the neural network's outputs are used to modify the gains of the sagittal hip joints. Additional outputs from the same neural network can be used to change the bias positions of the hip joints. This would enable the robot to tilt its body forward or backward and thereby adjust its posture to prevent a fall. Of course, the training scenario would also need to be modified. The training experiments would need to be conducted on different kinds of sloped surfaces. For deciding how much and in which direction to tilt the body, information from the foot sensors can be used to track the position of the ZMP. This approach has been suggested in [74], where the balancing of a robot with a flexible vertebral column was controlled in a standing position.

The next logical step, after the simulation experiments which have been carried out in this thesis, would be to implement the hierarchical walking controller on the physical NICO robot. Although the general control framework can be transferred as-is to the real robot, some modifications related to the acquiring of sensory information will be necessary. The *phase reset* feedback mechanism, used in the low-level CPG, uses the force sensors for detecting ground contact. Once the physical NICO is equipped with force sensors, this information will be readily available. The current joint angles, needed by the *angle feedback* mechanism, can be obtained from the encoders available in the robot's joints. The high-level controller needs information about the state of the robot's torso. This can be obtained by utilizing sensors such as the IMU (Inertial Measurement Unit) or the robot's camera. For handling noise inherent in sensory readings in a physical environment, filtering mechanisms such as the particle filter can be used. In addition to these modifications, a more accurate simulation model is also desirable, since this will help to make the CPG network parameters, optimized in simulation, more suitable for the physical robot as well. The same approach can also be tested on other humanoid robots.

Considering the complex interplay between the robot's own movements and its interaction with the environment, achieving stable and robust bipedal locomotion is a difficult task. Given the limitations in robotic hardware and our incomplete understanding of the exact mechanisms that enable us to walk with so much efficiency, perfectly emulating the mechanism in animals and humans is difficult. However, even with the limited resources at our disposal, drawing cues from nature can help us to design artificially intelligent systems that can inherit some of the desirable characteristics of their biological counterparts. Biologically-inspired techniques, such as the one proposed in this thesis, can be improved upon by utilizing the latest developments in the field of artificial intelligence. It can be hoped, that in the days to come, such techniques will enable humanoid robots to walk with the same graceful agility that is a characteristic of our own walking movements.

Appendix A

Low-level Controller Results (M2)

The genetic algorithm and gait evaluation results from machine $M2$ are presented here. As explained in chapter 5, these results are given less emphasis due to the lower configuration of $M2$. For all the experiments, apart from the difference of the configurations of $M1$ and $M2$, all other aspects of the setup were the same. The axis ranges in all the plots have been kept similar to the plots in chapter 5.

A.1 Genetic Algorithm Results

The results of the genetic algorithm experiments are presented in figures A.1-A.3. Compared to the results from $M1$ (figures 5.4-5.6), all the evolved chromosomes in $M2$ show a lower fitness and a much lower distance. However, similar to the results on $M1$, *angle feedback* achieved the highest forward distance on $M2$ also.

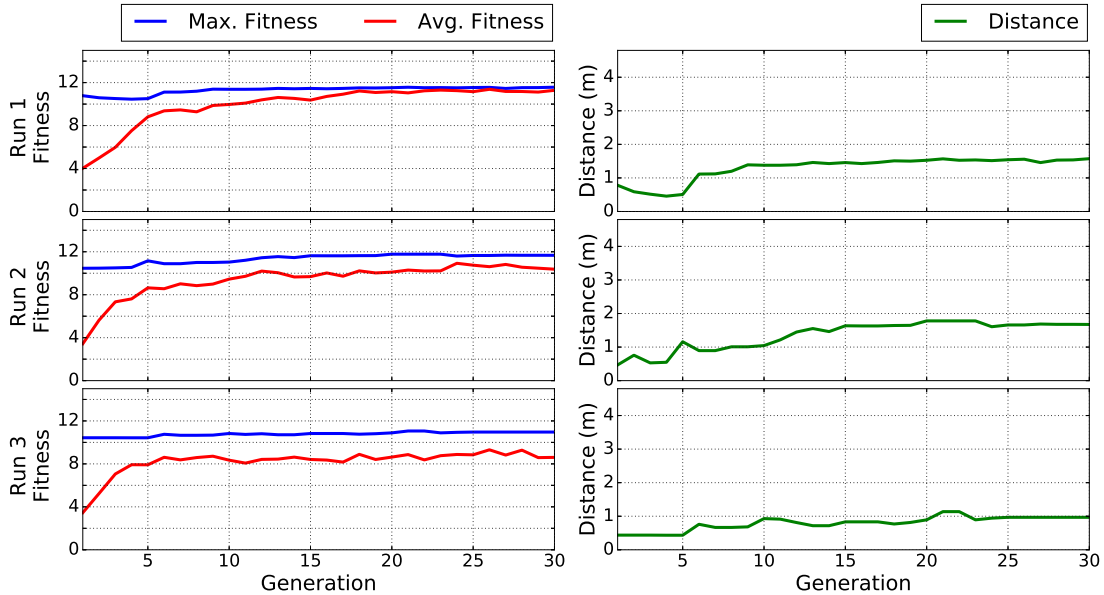


Figure A.1: **Open loop** results ($M2$). Left column: Fitness vs. generation, right column: distance vs. generation. Top to bottom: Three separate runs of the genetic algorithm.

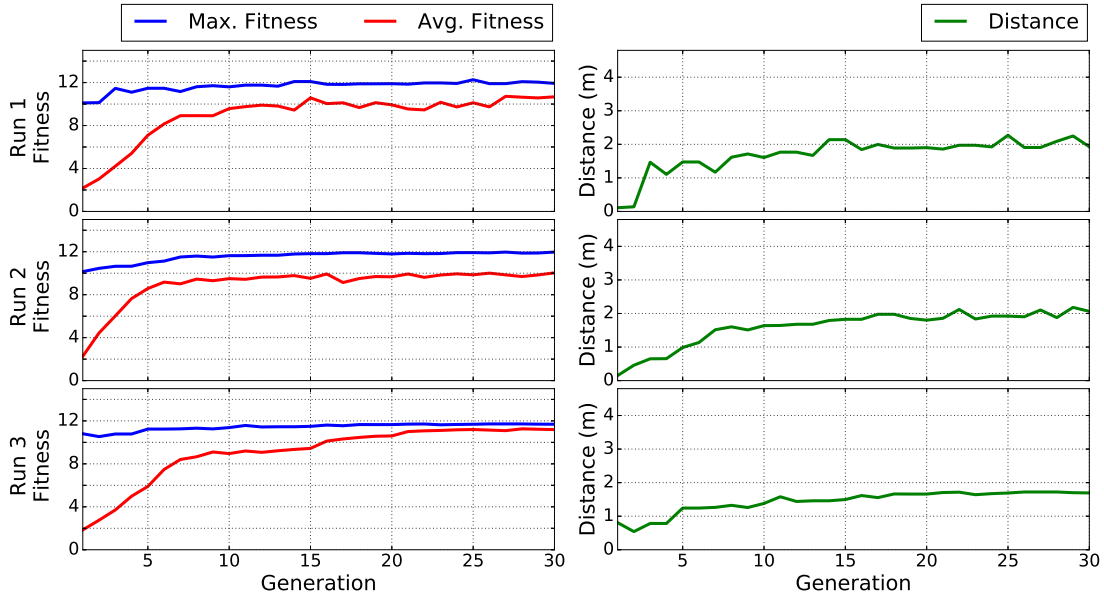


Figure A.2: **Angle feedback** results ($M2$). Left column: Fitness vs. generation, right column: distance vs. generation. Top to bottom: Three separate runs of the genetic algorithm.

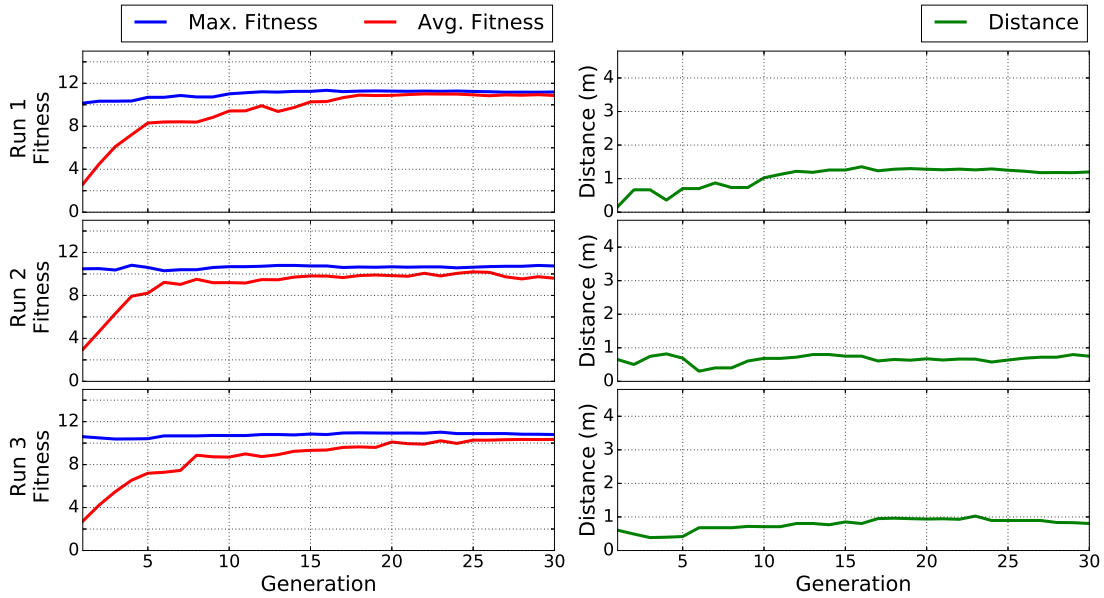


Figure A.3: **Phase reset** results ($M2$). Left column: Fitness vs. generation, right column: distance vs. generation. Top to bottom: Three separate runs of the genetic algorithm.

Table A.1: Best performance in the 30^{th} generation on $M2$.

	Open loop			Angle feedback			Phase reset		
	run 1	run 2	run 3	run 1	run 2	run 3	run 1	run 2	run 3
Best distance (m)	1.57	1.67	0.97	1.93	2.06	1.69	1.2	0.75	0.81
Best time (s)	20	20	20	20	20	20	20	20	20
Best fitness	11.57	11.67	10.97	11.93	12.06	11.69	11.2	10.75	10.81

The best results achieved in the last generation for the different setups on machine $M2$ are listed in table A.1. The distances achieved, when *angle feedback* was used, was higher than that for all the other setups. However, the overall performance of *phase reset* was worse than for *open loop*. The chromosomes which evolved on machine $M2$ (for all the setups) resulted in gaits which were slower (lower frequency of leg movements) than the corresponding gaits which evolved on machine $M1$. Hence the distance walked by the robot was also much lower. The lower configuration and simulation frame rate of machine $M2$ seemed to favor the evolution of slower gaits. The 9 chromosomes (for runs 1, 2 and 3 for the 3 setups) found by the genetic algorithm on $M2$ are shown in table A.2.

Table A.2: CPG network parameters found by the genetic algorithm (genotype of the chromosomes) on $M2$; BIASi is in radians; k_f, GAINi and k do not have any units.

	Open loop			Angle feedback			Phase reset		
	run 1	run 2	run 3	run 1	run 2	run 3	run 1	run 2	run 3
k_f	0.5884	0.5486	0.5429	0.7462	0.6450	0.6642	0.5146	0.3780	0.4563
GAIN1	0.5370	0.7970	0.9371	0.8423	0.7693	0.5257	0.7455	0.8206	0.8787
GAIN2	0.0913	0.0389	0.0442	0.0481	0.0193	0.0831	0.0257	0.1437	0.0195
GAIN3	0.0253	0.0588	0.3404	0.1321	0.0941	0.0112	0.1112	0.0840	0.2469
GAIN4	0.3494	0.4611	0.3787	0.6911	0.4993	0.5196	0.4533	0.3680	0.3530
GAIN5	0.1500	0.3213	0.2000	0.5980	0.3632	0.7228	0.3676	0.7905	0.6167
GAIN6	0.9377	0.3781	0.2638	0.2017	0.4918	0.8231	0.2105	0.4384	0.5989
BIAS1	-0.3100	-0.3094	-0.5352	-0.1162	-0.3334	-0.0575	-0.3906	-0.3562	-0.1184
BIAS2	0.2827	0.2508	0.4412	0.4894	0.4633	0.2024	0.4887	0.3578	0.4879
BIAS3	-0.0826	-0.0788	-0.0820	-0.3117	-0.2125	-0.1452	-0.2133	-0.1089	-0.2604
BIAS4	0.2218	0.2368	0.3757	0.5584	0.0182	0.3216	0.1151	0.2433	0.0676
k	NA	NA	NA	-0.3420	-0.1783	-0.7916	NA	NA	NA

A.2 Gait Analysis Results

The 9 chromosomes which were found by the genetic algorithm were named OL_1, OL_2 and OL_3 (for runs 1, 2 and 3 of *open loop*), AF_1, AF_2 and AF_3 (for runs 1, 2 and 3 of *angle feedback*), and PR_1, PR_2 and PR_3 (for runs 1, 2 and 3 of *phase reset*). Each of these chromosomes was tested 100 times by constructing the CPG network using the values in the chromosomes and letting the robot walk for 20s each time. During the robot's walk, certain parameters were measured. The setup and procedure of evaluating the different gaits were the same as on machine $M1$. A detailed description of the experimental setup and the parameters used for evaluation can be found in section 5.2.1. The eight different evaluation parameters are described in table 5.5. Median values of the gait evaluation parameters on machine $M2$ are listed in table A.3. The values of the different parameters are plotted using boxplots in figures A.4-A.9. The symbols in the plots have the same meanings as described in section 5.2.2.

Table A.3: Median values of the gait evaluation parameters for 100 trials on $M2$.

Parameter	Open loop			Angle feedback			Phase reset		
	OL_1	OL_2	OL_3	AF_1	AF_2	AF_3	PR_1	PR_2	PR_3
f	11.363	11.401	5.894	11.56	11.287	11.386	11.288	5.971	10.929
t_{up} (s)	20.00	20.00	11.845	20.00	20.00	20.00	20.00	12.275	20.00
$distance_x$ (m)	1.368	1.405	-0.029	1.636	1.372	1.391	1.294	-0.164	0.934
dev_y^{abs} (m)	0.128	0.148	0.179	0.223	0.276	0.175	0.229	0.183	0.152
$stride^{avg}$ (m)	0.051	0.053	0.031	0.09	0.083	0.071	0.053	0.02	0.034
$torso_\alpha^{var}$ (rad^2)	0.003	0.003	0.005	0.008	0.005	0.005	0.003	0.005	0.003
$torso_\beta^{var}$ (rad^2)	0.001	0.002	0.034	0.003	0.005	0.001	0.001	0.033	0.002
$torso_\gamma^{var}$ (rad^2)	0.062	0.033	0.017	0.051	0.018	0.033	0.009	0.025	0.044

The fitnesses scores (f) achieved by the *angle feedback* chromosomes are similar. Chromosome AF_1 achieved the highest median fitness of 11.56. For both *open loop* and *phase reset*, there was one chromosome which did not result in a stable walk. This can be seen in the low fitness score achieved by the chromosomes OL_3 (median $f = 5.894$) and PR_2 (median $f = 5.971$). The fitness scores of the different chromosomes are plotted in figure A.4.

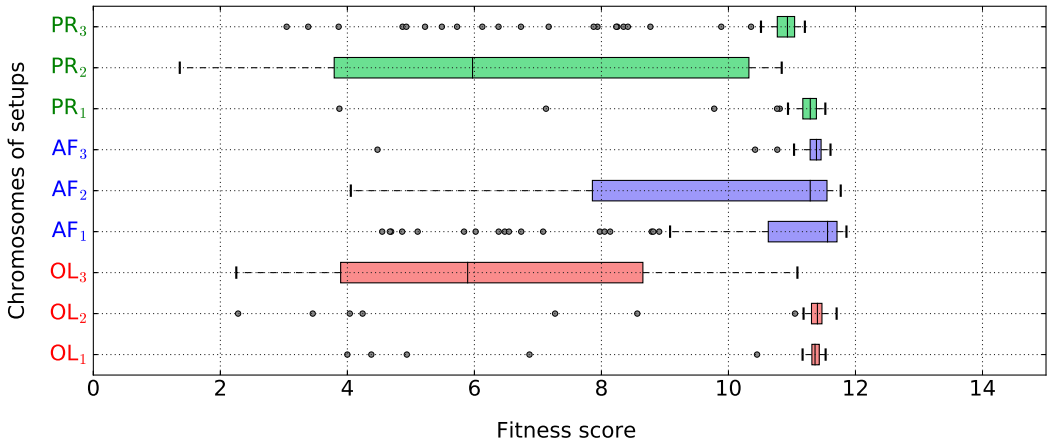


Figure A.4: Fitness f for 100 trials on $M2$.

The forward distances ($distance_x$) and absolute lateral deviations (dev_y^{abs}) of the robot during the gait evaluation tests are plotted in figure A.5. The highest median distance of 1.636m was achieved by AF_1. The performance of the other *angle feedback* chromosomes was also similar. For the chromosome OL_3 of *open loop* and chromosome PR_2 of *phase reset*, the median distance is negative. This shows that for these chromosomes, in most cases, the robot was unstable and fell down quite early. During a trial, if the forward distance was not much, and the robot fell down backward, its final distance was negative. Overall, the median distance scores of *phase reset* are worse than those of *open loop*, as shown in table A.3. From these results, it appears that the *phase reset* mechanism was not effective in the low-frequency gaits that evolved on machine $M2$. The absolute deviation (figure

A.5-right) shows similar values for all the chromosomes. Since the robot could not walk a long distance, the lateral deviation was also not very significant. Compared to the results obtained on machine $M1$ (figure 5.8), the distances on $M2$ are much lower.

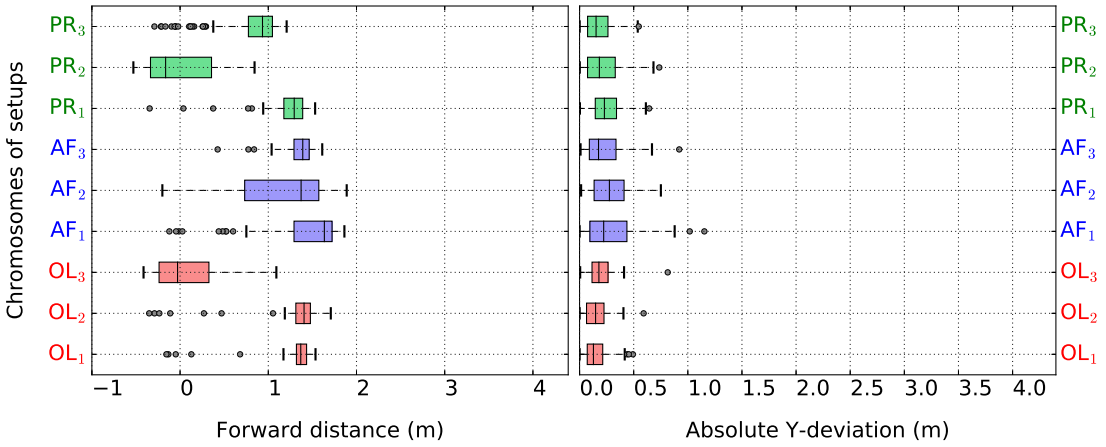


Figure A.5: (left) The forward distance $distance_x$ and (right) the absolute lateral deviation dev_y^{abs} for 100 trials on $M2$.

The average stride lengths of the robot for the different chromosomes are shown in figure A.6. There is a clear distinction between the average stride lengths of the *angle feedback* chromosomes and the stride lengths of the other chromosomes. The *angle feedback* chromosomes achieved longer stride lengths. The values for *angle feedback* on machine $M2$ are also somewhat similar to the values on machine $M1$ (figure 5.9).

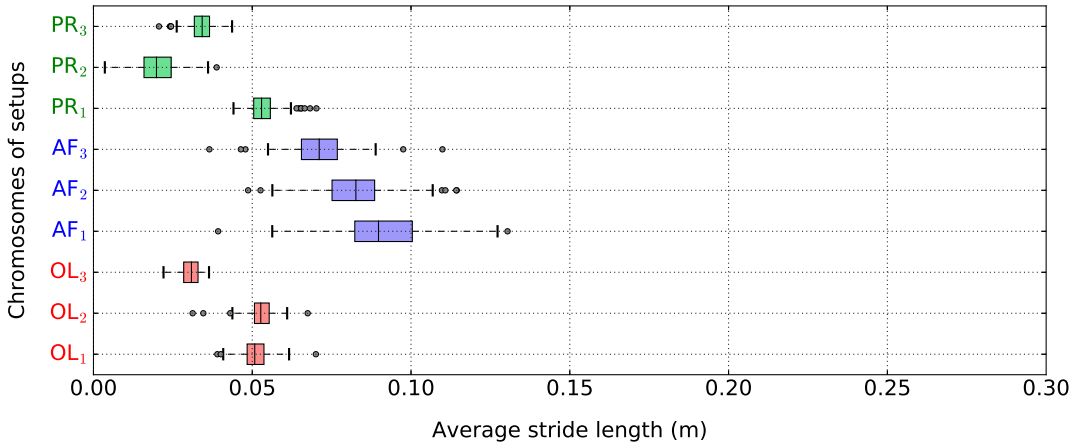


Figure A.6: Average length of stride $stride^{avg}$ for 100 trials on $M2$.

The time for which the robot was upright, during the evaluation of the different chromosomes, is shown in figure A.7. Apart from OL_3 and PR_2, all the other chromosomes were able to produce stable gaits, where the median value of t_{up} was equal to 20s (the maximum value possible).

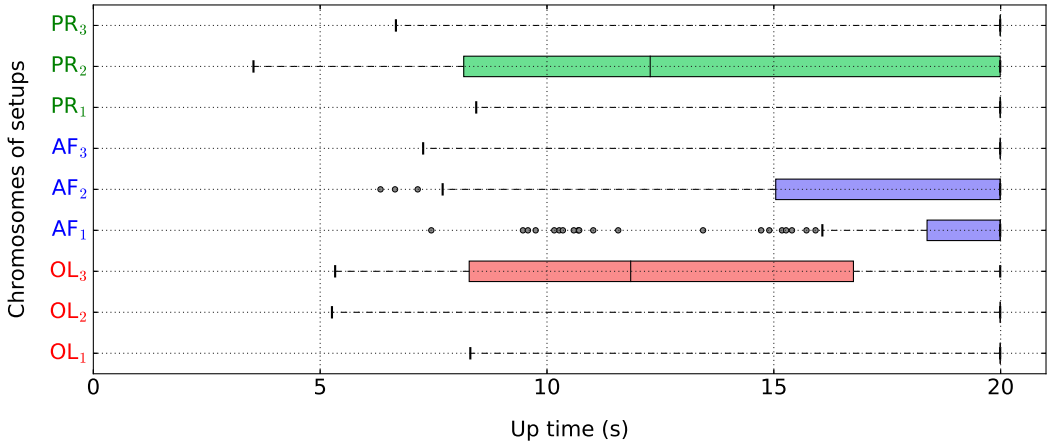


Figure A.7: Up time t_{up} for 100 trials on $M2$. When Q1=Q3=median, the boxes have zero width.

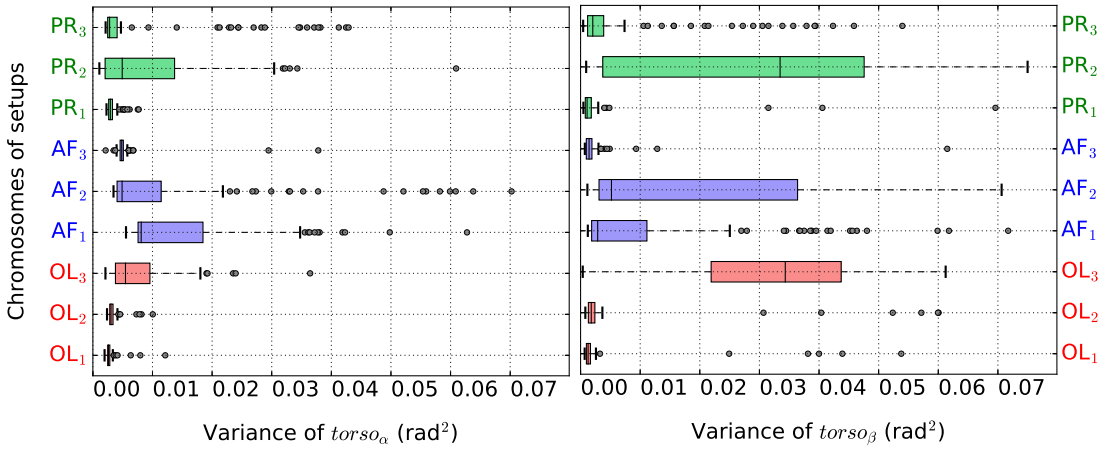


Figure A.8: (left) $torso_\alpha^{var}$ - variance in the torso's orientation about the world x-axis, (right) $torso_\beta^{var}$ - variance in the torso's orientation about the world y-axis, for 100 trials on $M2$.

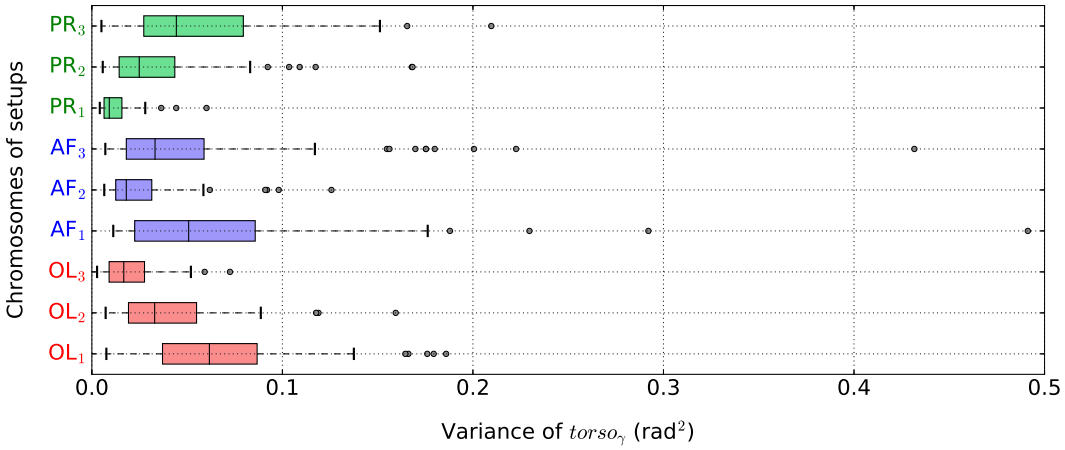


Figure A.9: $torso_\gamma^{var}$ - variance in the torso's z-axis orientation for 100 trials on $M2$.

The variance in the orientation of the robot's torso during the evaluation of gaits is shown in figures A.8 and A.9. For $torso_{\alpha}^{var}$, the median scores of all the chromosomes are similar. For $torso_{\beta}^{var}$, the chromosomes which could not achieve a stable gait (OL_3 and PR_2) have a high median score. The values obtained for $torso_{\gamma}^{var}$ are quantitatively similar for the different chromosomes.

Overall, on machine $M2$, the performance of *phase reset* was inferior to that of *open loop*. However, similar to machine $M1$, the performance of the *angle feedback* chromosomes was the best.

Bibliography

[1] Hayder FN Al-Shuka, F Allmendinger, Burkhard Corves, and Wen-Hong Zhu. Modeling, stability and walking pattern generators of biped robots: a review. *Robotica*, 32(6):907–934, 2014.

[2] A. D. Ames. Human-inspired control of bipedal walking robots. *IEEE Transactions on Automatic Control*, 59(5):1115–1130, May 2014.

[3] Shinya Aoi and Kazuo Tsuchiya. Stability analysis of a simple walking model driven by an oscillator with a phase reset using sensory feedback. 2006.

[4] P. Arena. The central pattern generator: a paradigm for artificial locomotion. *Soft Computing*, 4(4):251–266, Dec 2000.

[5] T Graham Brown. On the nature of the fundamental activity of the nervous centres; together with an analysis of the conditioning of rhythmic activity in progression, and a theory of the evolution of function in the nervous system. *The Journal of Physiology*, 48(1):18–46, 1914.

[6] Avis H Cohen and P Wallén. The neuronal correlate of locomotion in fish. *Experimental brain research*, 41(1):11–18, 1980.

[7] Steve Collins, Andy Ruina, Russ Tedrake, and Martijn Wisse. Efficient bipedal robots based on passive-dynamic walkers. *Science*, 307(5712):1082–1085, 2005.

[8] Sebastien Cotton, Ionut Mihai Constantin Olaru, Matthew Bellman, Tim van der Ven, Johnny Godowski, and Jerry Pratt. Fastrunner: A fast, efficient and robust bipedal robot. concept and planar simulation. In *Robotics and automation (ICRA), 2012 IEEE international conference on*, pages 2358–2364. IEEE, 2012.

[9] A Crespi and A.J. Ijspeert. Amphibot ii: An amphibious snake robot that crawls and swims using a central pattern generator. 01 2006.

[10] Julian Cristiano, Domenec Puig, and M Garcıa. Locomotion control of biped robots on uneven terrain through a feedback cpg network. In *XIV Workshop of Physical Agents*, pages 1–6, 2013.

[11] Julián Cristiano, Domènec Puig, and Miguel Angel García. Locomotion control of a biped robot through a feedback cpg network. In *ROBOT2013: First Iberian Robotics Conference*, pages 527–540. Springer, 2014.

[12] Julián Cristiano, Domenec Puig, and Miguel Angel García. Generation and control of locomotion patterns for biped robots by using central pattern generators. 2017.

[13] Igor Danilov, Bulat Gabbasov, Ilya Afanasyev, and Evgeni Magid. Zmp trajectory from human body locomotion dynamics evaluated by kinect-based motion capture system. In *VISIGRAPP (3: VISAPP)*, pages 162–168, 2016.

[14] M. Freese E. Rohmer, S. P. N. Singh. V-rep: a versatile and scalable robot simulation framework. In *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013.

[15] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pages 39–43, 1995.

[16] Agoston E Eiben and James E Smith. *Introduction to evolutionary computing*. springer, 2003.

[17] Gen Endo, Jun Morimoto, Takamitsu Matsubara, Jun Nakanishi, and Gordon Cheng. 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, 2008.

[18] S. Feng, E. Whitman, X. Xinjilefu, and C. G. Atkeson. Optimization based full body control for the atlas robot. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 120–127, Nov 2014.

[19] Andy Field. *Discovering statistics using SPSS*. Sage publications, 2009.

[20] L. A. Fuente, M. A. Lones, A. P. Turner, L. S. Caves, S. Stepney, and A. M. Tyrrell. Adaptive robotic gait control using coupled artificial signalling networks, hopf oscillators and inverse kinematics. In *2013 IEEE Congress on Evolutionary Computation*, pages 1435–1442, June 2013.

[21] Tao Geng, Bernd Porr, and Florentin Wörgötter. Fast biped walking with a reflexive controller and real-time policy searching. In *Advances in Neural Information Processing Systems*, pages 427–434, 2006.

[22] David E Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley Professional, 1989.

[23] David E Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. *Foundations of genetic algorithms*, 1:69–93, 1991.

[24] Sten Grillner. Neural control of vertebrate locomotion-central mechanisms and reflex interaction with special reference to the cat. In *Feedback and motor control in invertebrates and vertebrates*, pages 35–56. Springer, 1985.

[25] John H Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 2(1):84–90, 1960.

[26] Hooshang Hemami and R Farnsworth. Postural and gait stability of a planar five link biped by simulation. *IEEE Transactions on Automatic Control*, 22(3):452–458, 1977.

[27] Masato Hirose and Kenichi Ogawa. Honda humanoid robots development. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 365(1850):11–19, 2007.

[28] John H Holland. Genetic algorithms and the optimal allocation of trials. *SIAM Journal on Computing*, 2(2):88–105, 1973.

[29] Christian Hubicki, Jesse Grimes, Mikhail Jones, Daniel Renjewski, Alexander Spröwitz, Andy Abate, and Jonathan Hurst. Atrias: Design and validation of a tether-free 3d-capable spring-mass bipedal robot. *The International Journal of Robotics Research*, 35(12):1497–1521, 2016.

[30] A. J. Ijspeert and A. Crespi. Online trajectory generation in an amphibious snake robot using a lamprey-like central pattern generator model. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 262–268, April 2007.

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

[32] JBAJ Ijspeert. A simple adaptive locomotion toy-system. In *From Animals to Animats 8: Proceedings of the Seventh [ie Eighth] International Conference on Simulation of Adaptive Behavior*, volume 8, page 153. MIT Press, 2004.

[33] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[34] Takanishi Ishida and Yamazaki Kato. Realization of dynamic walking on biped locomotion robot wl-10rd. *Journal of Robotic Society of Japan*, 3(4):67–78, 1985.

[35] Akio Ishiguro, Akinobu Fujii, and Peter Eggenberger Hotz. Neuromodulated Control of Bipedal Locomotion Using a Polymorphic CPG Circuit. *Adaptive Behavior*, 11(1):7, 2003.

[36] I. B. Jeong, C. S. Park, K. I. Na, S. Han, and J. H. Kim. Particle swarm optimization-based central patter generator for robotic fish locomotion. In *2011 IEEE Congress of Evolutionary Computation (CEC)*, pages 152–157, June 2011.

[37] Chia-Feng Juang. A hybrid of genetic algorithm and particle swarm optimization for recurrent network design. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(2):997–1006, 2004.

[38] D Juričić and M Vukobratović. Mathematical modeling of biped walking systems. *ASME Publ. 72-WA/BHF*, 13, 1972.

[39] Shuuji Kajita, Hirohisa Hirukawa, Kensuke Harada, and Kazuhito Yokoi. *Introduction to humanoid robotics*, volume 101. Springer, 2014.

[40] Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kiyoshi Fujiwara, Kensuke Harada, Kazuhito Yokoi, and Hirohisa Hirukawa. 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, 2003.

[41] Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kazuhito Yokoi, and Hirohisa Hirukawa. The 3d linear inverted pendulum mode: A simple modeling for a biped walking pattern generation. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 1, pages 239–246. IEEE, 2001.

[42] Shuuji Kajita, Mitsuharu Morisawa, Kanako Miura, Shin'ichiro Nakaoka, Kensuke Harada, Kenji Kaneko, Fumio Kanehiro, and Kazuhito Yokoi. Biped walking stabilization based on linear inverted pendulum tracking. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 4489–4496. IEEE, 2010.

[43] Akiya Kamimura, Haruhisa Kurokawa, Eiichi Yoshida, Satoshi Murata, Kohji Tomita, and Shigeru Kokaji. Automatic locomotion design and experiments for a modular robotic system. *IEEE/ASME Transactions on mechatronics*, 10(3):314–325, 2005.

[44] Y. Kamogawa, K. Yamada, H. Masuta, and H. o. Lim. Stability control and pattern generation for biped humanoid robot. In *2013 13th International Conference on Control, Automation and Systems (ICCAS 2013)*, pages 910–915, Oct 2013.

[45] K. Kaneko, K. Harada, F. Kanehiro, G. Miyamori, and K. Akachi. Humanoid robot hrp-3. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2471–2478, Sept 2008.

[46] K. Kaneko, F. Kanehiro, M. Morisawa, K. Miura, S. Nakaoka, and S. Kajita. Cybernetic human hrp-4c. In *2009 9th IEEE-RAS International Conference on Humanoid Robots*, pages 7–14, Dec 2009.

[47] Kenji Kaneko, Fumio Kanehiro, Mitsuharu Morisawa, Kazuhiko Akachi, Go Miyamori, Atsushi Hayashi, and Noriyuki Kanehira. Humanoid robot hrp-4-humanoid robotics platform with lightweight and slim body. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 4400–4407. IEEE, 2011.

[48] Yi-Tung Kao and Erwie Zahara. A hybrid genetic algorithm and particle swarm optimization for multimodal functions. *Applied Soft Computing*, 8(2):849–857, 2008.

[49] Ichiro Kato. Development of wabot 1. *Biomechanism*, 2:173–214, 1973.

[50] J Kennedy and R Eberhart. Particle swarm optimization. *Proceedings of IEEE International Conference on Neural Networks*, 4:1942–1948, 2002.

[51] Matthias Kerzel, Erik Strahl, Sven Magg, Nicolás Navarro-Guerrero, Stefan Heinrich, and Stefan Wermter. Nico - neuro-inspired companion: A developmental humanoid robot platform for multimodal interaction. In *Proceedings of the IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 113–120, Aug 2017.

[52] Jesse Van Den Kieboom. Biped Locomotion and Stability - A Practical Approach. *Master's thesis, University of Groningen, The ...*, (March):66, 2009.

[53] Jeong-Jung Kim, Jun-Woo Lee, and Ju-Jang Lee. Central pattern generator parameter search for a biped walking robot using nonparametric estimation based particle swarm optimization. *International Journal of Control, Automation and Systems*, 7(3):447–457, 2009.

[54] Hiroshi Kimura, Seiichi Akiyama, and Kazuaki Sakurama. Realization of dynamic walking and running of the quadruped using neural oscillator. *Autonomous Robots*, 7(3):247–258, Nov 1999.

[55] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[56] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[57] J Zico Kolter and Andrew Y Ng. Policy search via the signed derivative. In *Robotics: science and systems*, page 34, 2009.

[58] J. Kuffner, S. Kagami, K. Nishiwaki, M. Inaba, and H. Inoue. Online footstep planning for humanoid robots. In *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, volume 1, pages 932–937 vol.1, Sept 2003.

[59] Scott Kuindersma, Robin Deits, Maurice Fallon, Andrés Valenzuela, Hongkai Dai, Frank Permenter, Twan Koolen, Pat Marion, and Russ Tedrake. Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot. *Autonomous Robots*, 40(3):429–455, 2016.

[60] M. A. Lewis, F. Tenore, and R. Etienne-Cummings. Cpg design using inhibitory networks. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 3682–3687, April 2005.

[61] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[62] Marco Locatelli. A note on the griewank test function. *Journal of global optimization*, 25(2):169–174, 2003.

[63] Zhenli Lu, Shugen Ma, Bin Li, and Yuechao Wang. Serpentine locomotion of a snake-like robot controlled by cyclic inhibitory cpg model. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 96–101, Aug 2005.

[64] Takamitsu Matsubara, Jun Morimoto, Jun Nakanishi, Masa-aki Sato, and Kenji Doya. Learning cpg-based biped locomotion with a policy gradient method. *Robotics and Autonomous Systems*, 54(11):911–920, 2006.

[65] K Matsuoka. Sustained oscillations generated by mutually inhibiting neurons with adaptation. *Biological cybernetics*, 52(6):367–376, 1985.

[66] Kiyotoshi Matsuoka. Mechanisms of frequency and pattern control in the neural rhythm generators. *Biological Cybernetics*, 56(5-6):345–353, 1987.

[67] Kiyotoshi Matsuoka. Analysis of a neural oscillator. *Biological cybernetics*, 104(4):297–304, 2011.

[68] Tad McGeer. Passive dynamic walking. *The international journal of robotics research*, 9(2):62–82, 1990.

[69] Giorgio Metta. icub: an open platform for research in robotics & artificial intelligence. In *CCIA*, page 5, 2015.

[70] Marcell Missura and Sven Behnke. Omnidirectional capture steps for bipedal walking. In *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 14–20. IEEE, 2013.

[71] Marcell Missura and Sven Behnke. Online learning of foot placement for balanced bipedal walking. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 322–328. IEEE, 2014.

[72] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[73] M. Okada, K. Tatani, and Y. Nakamura. Polynomial design of the nonlinear dynamics for the brain-like information processing of whole body motion. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, volume 2, pages 1410–1415 vol.2, 2002.

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

[75] Chandana Paul. Sensorimotor control of biped locomotion. *Adaptive Behavior*, 13(1):67–80, 2005.

[76] N. Perrin, O. Stasse, F. Lamiraux, Y. J. Kim, and D. Manocha. Real-time footstep planning for humanoid robots among 3d obstacles using a hybrid bounding box. In *2012 IEEE International Conference on Robotics and Automation*, pages 977–982, May 2012.

[77] Mitchell A Potter and Kenneth A De Jong. A cooperative coevolutionary approach to function optimization. In *International Conference on Parallel Problem Solving from Nature*, pages 249–257. Springer, 1994.

[78] Torsten Reil and Phil Husbands. Evolution of central pattern generators for bipedal walking in a real-time physics environment. *IEEE Transactions on Evolutionary Computation*, 6(2):159–168, 2002.

[79] L Righetti and A J Ijspeert. Programmable Central Pattern Generators: an application to biped locomotion control. *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, 2006(May):1585–1590, 2006.

[80] Ludovic Righetti, Jonas Buchli, and Auke Jan Ijspeert. Dynamic hebbian learning in adaptive frequency oscillators. *Physica D: Nonlinear Phenomena*, 216(2):269–281, 2006.

[81] Ludovic Righetti and Auke Jan Ijspeert. Pattern generators with sensory feedback for the control of quadruped locomotion. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 819–824. IEEE, 2008.

[82] Yoshiaki Sakagami, Ryujin Watanabe, Chiaki Aoyama, Shinichi Matsunaga, Nobuo Higaki, and Kikuo Fujimura. The intelligent asimo: System overview and integration. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 3, pages 2478–2483. IEEE, 2002.

[83] Jacob Shekel. Test functions for multimodal search techniques. In *Fifth Annual Princeton Conference on Information Science and Systems*, pages 354–359, 1971.

[84] Gordon M. Shepherd. *Neurobiology*. Oxford University Press, 1994.

[85] XH Shi, YC Liang, HP Lee, C Lu, and LM Wang. An improved ga and a novel pso-ga-based hybrid algorithm. *Information Processing Letters*, 93(5):255–261, 2005.

[86] Bruno Siciliano and Oussama Khatib. *Springer handbook of robotics*. Springer Science & Business Media, 2008.

[87] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 387–395, 2014.

[88] I.M. Sobol. On quasi-monte carlo integrations. *Mathematics and Computers in Simulation*, 47(2):103 – 112, 1998.

[89] Paul SG Stein. *Neurons, networks, and motor behavior*. MIT press, 1999.

[90] Shigeki Sugano and Ichiro Kato. Wabot-2: Autonomous robot with dexterous finger-arm–finger-arm coordination control in keyboard performance. In *Robotics and Automation. Proceedings. 1987 IEEE International Conference on*, volume 4, pages 90–97. IEEE, 1987.

[91] Tomomichi Sugihara, Yoshihiko Nakamura, and Hirochika Inoue. Real-time humanoid motion generation through zmp manipulation based on inverted pendulum control. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 2, pages 1404–1409. IEEE, 2002.

[92] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[93] Richard S. Sutton, Andrew G. Barto, and Harry Klopf. *Reinforcement Learning : An Introduction Second edition , in progress*. 2016.

[94] Gilbert Syswerda. Uniform crossover in genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms, 1989*, 1989.

[95] Csaba Szepesvári. Algorithms for reinforcement learning. *Synthesis lectures on artificial intelligence and machine learning*, 4(1):1–103, 2010.

[96] G. Taga, Y. Yamaguchi, and H. Shimizu. Self-organized control of bipedal locomotion by neural oscillators in unpredictable environment. *Biological Cybernetics*, 65(3):147–159, 1991.

[97] George E Uhlenbeck and Leonard S Ornstein. On the theory of the brownian motion. *Physical review*, 36(5):823, 1930.

[98] Miomir Vukobratović and Branislav Borovac. Zero-moment pointthirty five years of its life. *International Journal of Humanoid Robotics*, 1(01):157–173, 2004.

[99] Miomir Vukobratovic and Davor Juricic. Contribution to the synthesis of biped gait. *IEEE Transactions on Biomedical Engineering*, (1):1–6, 1969.

[100] Miomir Vukobratović and J Stepanenko. On the stability of anthropomorphic systems. *Mathematical biosciences*, 15(1-2):1–37, 1972.

[101] Matthew M. Williamson. Neural control of rhythmic arm movements. *Neural Networks*, 11(7-8):1379–1394, 1998.

[102] Jing Yang, Jing Ning, and Chengju Liu. Locomotion control of seven-link robot with cpg-zmp. In *Control Conference (CCC), 2016 35th Chinese*, pages 4517–4522. IEEE, 2016.

[103] Zhangguo Yu, Maoxing Zheng, Qinqin Zhou, Xuechao Chen, Libo Meng, Weimin Zhang, Aiguo Ming, and Qiang Huang. Disturbance rejection controller for biped walking using real-time zmp regulation. In *ROMANSY 21-Robot Design, Dynamics and Control*, pages 179–188. Springer, 2016.

[104] Dingguo Zhang, Philippe Poignet, Ferdinan Widjaja, and Wei Tech Ang. Neural oscillator based control for pathological tremor suppression via functional electrical stimulation. *Control Engineering Practice*, 19(1):74–88, 2011.

Erklärung der Urheberschaft

Hiermit versichere ich an Eides statt, dass ich die vorliegende Master's Thesis im Studiengang Intelligent Adaptive Systems selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel - insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Ort, Datum Unterschrift

Erklärung zur Veröffentlichung

Ich stimme der Einstellung der Master's Thesis in die Bibliothek des Fachbereichs Informatik zu.

Ort, Datum Unterschrift