Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät II
Institut für Informatik

Simloid

Evolution of Biped Walking Using Physical Simulation

Diploma Thesis

Daniel Hein
dhein@informatik.hu-berlin.de

October 27, 2006

Erklärung

Hiermit erkläre ich, die vorliegende Diplomarbeit selbständig und nur unter Zuhilfenahme der angegebenen Literatur verfasst zu haben.

Ich bin damit einverstanden, dass ein Exemplar dieser Arbeit in der Bibliothek des Instituts für Informatik der Humboldt-Universität zu Berlin ausgestellt wird.

Berlin, den 1. November 2006

Daniel Hein

Abstract. Locomotion can be defined as the capability for a living being or an inanimate object to move from one place to another. It is an essential task for any component of the animal world. In nature there are many different examples of locomotion means, such as legs, wings, fins. In human beings the evolution lead to the adoption of biped locomotion, which is the prerogative of few species.

Anthropomorphic fascination and the advantages of biped locomotion in environments with discontinuous support are among the reasons why walking and running biped robots have become a popular area of research. Biped robots could walk in almost any type of terrain included those that are impossible for robots with wheels. Biped robots compared with other type of robots are better skilled for certain works and have a better degree of mobility especially in environment with obstacles.

But controlling a biped robot with a high degree of freedom to achieve stable movement patterns is still an open and complex problem. Thus, developing of biped robots controlling algorithms have become an important field of research. With growing computational power of computer hardware, high resolution realtime simulation of such robot models becomes more and more applicable. This thesis presents a physical simulation of a 19 degree of freedom real biped robot model and the application of evolutionary algorithms to generate and optimize walking pattern generation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The subject of biped motion is a wide complex theme, that imports biological, physical, mechanical and mechatronical aspects. Within the field of humanoid robotics research further topics of controlling architectures and autonomous decision-making are raised: How to achieve stable biped motions? How to generate an energy-efficient walking? How to control all joints to achieve a specific aim?

This chapter gives a brief introduction into the topic of biped motion generation. In the face of the complexity and breadth of this subject it tries further to illuminate some related questions and introduces to some prerequisites to understand the processed problem.

1.1 Problem Description

Given a certain biped entity (e.g. a robot) the biped motion problem can be described as the control task of all joint actuators, given certain sensory informations to achieve a specific target. This is very general, but the problem itself depends on what actuators are given (e.g. muscles, motors), how the actuators are controlled (e.g. muscle contraction is stimulated by electrical impulses transmitted by the nerves vs. electricity causes servo motor torques or positional control via control circuit), what sensory information is provided (e.g. touch sensors, balance sensors, actuator force and/or position feedback sensors) and finally what target is pursued (e.g. stand up, balancing stand, walk, turn, kick, etc.).

Among some experimental actuators in humanoid robotics at present the commonly applied actuators are servo motor drives. Servo motors are well approved in

industry robotics for years and there is a wide range of available types, regarding e.g. size, torque or mechanical properties. Servo motors are typically driven via a control circuit supporting target positioning or target velocity setting. The most established control circuit is the PID controller, which is shortly explained in section 3.4.3.1. Anyhow, servo motor drives do not have implicitly the best properties for generating a human like motion behavior - see section 2.3.

Regarding the use of control circuit driven servo motors, the problem of biped motion generation can be reduced to the trajectory generation problem. The trajectory based approach points to the task of generating an ideal target angle function (= target trajectory) for each joint drive (= servo motor), given the available sensory information and the task target.

Another open question about biped locomotion refers to the higher level controlling of motion behavior: Meanwhile e.g. walking itself as a basic skill corresponds to a reactive and regular motion behavior without the need for higher level controlling, the task of piloting to a specific target needs a higher (deliberative) mechanism, that chooses, starts, stops and adjusts the lower basic skills using the feedback information by the sensors. Different architectures of information and control flow on specific robot types are already examined [29, 34], a general approach even for biped is still open to be discussed.

1.2 Related Work

Recently many robotics research groups have developed humanoids in academic institutes [21, 16], and in commercial companies [22, 58, 56]. In spite of the amount of already published work, there are few studies concerning whole body movement of a humanoid robot [37, 28]. Ogino [30] states two difficulties in generating humanoid behavior that e.g. a wheel type robot does not have: The first one is the large number of degrees of freedom (DoF). This causes explosion of combinations of motions that should be examined. The other is non-linear motion and intrinsic instability of a body balance. This makes it difficult to search without falling down in exploration space.

However, the general proposed methods for generating humanoid motions can be mainly categorized into two groups. One is the *model based approach*, and the other is the *model-free approach*. In the former, a designer precisely constructs a

physical model of the target system and builds a controller based on the precise model. In the latter, it is more important to make use of the intrinsic dynamics of a robot or to associate the sensor information with motions, instead of using the controller based on a physical model. For that reason, this approach has been attracting many researchers and it is called the *embodiment approach* [5] or *dynamics based approach* [20]. Masaki Ogino [30] gives a good status quo about present approaches, which are briefly discussed in the following sections.

1.2.1 Model Based Approach

Supposing that all information concerning the interaction between a robot and the environment is available in advance, a designer can predict the dynamics of the whole system and build a controller based on prediction. Conversely, a designer processes sensor information so that he/she can comprehend the whole system. One of the common features in this approach is to set the reference trajectories for each joint angle. The basic ideas for calculating the trajectories differs from methods to methods. The two well-established representatives of this approach are the Zero Moment Point (ZMP) and the inverted pendulum.

1.2.1.1 ZMP Based Approaches

The basic idea of the Zero Moment Point approach is to keep the projected point of center of mass (CoM) of a robot within the polygon area of the ground touching legs. The ZMP can be defined as the point on the ground where the total moment generated due to gravity and inertia of the moving system equals to zero [23, 26]. In terms of bipedal walking this means, the ZMP has to be kept always within the area of the foot of the support leg. Although ZMP originally refers to static motion generation, it was proposed so that the basic idea for the stability can be extended to the dynamic walking as well.

In the approaches based on the ZMP index, the reference trajectories to be followed by high gain PID[1] controllers are designed so that ZMP always exists within the polygon consisting of the support legs. However, the ZMP approach requires an exact physical model of the robot as well as precisely working joint actuators to ensure the exact following of the calculated trajectories. As an example, Takanishi

[1] see section 3.4.3.1

group in Waseda University presented the humanoid robot WABIAN [43], where the trajectories of the arms, legs and ZMP were described by Fourier series. The coefficients to ensure the ZMP conditions were determined in simulation.

1.2.1.2 Inverted Pendulum Model

In the single support phase human walking can be modeled as an inverted pendulum. Within a simple (kneeless) biped model, the swinging leg is represented by a regular pendulum, while the standing (or supporting) leg is represented by an inverted pedulum. The standing leg is to be controlled by the hip joint's torque. The corresponend dynamics equations of the model may be found by the Lagrangian method [8].

As an example, Kaneko et al. presents in [46] the application of a 3D linear inverted pendulum model to generate the walking pattern. Kajita et al. modeled a simplified robot in which the legs were massless, the upper body was a rigid inertial element and the robot walked with the center of mass at a constant height [45]. Komura et al. simulated human gait motion when muscles are weakened with appliance of an enhanced version of three-dimensional linear inverted pendulum model [1].

1.2.2 Dynamics Based Approach

This section presents some studies in the dynamics based approach. Considering the originating idea, they can be classified into three groups: The passive dynamic walking (PDW) approach, the central pattern generator (CPG) approach and the ballistic walking approach.

1.2.2.1 Passive Dynamic Walking

In the passive dynamic walking approach, the biped walks down on a shallow slope without any actuator. The only operating force descends from gravity. The walking motion is the result of two pendulums (the legs) swinging in their natural frequency. The mechanical design of the legs, such as length, mass distribution and foot form, determines the stability of the walking motion. The probably oldest record of a PDW is the patent about the shape of the feet of a walking toy by G. T. Fallis [9] (see fig. 1.1).

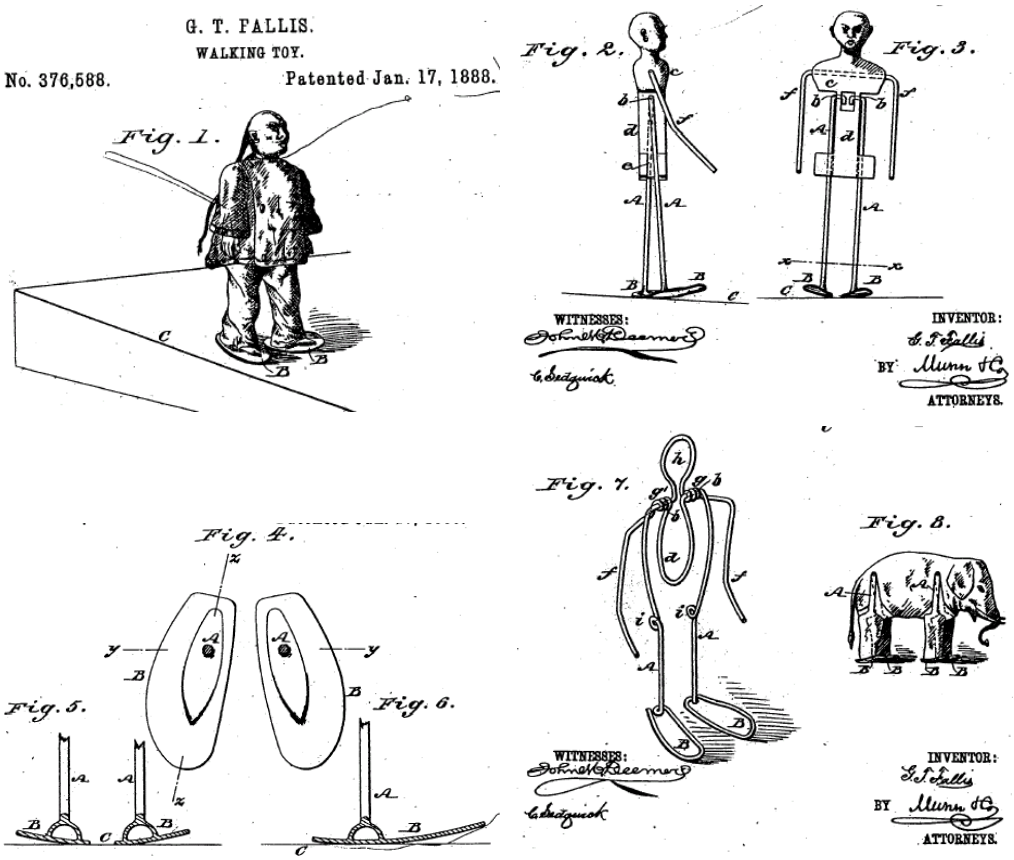

Figure 1.1: Oldest record of PDW: Walking toy by G. T. Fallis (1888)

Tad McGeer was the first, who applied the PDW idea in robotic research field. In 1990, he built a two dimensional bipedal robot with knee joints and curved feet [52, 53]. Leg length, mass, and foot shape were designed so, that the robot was able to walk down a three-degree slope with no actuators and no control system. Expanding on McGeer's work, a team of researchers at Cornell [42] constructed a three dimensional PDW that could walk the length of a three degree, five meter ramp. The interesting part of this research was, that due to the using of the intrinsic behavior of the robot's morphology an efficient and natural looking walking emerged. This demonstrated for the first time that the morphology of a robot might be more important than its control system. It also set a benchmark for walking machine efficiency. The estimated amount of potential energy used by this walker was only three watts.

1.2.2.2 Central Pattern Generator

Vertebrate animals are thought to have neural basis for locomotion in spinal cord, named central pattern generator (CPG) [50, 51, 2]. Central pattern generators are circuits which are able to produce periodic signals in a self-contained way, i.e. without having any rhythmic input to themselves. These rhythmic activities can be initiated by a simple non-oscillating (tonic) signal. In vertebrates, the CPGs consist of several oscillatory centers in the spinal cord. In legged animals, the locomotor CPGs often contain several centers that control the different limbs. These are further decomposed into subcenters for each joint which, in turn, have divided control for flexor and extensor muscles. Although CPGs do not necessarily need any sensory feedback to generate oscillatory output for the motor neurons in animals, it is important to coordinate the activities of the different oscillatory centers and to adjust the patterns to the changes in the environment. Sensor feedback is e.g. used to promote the switch from flexion to extension once the joint has reached its most posterior position.

In order to build structures with similar properties as the neural oscillators found in animals, several mathematical models have been proposed, e.g. [3, 13, 35]. Matsuoka proposed a mathematical model of CPG and demonstrated that the combinations of simple neural models can generate the neural activities for biped locomotion [24], which was used in several biped simulations (e.g. [44]) and in real robots (e.g. [12]).

What these models have in common is that they all are able to produce continuous oscillation signals when they are stimulated by a tonic input. By coupling the neural oscillators they are able to synchronize their frequencies. Two brief examples of synchronization by sensor coupling are shown in section 4.3.6.

One of the difficulties in application of CPG model to real robots is to determine the coefficients of neural connections. This is the main reason why genetic algorithm have been often used to solve this problem in general [4, 18]. Another problem is that almost all robots employs the proportional and derivative control for motor control in which the desired angle and angle velocity should be given, whereas, in human model, the outputs of the CPG neurons are used as torque (see also section 2.3).

1.2.2.3 Ballistic Walking

The ballistic walking model was originally introduced by Mochon and McMahon [54, 47]. The idea of this model is based on the observation of human walking, in which the muscles of the swing leg are activated only at the beginning and at the end of the swing phase. Thus, the model starts with a certain initial torque at the beginning of a leg's swing phase. Afterwards the leg moves only under its inertia and gravity, wherefore this model is called "ballistic". In the end of the swing phase again a torque is applied to ensure a certain entry leg angle for the beginning of the ground contact (or support) phase. Variant from this model, Ogino [30] defines the "ballistic walking family" as the walking controllers, in which the movement of the swing leg is governed by the gravitational and the inertial force in the middle of the swing phase.

In practice, several robots were presented with ballistic walking controllers (e.g. [25, 15]). Delft Biped Laboratory of Delft University in Holland built real robots with pneumatic actuators, which were only used for the moving of the swinging leg [27]. To ensure stability of the robot's walking, the robot itself was designed so that it could realize passive dynamic walking without any actuation. Interestingly, although pneumatic actuators are difficult for positional control, they could be successfully applied for this type of motion generation. Different actuation timings and working pressures of the pneumatic "muscles" allow different robust biped walking motions of the robot.

The ballistic walking approach is also a good source for exploring energy efficient walking pattern. Ogino demonstrates in [30] several biped simulations with ballistic walking controllers for up to seven DoF robots. The parameters of the controllers were determined by a higher level learning module to minimize the overall energy consumption.

1.3 Outline

The thesis at hand concentrates on the question, how the task of biped walking of a high DoF servo motor driven robot can be generated and described. Due to the high complexity of the underlying target system – the Bioloid robot, which is presented

in chapter 2 – robot's kinematic and non-linear behavior of the servo motor joints, a model-based approach is hardly applicable.

The aim of this work is to explore appropriate control structures which are potentially able to generate robust biped motions of the target system. Certain parameter instances of the structures are to be find with the appliance of genetic algorithms.

Two different low-level architectures generating joint target trajectories are presented and compared. Due to the walking task and the servo motor's properties, the presented approaches derive mainly from CPG approach. The architectures are applied to a physical simulation of the Bioloid biped robot. Further it is shown, how parameter sets for the individual architectures can be found using evolutionary algorithms. It is shown, that even with simple architectures without any sensor feedback a robust and fast biped walking can already be generated. Furthermore, two examples demonstrate, that the presented evolution setup and controlling structure can be applied also for other tasks than walking.

The thesis on hand is split into six chapters. After a brief introduction into the field of humanoid biped motion generation and presenting the basic approaches regarding biped locomotion within this chapter, the second chapter presents the underlying target system of this thesis – the Bioloid robot. It illuminates the robot's actuators and sensors. Furthermore it discusses the given joint actuators in face of the biped walking problem. The question of appropriate joint actuators is not further discussed within this thesis, but should classify the thesis' course and results.

The third chapter presents the simulation software, that was implemented within this thesis and which forms the base of this work. It introduces into the applied physics simulation library *ODE*, presents the simulated Bioloid robot, and explains the main parts of the simulation environment.

The fourth chapter deals with the core topic of this thesis: The biped robot's motion generation. It presents three different approaches how to control the joints of the robot to generate walking pattern. Moreover, it illustrates the processed experiments in the simulation and the application of artificial evolution to determine optimal parameter sets for the particular approaches.

Chapter five engages in the question, how the gathered knowledge of the simulation's outcome can transfered to the real world. The types of gathered knowledge are classified and some processed real robot experiments are briefly discussed.

The closing chapter summarizes and discuss the work, and gives a review about possible continuing topics that could carry on the processed work.

Appendix A lists the detailed parameters of the evolution setups for all drawn evolution runs. This data is intended for anyone, who wants to reproduce or continue the processed evolutions.

Chapter 2

Bioloid Robot

This chapter presents the target system of the thesis' approaches – the Bioloid robot, which constitutes the archetype of the physical simulation presented in chapter 3. Furthermore this chapter discusses the suitability of the given servo motor joints to generate a human like walking behavior.

2.1 RoboCup

For having an exchanging and discussion platform for recent research around robotics, the RoboCup [39] was founded in 1998. The RoboCup is an annual international event where participating teams play soccer against each other in several leagues. The different leagues point to different areas of research within the field of robotics – from mechatronical design of the robots up to software architectures and controlling strategies for constituting autonomous team work. The aim of the RoboCup Federation is to build a competitive team consisting of completely autonomous humanoid robots that have a realistic chance of winning against the human world champion soccer team by the year 2050.

The Humboldt University participates in 2006 for the first time in RoboCup's humanoid league. The deployed robots are based on a commercial construction kit named Bioloid.

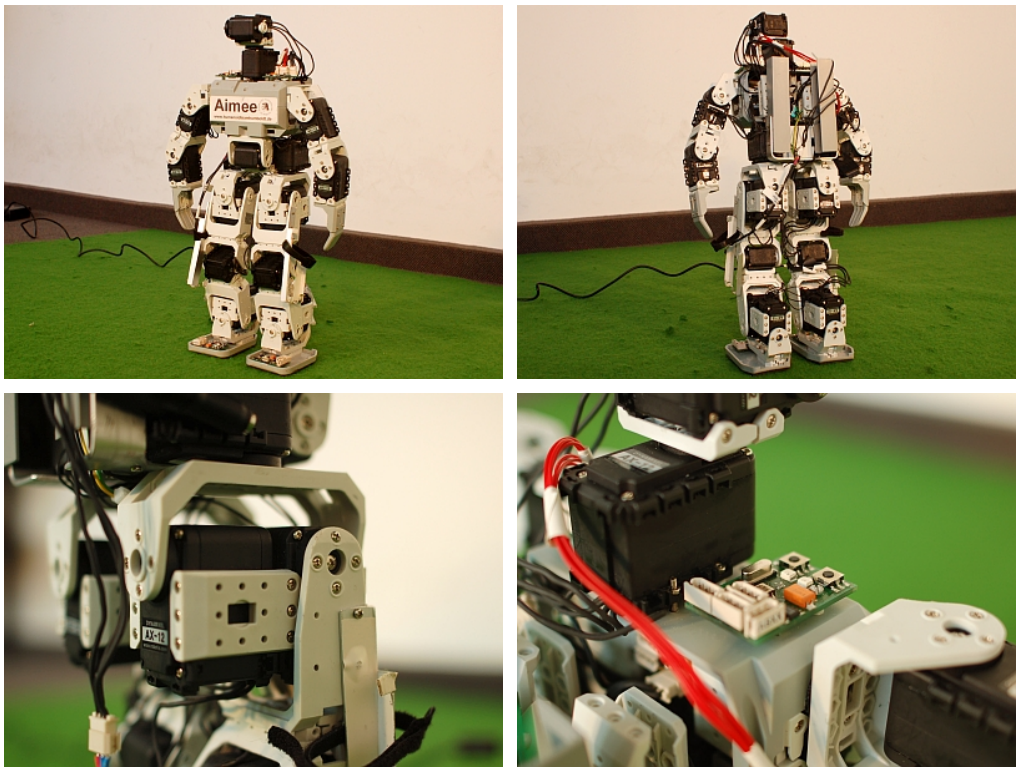

Figure 2.1: Bioloid: Above: Front view and back view without PDA and batteries. Below: Servo motor AX-12 at hip joint, acceleration sensor board at shoulder.

2.2 Bioloid

The commercial biped robot construction kit Bioloid is a 19 DoF robot and consists of approximately 700 basic parts. The assembled robot has a shoulder height of 34cm, weigh approximately 2.3kg and is actuated by 19 servo motors *AX-12*. All servo motors are controlled by a bus connector, allowing to set control commands and to get sensor information. The main control commands contain instructions for setting the goal position, the moving speed and the torque limit. Further, each servo motor provides informations about present position, speed and load. The kits were upgraded by acceleration sensors, developed and manufactured by Manfred Hild et al. [14], in total 8 sensors per robot, each sensor with two perpendicular sensoraxes. All acceleration sensors and servo motors are connected by a 100Hz clocked serial bus, called *spinal cord*. Furthermore for having a visual sensor, a camera is mounted on the body, controlled by two additional servo motors. Since the camera servos do

not control limb parts, they are not considered in this thesis. Figure 2.1 shows the assembled robot with the acceleration sensor boards.

2.3 Comparing Joint Drives

Biped motion and in particular biped walking has its archetype in human nature. But considering the actuators shows, that the control mechanism of human joint actuators is critical different to the mechanism of an electrical servo motor joint.

Most of human single dimension joints (one degree of freedom, e.g. knee, elbow) are more dimensional controlled: To exemplify consider the human elbow joint: To control the lower arm a human has two indepent muscle actuators - the biceps (flexor) and the triceps (extensor). Both muscles can be contracted independently, hence four general states of control activation are possible: With both muscles released the lower arm is laid-back and follows all external forces. A human uses this state e.g. to move down the lower arm by using the (external) gravity. When activating one of both muscles, the lower arm gets accelerated, for example for moving up or for throwing a ball. With both muscles tensioned, the arm is fixed at a certain position, but with elastic and damping properties. This state can be applied e.g. to position the arm or to catch a fast ball. Due to the elastic and damping properties of this state, the joint is able to absorb high kinetic impulses without getting damaged.

In contrast, a servo motor drive has only one control dimension: the current of the motor itself (what is done in fact in servo motor technique is a pulse-width modulation, so the control dimension consists in the pulse width modulation ratio). This enables to apply a certain force at the joint, but only in one direction at one time. Thus, by nature it is not possible to tense it at a position, similar to human joints. Only together with a control circuit it is possible to fix and position the joint at a certain angle. Anyhow, what is done with a control circuit is a steady comparing between present and target angle and to correct with the one dimensional control mechanism. Additionally, a servo motor has gears between the motor and the (outgoing) driven axis. Due to the ratio and the inner friction of the gears, a servo motor is not capable of quickly absorbing high kinetic external impulses. Catching a fast ball would face a joint break instead of absorbing the kinetic energy of the ball.

	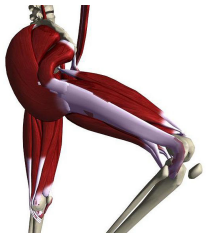	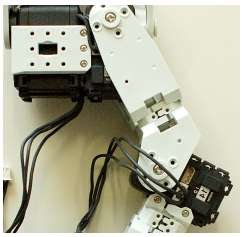
Type	**Muscle**	**Servo Motor**
Drive	Contraction	Voltage / Electric Current
Control Dimension per Joint	2	1
Characteristics	Scalable Tension	Scalable Torque
	Low Friction	High Friction
	Elastic	Inflexible
	Damping	Stiff
	No Tolerances	Gears Tolerances

Table 2.1: Comparing muscle and servo driven joints. The different design of the joint drives causes completely different characteristics regarding capacity of walking behavior.

Thus, using servo motor joints running and jumping movements could just be imitated - to make use of intrinsic dynamics is hardly possible.

Table 2.1 summarizes the most important differences between muscle and servo motor driven joints.

Chapter 3

Simulation Environment

Part of this thesis was the development of a simulation environment *Simloid*, which incorporates a physical simulation of the Bioloid biped robot (*Simloid* - *Sim*ulation of the Bio*loid*). This chapter introduces into some basics of physical simulation and presents the developed simulation environment with brief descriptions of its main modules.

3.1 Motivation

Research on motion pattern generation on real robots has some major barriers: Hardware easily gets broken, experiments need manual supervision, exact environmental information of experiment states are hard to achieve, batteries need to be charged and finally the number of robots limits the number of parallel experiments.

A simulation of a robot model could eliminate these disadvantages. But the benefit of a simulation depends strongly on the congruence of the characteristics between the real and simulated hardware. This point refers to the modeling of the real hardware, including adjusting all behavioral aspects of real world, such as e.g. friction and collision properties between all simulated parts.

Figure 3.1 draws the ideal course of use of a simulation environment to solve a real world problem. The course of solving a problem using simulation consists of mainly three phases: The *modeling phase*, the *phase of experiments in the simulation* and the *knowledge transfer to real world*.

The phase of modeling is the most sensitive one regarding the validity of the whole simulation process, since the congruence of real and simulated world behav-

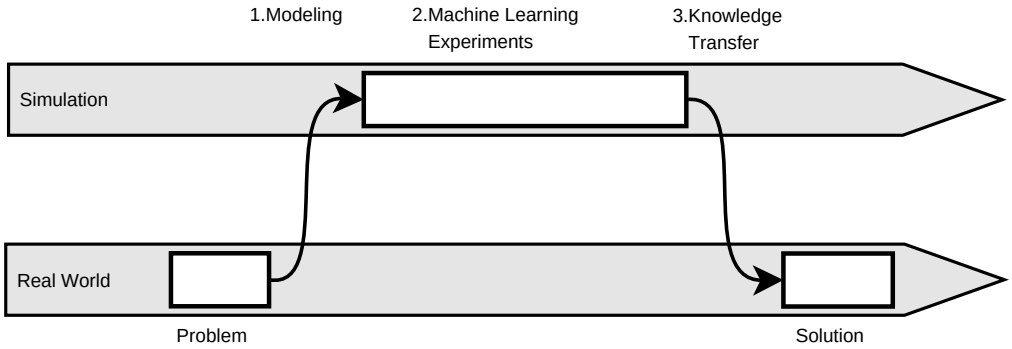

Figure 3.1: Ideal course of simulation benefit to solve a problem in real world. Due to continuously comparing and adjusting between real and simulated world behavior, in practice the phase of modeling often extends up to the end of the experiments phase.

ior depends on it. For a complex system, such as in this case of modeling a highly detailed robot, an exact match of real and simulated world is hardly possible. It depends on the system to be simulated, on the problem to be solved and on the desired yield of knowledge, what degree of detail is required for the modeling to have an appropriate benefit of the simulation experiments. Anyway, in practice simulation validating by continuously adjusting and comparing between real and simulated world leads to a steady modeling phase up the end of simulation experiments. Another approach to model nonlinear systems using coevolutionary algorithms is introduced by J. C. Bongard et al. [17]. Within this approach, both the structure of the system and informative tests to extract new information from the system to be modeled are evolved to converge model and target system behavior.

Once a system is modeled, all experiments can be done in the simulation without stressing the hardware. The aim of the experiments phase is to achieve the desired knowledge for solving the problem in real world. This can be either (direct) knowledge that represents the solution of the problem in real world (e.g. concrete values for controlling the servo motors) or *meta knowledge* that represents knowledge about how to solve the problem in real world (e.g. setup and values of a machine learning method that solves the problem). While working with complex systems which are hard to simulate exactly the latter one will be mostly the more applicable knowledge (see also chapter 5).

The aim of the knowledge transfer is to apply the extracted knowledge from simulation experiments in real world. Due to differing behavior of real and simu-

lated world, in most cases the gained knowledge has to be adjusted to real world to get best results.

3.2 ODE

The biped robot simulation in this approach is based on the discrete physical simulation environment *ODE - Open Dynamics Engine* [48], originated by Russel Smith. ODE is an open source, high quality library for simulating articulated rigid body dynamics. It is fully featured, stable, mature and platform independent with an easy to use C/C++ API[1]. It has advanced joint types and integrated collision detection with friction. ODE is useful for simulating vehicles, objects in virtual reality environments and virtual creatures. It is currently used in many computer games, 3D authoring tools and simulation tools.

3.2.1 Body and Joint Primitives

While working with ODE, all participating physical entities to be simulated are constructed of atomic rigid body primitives. These primitives include sphere, box, different cylinder types, infinite plane, ray and triangle mesh objects. Each rigid body has several constant properties like mass, its center of mass and mass distribution. Other properties change over time. These are its position and orientation in space and further linear and angular velocity. ODE provides forces and torques as the two basic concepts used to act on rigid bodies. These two concepts model all interesting properties one expects from a physical simulation.

Body primitives can be physically connected by *joints*. Joints are used to actively enforce a relationship between two connected bodies. Presently ODE supports the following joint types: ball and socket, hinge, two-hinge, slider, universal, fixed and angular motor joints (see figure 3.2 to get an idea how they work). Joints are nothing else than constraints, that limit the relative movement of the two connected bodies to along one or more axes. Additionally joints can act as motors by enforcing the movement along the non-restricted axes. The angular motor joint allows the relative angular velocities of two bodies to be controlled. Further ODE provides to restrict the possible rotation angle of a joint by setting *joint stops*. Joint stops are used for

[1] Application Programming Interface

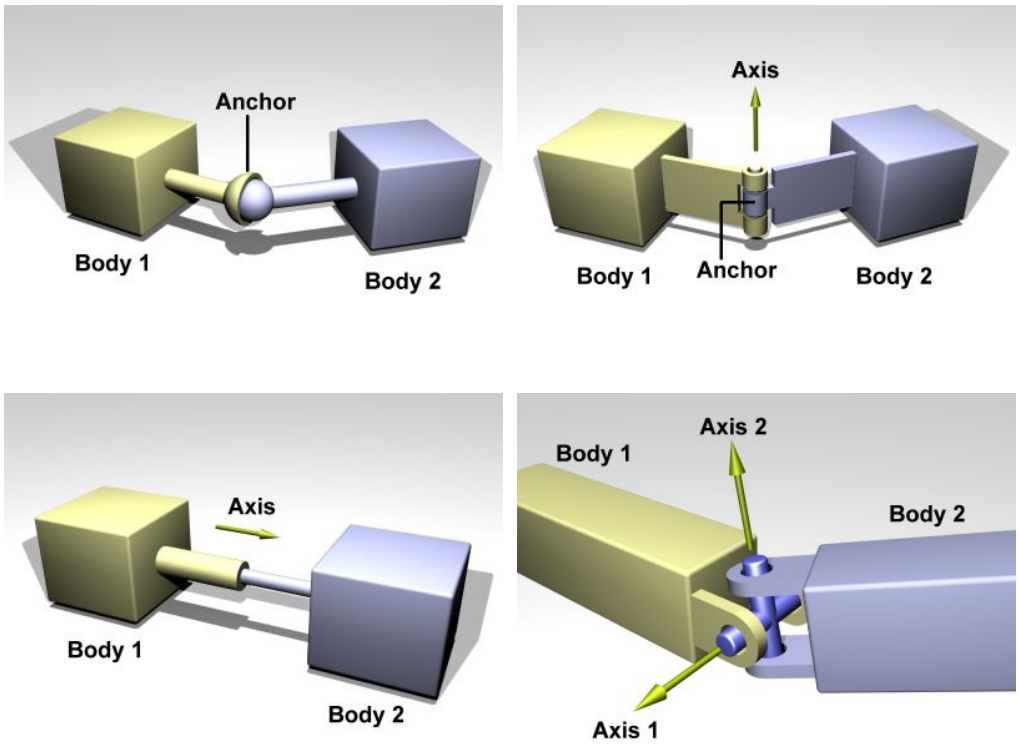

Figure 3.2: ODE joint types: Above: ball and socket joint, hinge joint. Below: slider joint, universal joint.

the AX-12 servo motors joints as well to set the valid working range. A set of bodies which are connected with joints form an *articulated structure*, used to simulate e.g. vehicles or legged creatures.

3.2.2 Collision Detection

When two bodies collide, they influence each other, which can be accurately described in terms of forces and torques that are applied on the two colliding bodies. As mentioned, ODE provides collision detection and handling. This is done via so called *collider*, which correspond to the geometric shape of the bodies. Their only purpose is to detect intersections with other colliders. A collider does usually not model the exact shape of the associated visible object but a computationally less expensive one. When a collision is detected it must be resolved. The correct forces that prevent the objects to interpenetrate must be applied to the bodies. This is done with the help of so called *contact joints* that are temporary generated in response

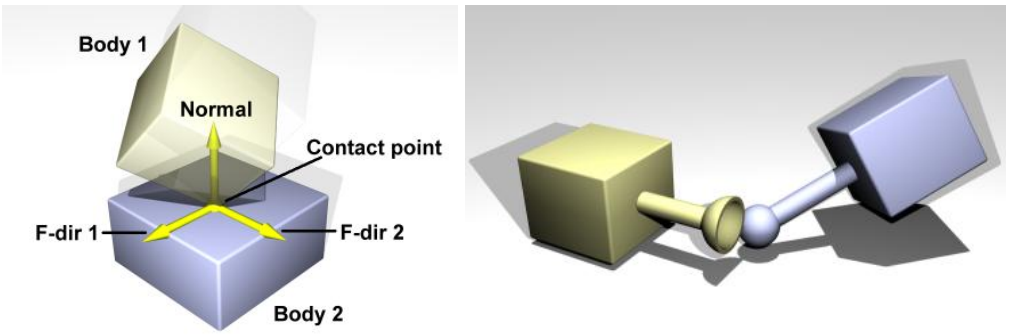

Figure 3.3: Joints are constraints: A joint constrains the relative movement of the two connected bodies along one or more axes. Left: Contact joints prevent two bodies from inter-penetrating. Right: Broken ball and socket joint constraint. ODE performs a steady error reduction by applying additional correcting forces.

to a detected collision (see figure 3.3, left). A contact joint has several parameters allowing to simulate different collision and friction behavior. Although ODE treats all bodies as pure rigid, this parameterized collision handling enables to simulate e.g. damping, elasticity or slipping behavior between two bodies. In particular this is useful for simulating different floor types. Notice that even the standing of a body on another (or e.g. on the floor) is a steady collision - the body is only kept in position by collision handling forces.

From computational point of view, collision detection has to be done at each simulation step for every single pair of body geometries. For N objects this is $O(N^2)$. For simulated environments with many objects, collision detection may become computationally expensive. ODE provides some strategies, to optimize the collision detection using *spaces*. A space accumulates a set of body geometries and can quickly identify pairs of geometries, that are potentially intersecting. Finally, for saving computation time the collision detection may be disabled for any body geometry.

3.2.3 Simulation Loop

ODE makes use of the time integrated approach. This means, the physical simulation is done by advancing in discrete time steps. The increment of each single

simulation step can be chosen arbitrary. In general smaller increments (=higher time resolution) will give a more stable and accurate simulation. On the other hand, of course the increment (or *step length*) influences linearly the simulation speed.[2] Within each integration step, the new states of all simulated rigid bodies are calculated and adjusted. Further, at each step the user can interact with the world by e.g. adding forces or torques to bodies or getting informations about their states. Thus, one simulation step consists in mainly three phases:

1. Users interaction with world (setting/getting body states, forces, torques).
2. Collision detection by ODE (marking colliding bodies).
3. Physical simulation step, advancing the simulation time (calculating new states of bodies).

During the physical simulation step, which advances the simulated time from t to $t+1$, the new states (position, rotation, linear and angular velocity) of all bodies are calculated, taking into account all given joint constraints. It has to be mentioned, that constraints are generally not completely fulfilled: In order to have a stable and reasonable simulation behavior while working with the discrete time integrated method constraints usually have a certain temporal and local valid error range. Different parameters control the error ranges, the interested reader wants to have a look at [41] for *Joint error and the error reduction parameter (ERP)* and *Soft constraint and constraint force mixing (CFM)*.

3.3 Simloid - Simulation of the Bioloid

To model the given Bioloid robot, all detailed physical properties of it have to be known. Hence, the robot was disassembled to get all mass and lengths properties of the single parts. As mentioned above, the robot consists of more than 700 mechanical parts, not including the whole electric part - cables, connectors, whole inner life of servo motors, individual parts of the acceleration sensor boards are not counted. The modeling of every single part would not only be time expensive, it would not make sense at all, since the simulation could not handle reasonable this amount of parts.

[2] see also section 3.4.5

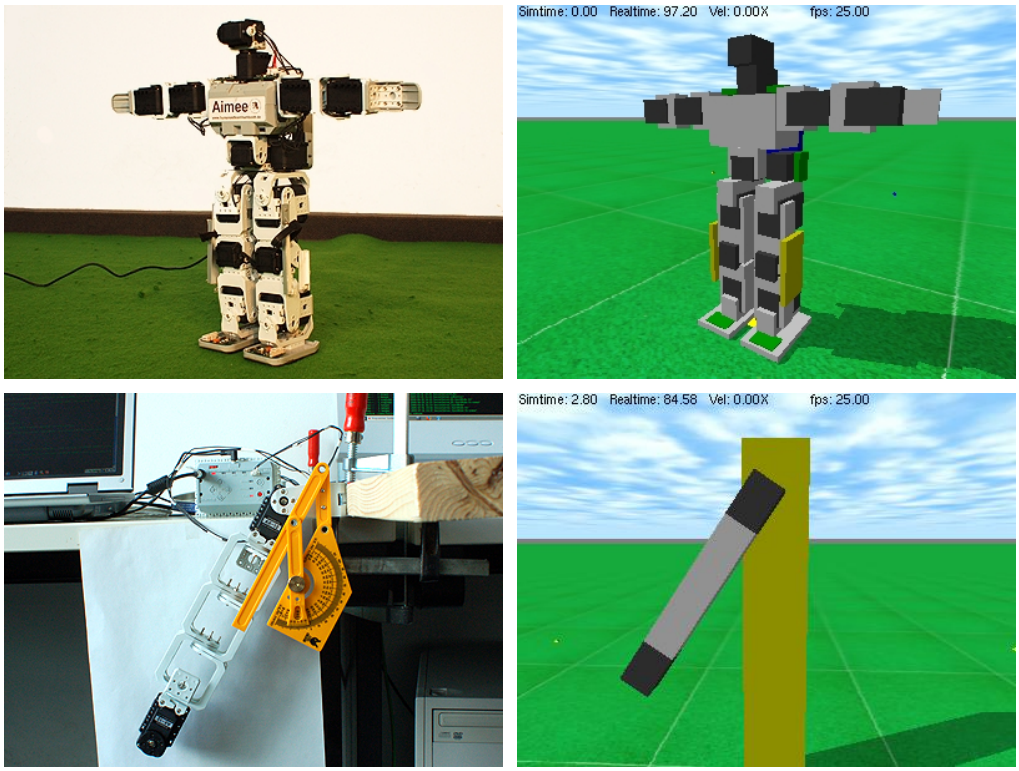

Figure 3.4: Real and simulated world: Above: Real Bioloid, Simulated Bioloid (Simloid). Below: Real servo motor torque and friction experiment setup, Simulated servo motor torque and friction experiment setup.

Hence, all statically connected parts were modeled as one ODE rigid body, except the acceleration sensors boards. But most of these rigid body parts are composed of more detailed parts - either to get a more detailed surface (geometry) at complex parts, or to simulate accordingly the inertia tensor at non-uniformly distributed masses.

Actually the simulated Bioloid consists of 28 rigid bodies, formed by 57 box geometries and 19 virtual servo motor joints. All lengths, masses and axis positions correspond to the real robot model. The servo motors are simulated with adequate torque and friction forces, determined by isolated motor characteristic experiments. The acceleration sensors are simulated accordingly, getting congruent values to the real sensor boards. The virtual environment simulates a gravity of $9.81 m/s^2$. The simulated static friction is adjustable with different models and parameters, allowing to simulate different floor types. Air drag is actually ignored.

3.3.1 Simulation of servo motor joints

Meanwhile single body related properties like mass, position and dimension of a part are quite simple to detect and implement, the measuring and modeling of dynamic body interacting properties like friction between two parts or the effect of active actuators are much harder. While simulating an actuators behavior, it has to be distinguished between its *passive behavior* and its *active behavior*.

Passive behavior of a servo motor means, the behavior of the two connected bodies results only from external forces (e.g. gravity) or torques that effect the bodies and the inner state of the bodies under the constraint of the servo motor joint properties.

Active behavior of a servo motor means, among the influence of external forces and torques the behavior of the two connected bodies is also affected by internal forces - the applied torques of the servo motor itself.

Both the passive and the active behavior have to be modeled appropriately to get a reasonable joint simulation behavior. The still open problem consists in finding a reasonable model that describes all properties of these behaviors. Thenafter appropriate experiments have to be set up to determine the correspondent parameters of the model.

In practice another problem of modeling a real servo motor is the gearbox between the motor and the driven axis, which causes a complex friction behavior that is hard to model exactly. It has influences in both the passive and the active behavior of the joint. For example the friction depends highly on the relation between internal and external forces. Meanwhile the inner friction of the servo becomes high when internal and external forces work against each other, it can become nearly negligible when they work in same direction. Additionally it effects a tolerance between the two connected bodies, which is caused by bearing and tooth clearances inside the gearbox.

In terms of ODE, the present model of the AX-12 servo motor joints consists of a hinge joint and an angular motor. The hinge joint constrains the relative position and rotation of the two connected bodies, the angular motor enables to apply a torque between the two connected bodies. Further the hinge joint enables to set joints stops, that limit the rotational working range, and the angular motor permits to simulate friction. In the presented simulation a linear friction model is implemented, which is

done by applying a steady torque against the current rotation direction. The torque is calculated linearly to the angular velocity of the joint.

3.3.2 Comparing Bioloid - Simloid

While working with physical real world simulation, the fidelity and validity of the simulation outcome is the most important key issue. From point of theoretical view, we assume equal states both in real and simulated world. A simulation matches real world, if a given control pattern leads in real as well as in simulated world to the same state, thus all world physical aspects have to be equal after executing the control pattern.

However, due the use of simplifying approximations and assumptions within the simulation, real and simulated world will never match exactly. Since the number of states and the number of control pattern is infinite, it is hard to demonstrate the degree of simulations validity.

Nevertheless, some random breath testing could demonstrate a quite good correlation between real and simulated worlds behavior: Control pattern that were developed and tuned on real hardware were transfered to the simulation. This was done for a stand up moving pattern and for different walking patterns. The simulation showed quite same behavior as the real robot.

Regardless, there are still some remarkable differences observed between simulation and real world. They can be categorized into two groups: Firstly, movements in the simulation are more stable than with real robot. Secondly, some walking pattern that work good with real hardware tend to slip in the simulation causing little progress of locomotion.

The present differences are caused mainly by the joint tolerances, which are already discussed in section 3.3.1. The effect of the tolerances grows with leverage that effects a servo motor. Thus, the movement of the ankle joints is strongly influenced by these tolerances. The missing tolerance simulation causes, that movement patterns which are developed and tuned on real hardware do not work with same results in simulation. Another point are the material properties of the body primitives or parts respectively: While ODE simulates a single part as an ideal rigid body, in real world they are not. The effects are similar to the gearbox tolerances, except that they apply to the bodies instead of the joints. But in contrast to the adjustable joint behavior, with ODE there is no way of simulating accordingly deformable bodies.

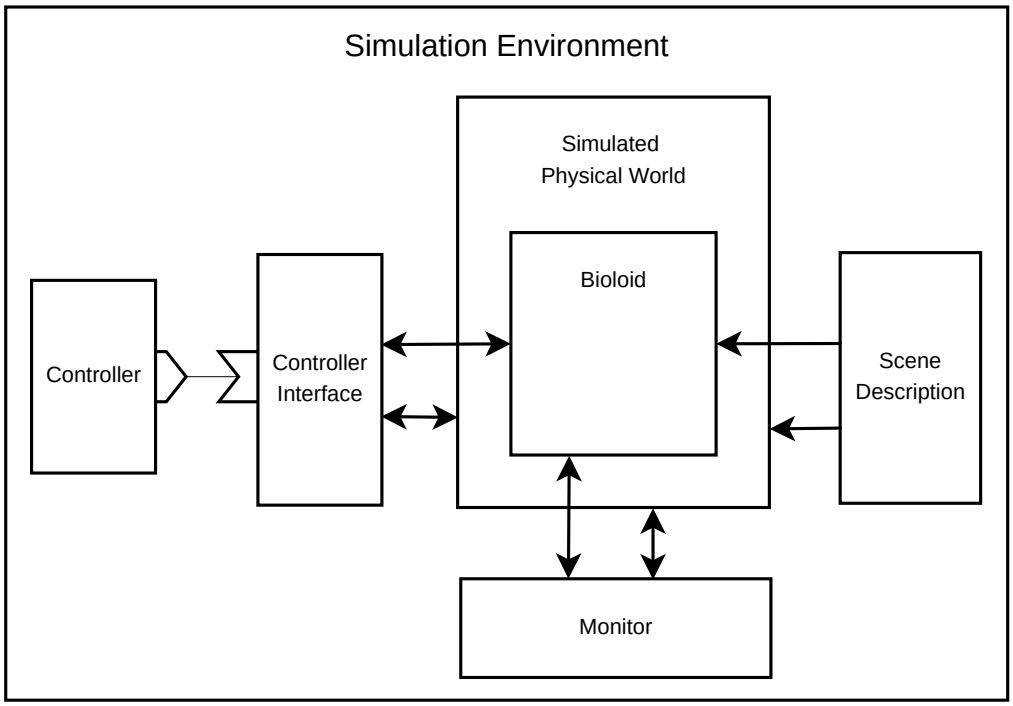

Figure 3.5: Basic modules of the simulation environment.

3.4 Architecture

The whole simulation environment is written in C++ and consists of 33 classes. Among some commonly used basic functionality they can be categorized roughly into the *simulated world*, which contains the biped robot model, the *scene description module* and the *controller module*. In addition a *monitor module* provides three-dimensional graphical output of the simulated world's state using the OpenGL interface. The monitor underlying drawstuff library is based on the ODE's package delivered one, extended by functions to visualize some alphanumeric data and a modified camera handling. The monitor module is optional since it is not necessary for the (numerical) simulation itself. Figure 3.5 gives a graphical overview of these modules. They are briefly presented in the following sections.

3.4.1 Simulated World and Main Loop

The whole state of the world is kept in a list of *body objects*. A body object is a structure, that stores the name, the color, a pointer to the ODE bodies geometry and a pointer to the ODE bodies physical state. Every single body participating in the

simulation is kept as a single element in this list. A second list keeps pointer to all joint objects of the simulation.

When the simulation environment is started, the world is built up by informations of the *scene description module* (see section 3.4.2) and the lists get filled. After some additional initialization of the simulation, the *main loop* is started. Within the main loop, the *controller module*, the *monitor module* (if enabled) and the ODE physics itself get successively in turn access to this body objects list. The main loop acts as a timer - it controls when the monitor (if enabled) is called to redraw the scene, when the collision detection is called, when the world's state is advanced by the ODE physics calculation and when the controller gets access to the simulation. Figure 3.6 shows a flow chart of a simulation run.

3.4.2 Scene Description Module

The scene description module holds all information about how the world is build up at the beginning of the simulation. This includes the description of all participating simulated systems – e.g. the Bioloid robot. It has a lower layer interface, the *primitives module* and two upper layer interfaces, the *Bioloid module* and the *experimental module*.

The primitives module encapsulates methods for creating basic elements of the ODE simulated world. These methods include creating boxes, spheres, servo motor joints and body geometries. Additionally a macro for creating and positioning acceleration sensor boards is provided. It works as a wrapper around the basic ODE interface methods to simplify the handling of the objects primitives. It further provides small statistics summary, to detect e.g. the weight, the number of atomic parts or the center of mass of a complete simulated system (robot model). The primitives module interfaces are used by the upper layered Bioloid and experimental module.

The Bioloid module contains a method to create and destroy a Bioloid in the ODE world. It fills the body objects list using the primitives module. The module contains all modeling information, hence all weights, lengths, positions, orientations, joint axes and color information of the simulated Bioloid robot.

The experimental module is a container for some experimental setups in the simulaton. This was used in particular to test and compare the servo motor joint behavior. During a (pure) Bioloid simulation run this is not used, but mentioned here for the sake of completeness.

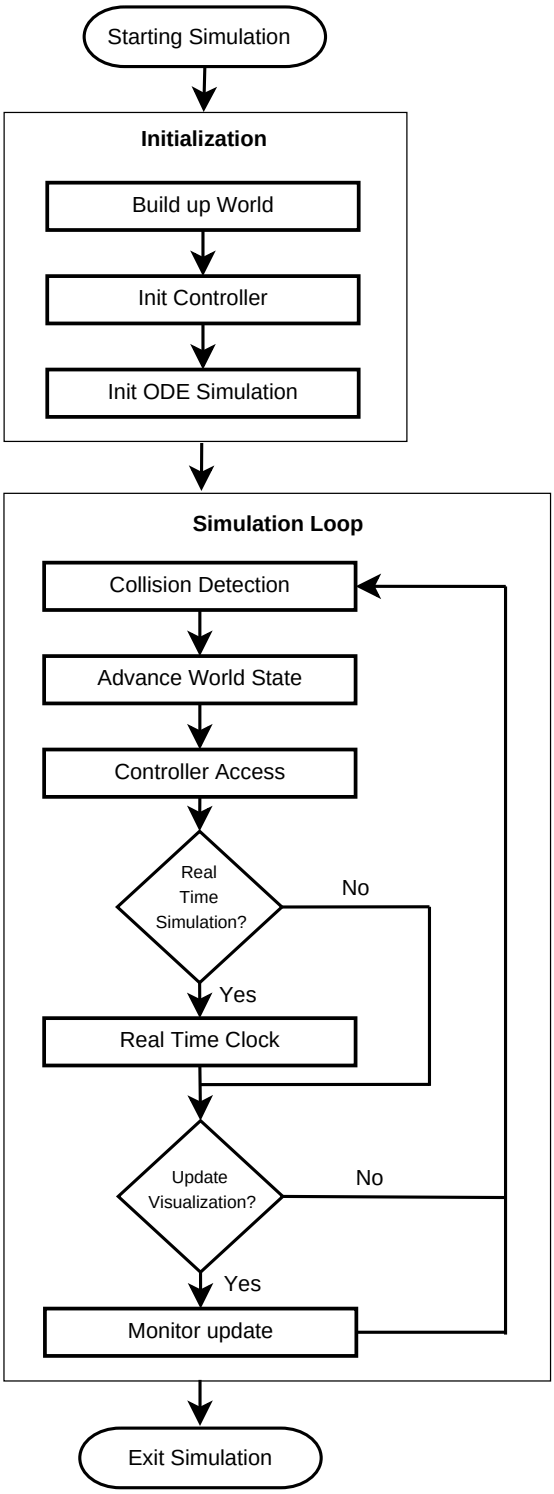

Figure 3.6: Flow chart of a simulation run. Simplified view for a single run.

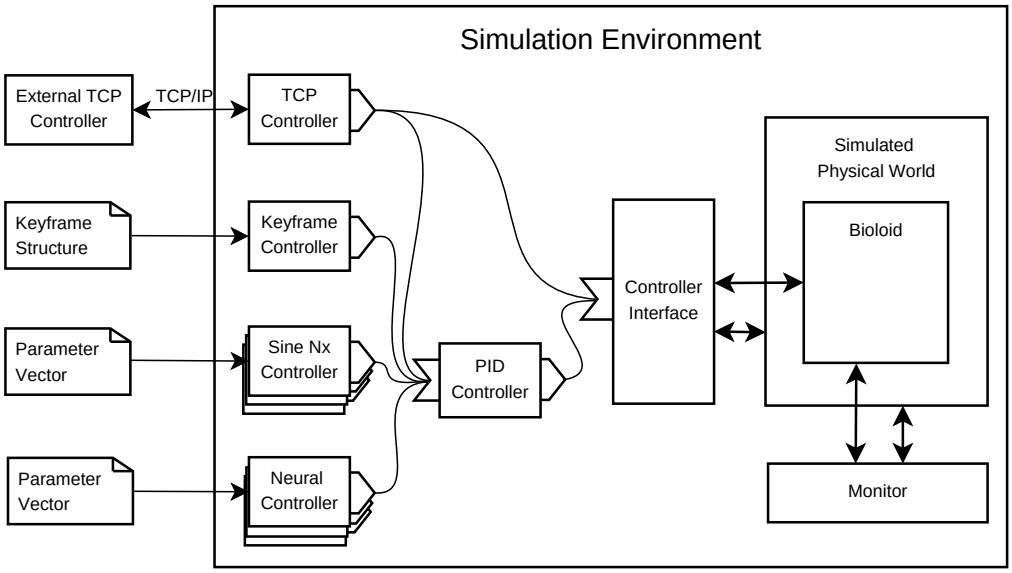

Figure 3.7: Overview about presently implemented controller modules.

Remark: Conceptually, the modeling data of the simulated world could be kept in an external structure, e.g. a configuration file. Since the scene description can be handled as a language [6, 49], the scene description module could be substituted by a scene description language parser. As an example, the agent-based, physical simulation framework SPARK[3] by Markus Rollmann makes use of such a parser [31].

3.4.3 Controller Module

To control the simulation a controller interface is provided. An object that uses this interface is called a *controller module*. This architecture allows to easily implement, test, interchange and compare different controller modules. The controller interface is fully synchronized with the simulation - at each simulation step the controller gets complete access to the body objects list[4] before the simulation is advanced. Complete access means, it can read and/or manipulate the state of each simulated body object. This includes the control of the actuators and the sensor data gathering. Actuator controlling in the case of servo motor joints is nothing else than a state manipulating access of the corresponded angular motor joints. Sensor data gathering

[3]SPARK is a generic simulator for physical multi-agent simulations, recently used for the RoboCup's 3D soccer simulation league

[4]see section 3.4.1

means the raw state reading of all sensor related bodies and/or joints. Further, the controller interface allows to reset the whole simulation, meaning all bodies are relocated to their initial state, setting the simulation time to zero and where necessary resetting some additional structures (e.g. PID-Controller, see section 3.4.3.1). The reset functionality is even used for the episodic evolution experiments, which are presented in chapter 4.

The main task of a controller module is to control all actuators of the simulated robot. Thus, the critical component regarding motion generation is the control architecture of this module. To give a review, all implemented controller modules are briefly presented in the following sections. The details and underlying concepts of the controller modules are presented in chapter 4.

3.4.3.1 PID Controller

A PID controller (*P*roportional-*I*ntegral-*D*erivative controller) is a common feedback loop component in industrial control systems. The controller compares a measured value from a sensor with a reference setpoint value. The difference (or ”error” signal) is then used to calculate a new value for a manipulatable input to the process that brings the sensors' measured value back to its desired setpoint. Unlike simpler control algorithms, the PID controller can adjust process outputs based on the history and rate of change of the error signal, which enables a flexible, more accurate and stable control.

The output of a PID controller is calculated by three terms: The proportional term, the integral term and the derivative term. The three terms are defined below:

$$P_{contrib} = K_p e(t) \tag{3.1}$$

$$I_{contrib} = K_i \int_{-\infty}^{t} e(t')dt' \tag{3.2}$$

$$D_{contrib} = K_d \frac{de(t)}{dt} \tag{3.3}$$

$$\Rightarrow Output = K_p e(t) + K_i \int_{-\infty}^{t} e(t')dt' + K_d \frac{de(t)}{dt} \tag{3.4}$$

where K_p, K_i and K_d are the proportional gain, integral gain and the derivative gain, e is the error and t the current time. These constants influence the characteristic of the PID controllers output (response time, error handling) and are generally

adjusted once. More detailed informations about theory and work of PID controllers can be found at [19].

As mentioned in section 3.2.1, the control interface of an ODE joint allows only to apply a torque or - with the use of an angular motor - to set a target relative angular velocity. The implemented PID controller, which can be plugged between the controller interface and the controller module, allows to set target angles of the joints. This in turn is used when generating target trajectories with the trajectory based approach (see section 1.1). The PID controller controls the relative angular velocity, although it could be applied to the torque setting interface, too. The gain parameters are adjustable via the simulation configuration.

3.4.3.2 TCP Controller

The TCP[5] controller is not a standalone enclosed controller model, but allows to control the simulation by an external program. This controller was used for a students laboratory [33], where isolated tasks, such as balancing problems, where investigated with appliance of machine learning algorithms. Due to the TCP/IP communication there is no restriction of program languages or operating systems working with the simulation.

The controller provides a TCP/IP port interface, where the external program is connected to. The string based communication follows the sense-think-act cycle and is synchronized with the simulation - at each simulation step the controller receives all sensory information and has to react to advance the simulation. The provided sensory information is configurable and adjusted to the sensory information of the real robot. For machine learning experiments additional feedback data is provided (e.g. hip position). During the thinking cycle, the controller may send different control commands to the TCP controller until it sends a 'done' command confirming the end of the thinking cycle. This controller reduces the capabilities of actions and the amount of information adjusted to the real Bioloid robot control capabilities.

Table 3.1 gives a review about provided environmental information and the set of commands.

[5] transmition control protocol

	Bioloid	**Simloid**
Joint Angles	$0 \ldots 1024$ $(0° \ldots 300°)$	$0 \ldots 1024$ $(0° \ldots 300°)$
Acceleration Sensors	$-1.0 \ldots +1.0$ $(1.0 \approx 11.72 \frac{m}{s^2})$	$-1.0 \ldots +1.0$ $(1.0 \approx 11.72 \frac{m}{s^2})$
Present Moving Speed	$0 \ldots 1024$ $(0 \ldots 6840° s^{-1})$	$0 \ldots \infty° s^{-1}$
Hip Position	n.a.	$\checkmark([x, y, z])$

Table 3.1: Comparing available sensory information between real and simulated robot.

	Bioloid	**Simloid**
Goal Position	$0 \ldots 1024$ $(0° \ldots 300°)$	
Max Torque	$0 \ldots 1024$ $(0 \ldots \sim 16.5 kgfcm)$[6]	
Torque	$0 \ldots 1024$ $(0 \ldots \sim 16.5 kgfcm)$[6]	

Table 3.2: Available (at present identical) actuation commands of real and simulated robot.

3.4.3.3 Keyframe Controller

The keyframe controller is a target trajectory generator. The basic concept is to blend between different robots poses. The poses are kept in keyframe structures. Transitions describe, how to blend from one pose to another. The concept is presented in detail in section 4.1.

One aim of this controller module was to validate the simulation against real world behavior. This controller enables to apply real robot motions in the simulation: At present, the motions of the real robots are generated by the keyframe-transition approach. The motion descriptions are kept in keyframe-transition structures. These structures can be imported into the simulation via the keyframe controller. As a result, several motions, such as e.g. the stand up motion of the robot, can be reproduced in the simulation with similar behavior (see section 4.1).

[6] in case of the real Bioloid maximal torque depends on operating voltage

3.4.3.4 Sine Nx Controller

The sine target trajectory controller makes use of parameterized trigonometric functions. Each joint trajectory is described by such a cyclic function. The Nx label refers to the function structure, where $x = 0, 1, 2, \ldots$ declares the parameter space of the structure. The concept is presented in detail in section 4.2.

3.4.3.5 Neural Oscillator Controller

The controlling architecture of the neural oscillator controller consists of a neural network. Instead of trigonometric functions, the core oscillations in the neural oscillator controller are generated by a recurrent two neuron network, a neural oscillator. The target trajectories are given by the activities of the correspondent neurons. The concept is presented in detail in section 4.3.

3.4.4 Evolution Module

The applied method to find optimal parameter structures in this thesis is artificial evolution [36]. Artificial evolution enables to solve/optimize a given problem without having an explicit model of the problem. It is performed by the *evolution module*, which is broadly configurable allowing to accomplish a wide range of evolution characteristics. It is even configurable while during a running evolution, allowing to change evolution parameters over time.

One main feature of the implemented module is the pulling architecture: Compared to usual top-down evolution architectures, where the evolution process controls the start of the fitness determing process of an individual in this case individuals to be tested are fetched by the fitness determining process itself. This architecture enables an easy distribution of the artificial evolution among multiple instances of the fitness determining processes, while the processes can be dynamically attached and detached to the evolution process.

3.4.4.1 Task Modules

The direction and results of the individual's behavior during an artificial evolution depends on the fitness function. Since the fitness function defines (more or less) the task to be solved, it is encapsulated in an own structure called *task module*.

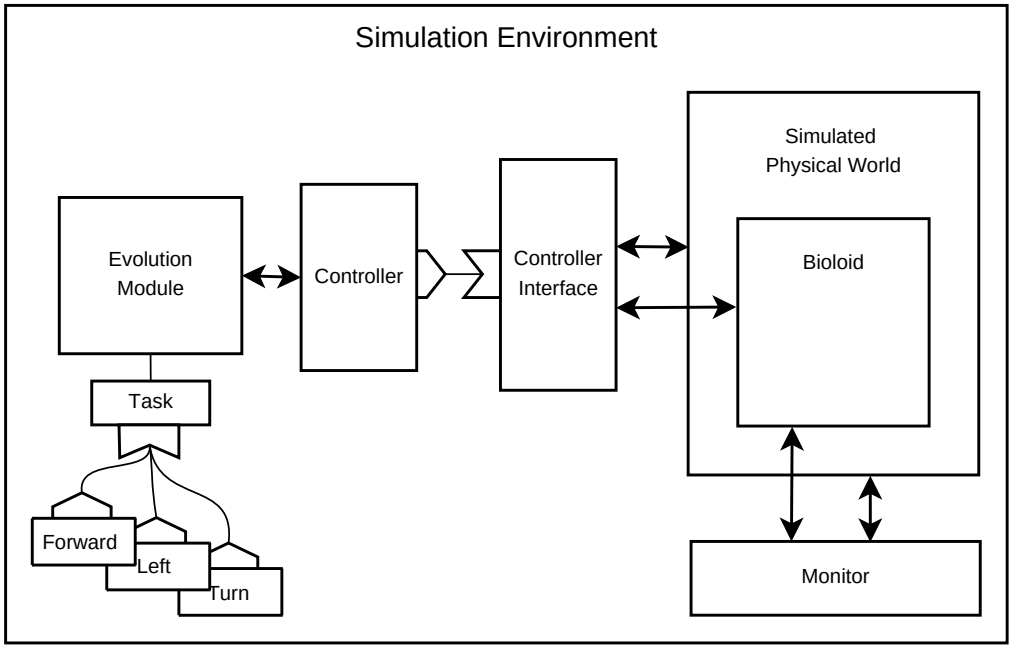

Figure 3.8: Evolution setup: Main modules of the simulation environment. The evolution module performs an artificial evolution to solve/optimize a given task. The tasks are pluggable.

A task module observes an episodical run of an individual and calculates in the fitness value of the observed individual. This architecture enables a modular task plugging. It allows to construct easily new fitness functions, and to test, interchange and compare them. A task defines the fitness value at the end of an episode and optional abort conditions, which can abort a running episode to save calculation time.

3.4.5 Simulation speed

As mentioned in section 3.2.3, the chosen step length of a single simulation step conditions the all over simulation speed. The present complexity of the simulation environment with all its modules and controllers needs about $0.002 - 0.001s$ per one simulation cycle, or rather allows to calculate $500 - 1000$ simulation steps per second as measured by actual hardware (3Ghz CPU). Thus, choosing a step length of $0.01s$ leads to a simulation/real time ratio of about $5 - 10$, a step length of 0.02 leads to a ratio of about $10 - 20$ and so on. Within this thesis, after some tests step

lengths between $0.005s$ and $0.02s$ were proved to be good compromises between simulation accuracy and simulation speed.

The presented results in this thesis are the outcome of about 14.000 (realtime) hours of simulation. This corresponds to a simulation time of about 210.000 (simulated) hours.

Chapter 4

Motion Generation

This chapter presents three different approaches, how to control the joints of the biped robot to achieve a certain motion. Among some additional tasks, the main focus was set on walking pattern. All approaches are designed to generate target trajectories. The target trajectories are handled by PID controllers[1], which control the joints. All presented approaches were implemented in controller modules which were introduced in section 3.4.3.

4.1 Keyframe-Transition Approach

The keyframe-transition approach consists of a directed graph, with keyframes as nodes and transitions as edges. A keyframe is a snapshot of a robot's pose. Thus, a keyframe is a vector of all joint angles of the robot corresponding to this pose. A transition connects one keyframe with another. Along the transition all joint angles are linearly interpolated over time. The transition describes, how the interpolation is done. Playing a keyframe-transition structure means, starting from a keyframe all joint angles corresponding to the keyframes along the transition path describe the target angles of the robot's servo motors.

The keyframe-transition model is continuously enhanced by the Humanoid Team Humboldt [14]. Meanwhile e.g. informations about torques to be used and selectors to switch between different outgoing transitions are added to the transition structures. Furthermore environmental feedback, such as measured joint angles are incorporated in the model.

[1] see section 3.4.3.1

Figure 4.1: A first weak validation of the simulation: The stand up motion is keyframe-transition based and was developed on the real Bioloid. The (raw) transfer of the identical keyframe structure to simulation shows similar behavior.

This approach is applied at the present motion control framework of the real Bioloid robot. The advantage of this approach is the easy and fast way of motion pattern generation - for example a working walking pattern could already be generated by four keyframes. The poses may be knead by hand and together with software tools[2] the motions can be tuned or expanded. A given keyframe-transition structure is easy to analyze and to modify. The graph model allows to switch between different motion patterns (e.g. walk and stand).

On the other hand, the linearly interpolated target trajectories do not model natural motions. Smooth curves can only generated by a higher resolution of keyframe points on the trajectory graph, which are harder to handle in turn. A second point is the question of sensor coupling. At present, sensor feedback is only used for topple detecting, to initiate a stand up motion after a downfall. But using of sensor feedback for interferencing the motion generation in order to stabilize the motion or to detect and handle obstacles is hard to implement and may need a physical model within this approach.

4.2 Cyclic Function Approach

In general biped walking is a more or less cyclic motion with a period length of one full step. Each joint repeats its motion after one period. Based on this fact, this approach assumes a cyclic ideal angle function for each joint. One period of each function corresponds to one full step, hence all functions have the same frequency.

4.2.1 Partial Fourier Series

The most elementary cyclic function we know from nature is - among the trivial constant function - the sinusoid. Its shape describes for example the motion of a tuning fork or the harmonic oscillation of a physical pendulum.

Thus, with given frequency we assume for each joint j the following parameterized sine function:

$$\alpha_j(t_p) = a_{j_0} + a_{j_1} * sin(t_p + b_{j_1}) \tag{4.1}$$

where t_p is the periodic time and $\alpha_j(t_p)$ is the target angle of joint j at time t_p. The parameters a_{j_n}, b_{j_n} describe the trajectory shape: a_{j_0} determines the offset, a_{j_1} the

[2] e.g. *motion editor* by Christian Thiele

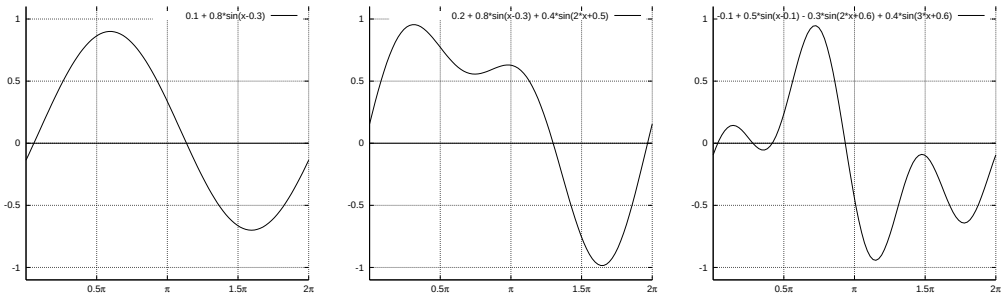

Figure 4.2: Example wave shapes for parameter space $N = 1$, $N = 2$ and $N = 3$. Graphs show one period. With growing N more complex shapes are possible.

amplitude and b_{j_n} the phase shift of the joint oscillation. Note that t_p is calculated by:

$$t_p = t_{real} \cdot f \cdot 2\pi \tag{4.2}$$

where t_{real} is the realtime and f is the full step frequency.

Although this a quite simple and restrictive motion model with few parameters, a robust forward walking could be generated with walking speeds up to $0.07 m/s$.

However, due to the restriction of one single amplitude per fullstep, human-like walking is not reachable: Since some human joints (e.g. knee and ankle) work with double step frequency during walking (compare figure 4.17), they need a trajectory shape of at least two possible amplitudes. This is done by adding a double frequency term to equation 4.1:

$$\alpha_j(t_p) = a_{j_0} + a_{j_1} * sin(t_p + b_{j_1}) + a_{j_2} * sin(2t_p + b_{j_2}) \tag{4.3}$$

Analog to a_{j_1}, b_{j_1} the additional parameters a_{j_2}, b_{j_2} describe amplitude and phase shift of the double frequency term.

This principle can be carried on by adding a triple frequency term and so on. In general this is a called a *partial Fourier series* which is given by:

$$\begin{aligned} f(t) &= A_0 + A_1 sin(\omega t + \phi_1) + A_2 sin(2\omega t + \phi_2) + ... + A_N sin(N\omega t + \phi_N) \\ &= A_0 + \sum_{n=1}^{N} A_n sin(n\omega t + \phi_n) \end{aligned} \tag{4.4}$$

where $f(t) = \alpha_j(t_p)$, $A_n = a_{j_n}$, $\omega = f * 2\pi$ and $\phi_N = b_{j_n}$.

Figure 4.2 shows example wave shapes for parameter spaces of $N = 1$, $N = 2$ and $N = 3$. It is shown, that for $N \to \infty$ any cyclic wave shape can be generated [32]. In this thesis wave shapes generated from $N = 1$, $N = 2$ and $N = 3$ were examined and compared.

4.2.2 Symmetry Assumption

The presented partial Fourier series has with given frequency $1 + 2N$ parameters. Without any constraints for 19 joints this would be $19 + 38N$ parameters. The dimension of the parameter space has a critical influence to the fitness progress, since more dimensions enable a larger set of possible individuals. Thus reducing the parameter space reasonable may sensibly facilitate the exploring of the search space for parameter solutions. The main consideration is to cut off the parameter space without reducing the space of optimal individuals.

The reduction in the case of walking is based on the assumption, that walking is a symmetric motion: A motion of a left sided joint is the same as the correspondent joint on the right side with a phase shift of half a period. Thus, for two symmetric joints the amplitude and phase shift of the individual Fourier series' terms have only to be determined once for a pair of symmetric joints. Assuming same (sagittal mirrored) orientation of two symmetric joints j and k the correspondent parameters would be:

$$a_{k_n} = a_{j_n} \tag{4.5}$$

$$b_{k_n} = b_{j_n} + \pi \tag{4.6}$$

The Bioloid's bodywork has 18 symmetric joints (6 per leg, 3 per arm) plus the "waist" joint in the middle of the body. Hence assuming symmetric motions the search space can be reduced to $9 + 1$ trajectory functions or rather $10 + 20N$ parameters using the partial Fourier series approach. Table 4.1 gives a review for the parameter dimension for $N = 1$, $N = 2$ and $N = 3$.

4.2.3 Evolution Setup and Results

This section describes the setup and results of artificial evolutions using the cyclic function approach. Among some additional tasks, the primary object was to cover a long as possible distance within a given time. For determining the fitness value of an

Parameter Space	**Parameter Space Dimension**	
	Without any Reduction	Symmetry Assumption
$N = 1$	57	30
$N = 2$	95	50
$N = 3$	133	70

Table 4.1: Overview about parameter dimensions for 19 joint trajectories using partial Fourier series as cyclic functions. Comparison between raw and reduced parameter space with symmetry assumption.

individual, an episode has to be processed. Each episode starts with the relocation of the robot to its initial position, processing the *init phase* (explained below) in order to pass the episode. Figure 4.3 shows a general evolution process flow and displays the interrelationship between the simulation, the evolution and the task module. Among the following presented setup, further parameters can be found in appendix A.

Genotype description. A single individual is given by its specific genotype. The genotype in this approach is a vector of double numbers, ranging within $[-\pi \ldots +\pi]$. The number and meaning of the genes is determined by the applied parameter space: The vector consists of $10 * (1 + 2N)$ Fourier series parameter plus one additional parameter denoting the frequency. The range of frequency was limited to $[0.3 \ldots 10Hz]$.

Episode duration. There were different tests about the episode duration - from $10s$ up to $90s$. Some issues have to be considered when choosing the time: A short episodic time may accelerate the evolution since each episode is finished in a shorter time. On the other side, individuals that tend to be unstable may pass a shorter episode without falling. As a result fast but unstable individuals may be favored by this setup. A long episodic time slows down the evolution. Stable individuals survive over unstable ones. As a good applicable value, an episode duration of $20 - 25s$ has been proved. In general, short episode times may be used for a fast fitness progress in the beginning, while increasing the time may fine-tune and stabilize the individuals.

Abort conditions. To accelerate the evolution progress, episodes may be aborted. Certain abort conditions which are defined by the task module (see section 3.4.4.1) are checked at each simulation step during the episode. If any gets fulfilled,

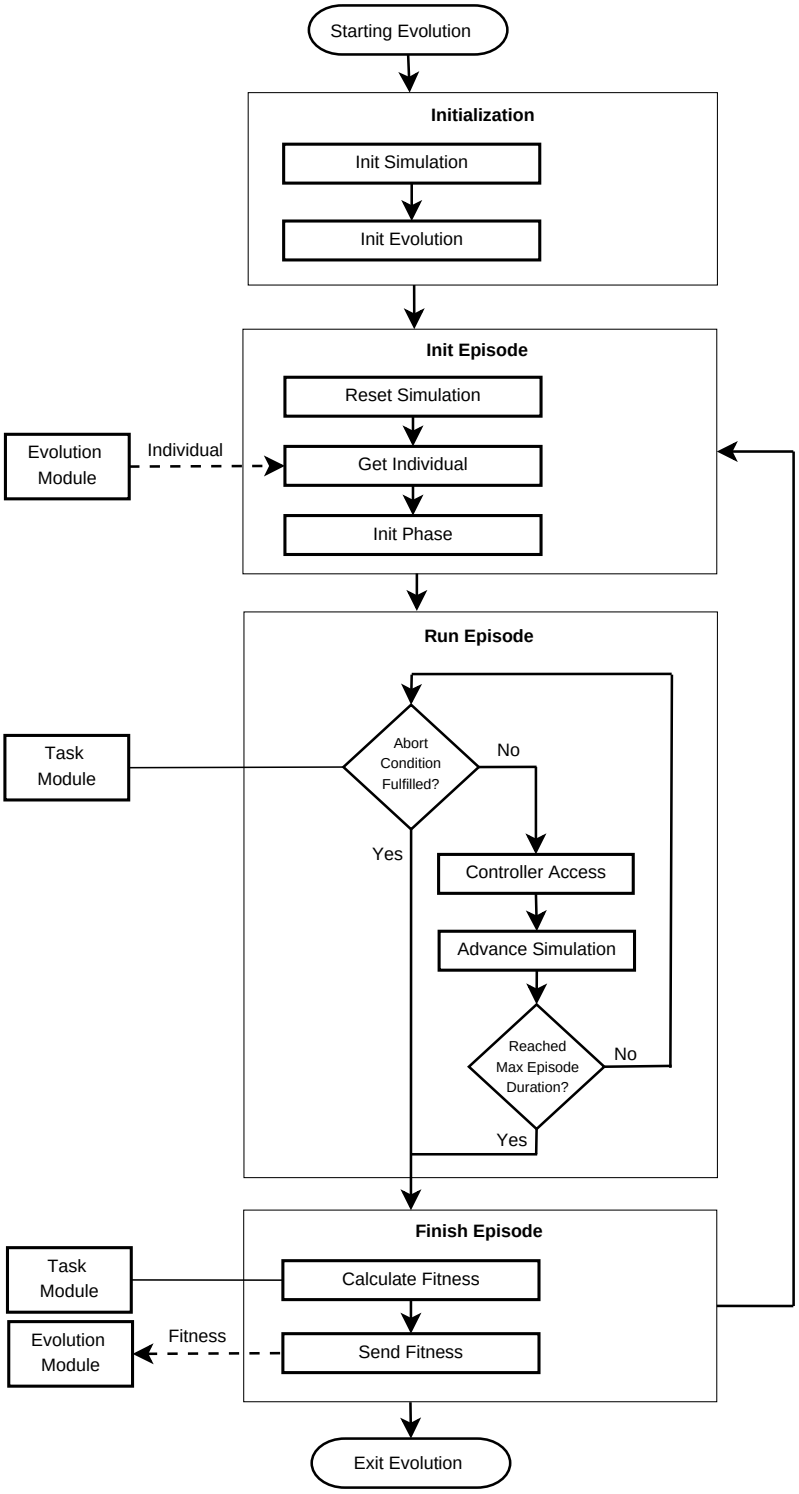

Figure 4.3: Flow chart of an evolution run and illustration of simulation – evolution/task module correlation. The dotted arrows describe the communication between the simulation and the evolution module. Further the two relevant task dependent operations are connected with the task module.

the episode is aborted, advancing to the next episode. For the forward walking task an episode was aborted when the robot fell down, which was detected by the z-coordinate of the hip, or when robot went off the preferred walking line, which was detected by the x/y-coordinates of the hip.

Init phase. Due to the constant term of the partial Fourier series a_{j_0} (see section 4.2), every joint has a constant offset of its trajectory oscillation. This offset leads to a starting pose different to the all-joints-zero-positioned pose. The zero positioned pose is initialized at the beginning of each episode. Without any smooth transition to an individuals starting pose it could fall in the beginning of the episode due of abrupt movements to reach the starting pose. The init phase is a time span in the beginning of each episode, in which the robots pose is interpolated smoothly from the 'zero' position to the starting pose. It was adjusted to $1s$ in all processed evolutions.

Step length. The step length denotes the time increment during one simulation step (see also section 3.2.3). It can be chosen arbitrary, but a larger value will cause growing simulation inaccuracy up to chaotic, nonrealistic behavior. Smaller values increase simulation's accuracy, but slow down the simulation speed linearly. Different tests showed, that in general a step length of $s = 0.02s$ is a good compromise between simulation speed and accuracy. For fine tuning of individuals, step length was set also to $0.01s$.

Fitness function. The fitness function is defined by the task module (see section 3.4.4.1). Within this thesis, the (simplest) value of covered distance in stated walking direction has been proved as best fitness function for evolving forward walking motion patterns. As "position" of the robot, the center of both feet centers was used. An episode is aborted, if the robot falls or gets off the stated walking line within a range of $0.2m$, where as reference point the hip's coordinates were used. A falling is detected, if the robot's hip height gets $0.10m$ below its initial height at episode's start.

4.2.3.1 Bootstrap Evolutions

Figure 4.5 shows the graphs of three evolution runs using the cyclic functions approach with the parameter space of $N = 1$, $N = 2$ and $N = 3$. Several evolution runs with different parameters were processed to find a reasonable parameter set which is listed below. The graphs represent runs with best fitness results. Each run started with a first generation consisting of 50 individuals, each individual with a Gaussian

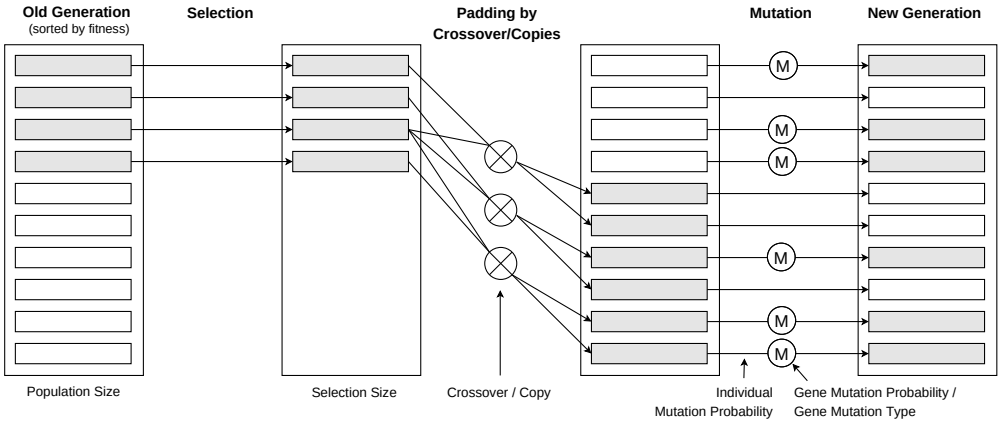

Figure 4.4: Generation of the new generation: Process of selection, crossover/copy and mutation, and the interrelationship of some evolution parameters.

distributed genotype with a mean of $m = 0.0$ and a standard deviation of $s = 0.01$. Table 4.2 gives a review about further evolution parameters:

Parameter	**Value**
Episode Duration	$25s$
Population Size	50
Selection Size	15
Padding Type	Crossover
Mutation Probability for an Individual	0.5
Mutation Probability for a Gene	0.2
Gaussian Variance when Mutating a Gene	0.05

Table 4.2: Summary of most important evolution parameters for the bootstrap evolutions.

As can be seen in in figure 4.5, within the first 20.000 generations, best results were found by $N = 2$ parameter space. It reached a walking speed up to $v = 0.13m/s$, meanwhile for $N = 1$ speeds up to only $v = 0.08m/s$ and for $N = 3$ speeds up to only $v = 0.06m/s$ were reached. This effect can be explained by the two main factors influencing the evolution results: The space of possible motion pattern and the dimension of the search space of parameter vectors. Both depend directly on each other, since a larger motionspace causes a larger search space of parameters. But meanwhile a higher dimension of motion space enables potentially

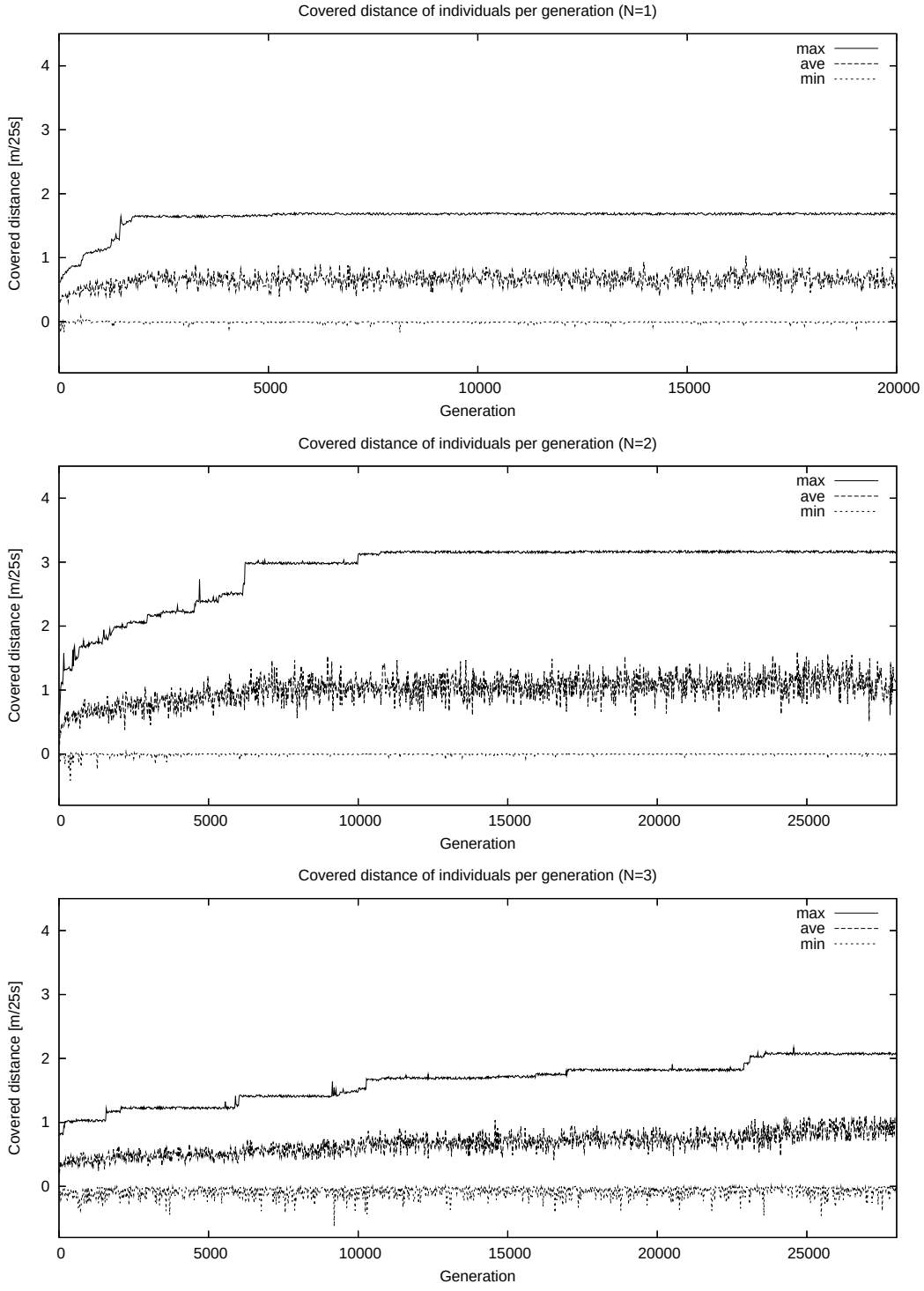

Figure 4.5: Fitness developing of evolution runs for parameter space $N = 1$ (above), $N = 2$ (center) and $N = 3$ (below). Meanwhile the fitness graphs for $N = 1$ and $N = 2$ remain static after the first 12.000 generations, the graph for $N = 3$ shows a small but steady gradient.

faster and more stable individuals, the higher dimension of search space slows down drastically the progress of fitness evolution.

Meanwhile the graph for $N = 3$ shows a small but steady gradient of fitness, the fitness for $N = 1$ and $N = 2$ remains static after the first 12.000 generations. Due to limited computation power, the evolution run for $N = 3$ was aborted without having a stagnated maximum fitness value.

4.2.3.2 Incremental Evolutions

The presented evolution runs in section 4.2.3.1 start with zero-initialized individuals - all genes are gauss distributed with a mean of $m = 0.0$ and a small deviation (detailed parameter see also appendix A). But it is also possible to initialize the first generation individuals at any region in the parameter space. This can accelerate the evolution progress, since the starting point can already have a better fitness value than the zero initialized individuals. On the other hand, this can be used for tuning an already good individual by expanding its motion pattern space.

Since each parameter space of $N > 0$ is a real superset of all parameter spaces of $N', 0 \leq N' < N$, each parameter set of a lower dimensional parameter space can be transferred to a parameter space of higher dimension. All additional parameters are zero initialized. This approach is used for the incremental evolution. An example demonstrates, that already evolved solutions can serve as a starting point for a higher dimensional parameter space evolution: Figure 4.6 shows the results of a $N = 2$ evolution using a $N = 1$ evolved individual and the results on a $N = 3$ evolution using a $N = 2$ evolved individual.

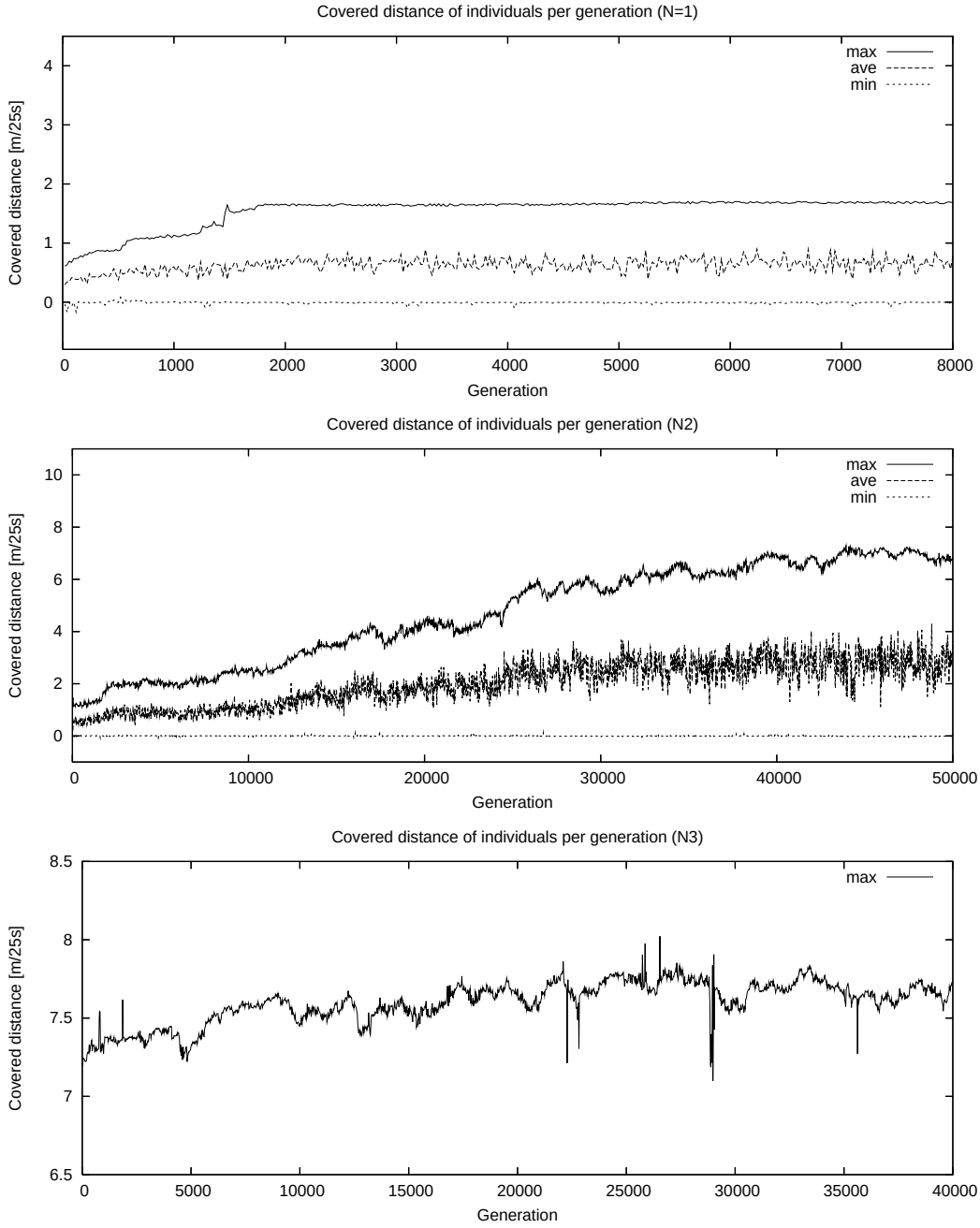

Figure 4.6: Fitness developing of evolution runs applying the ”incremental” evolution: Above: First evolution run, using $N = 1$ parameter space. Center: Evolution run in parameter space $N = 2$. First generation was initialized by final best individual of $N = 1$ evolution run. The covered distance could be enhanced from $1.74m$ to $7.43m$. Below: Evolution run in parameter space $N = 3$. First generation was initialized by final best individual of $N = 2$ evolution run. The covered distance could be enhanced from $7.43m$ to $8.07m$.

4.3 Neural Oscillator Approach

The cyclic function approach (see section 4.2) uses partial Fourier series for generating the wave shapes of the joint trajectories. This enables only harmonic wave shapes, and even for lower parameter spaces this may be a sensitive restriction. To enable more complex wave shapes, the parameter space grows quickly which makes it in turn harder to find optimal parameter vectors. Another disadvantage of the function based approach is the need for a (physical) model to couple available sensor information with the motion generation. The neural oscillator approach makes use of neural networks to generate the joint's target trajectories. This enables other than sinusoidal shapes and due to the synaptic architecture potentially an easier coupling of sensor information.

4.3.1 Two Neuron Networks

The neural oscillator approach generates the core oscillation by using discrete-time dynamics of two neuron networks. Aspects of discrete-time dynamics with recurrent connectivity have been studied extensively, e.g. [10, 38]. The basic idea of this approach is formulated by Pasemann, Hild and Zahedi in [11], which is also a good address for the mathematical background of this approach. The networks consists of two standard additive neurons with recurrent connectivity. In general it is given by a 6-parameter tuple $p = (\theta_1, \omega_{12}, \omega_{11}, \theta_2, \omega_{22}, \omega_{21}) \in \mathbb{R}^6$, where θ_i denotes the bias term of neuron i, and w_{ij} the synaptic weight from neuron j to neuron i. The output of a neuron is in general given by a sigmoidal transfer function σ, which here is chosen to be the hyperbolic tangent $\sigma = tanh$.

Furthermore, for convenience in this approach it is set $\theta_1 = \theta_2 = 0$. Figure 4.7 displays such a network. The resulting two neurons dynamics is then given by the equations:

$$a_1(t+1) := \omega_{11}\sigma(a_1(t)) + \omega_{12}\sigma(a_2(t)) \tag{4.7}$$

$$a_2(t+1) := \omega_{21}\sigma(a_1(t)) + \omega_{22}\sigma(a_2(t)) \tag{4.8}$$

Thus, the network is given by the weight matrix

$$\Omega = \begin{pmatrix} \omega_{11} & \omega_{12} \\ \omega_{21} & \omega_{22} \end{pmatrix} \tag{4.9}$$

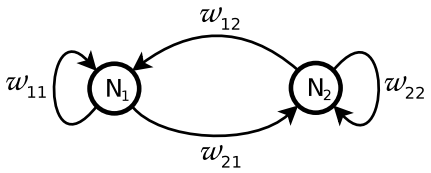

Figure 4.7: A two neuron network with recurrent connectivity and no bias terms.

It is shown, that certain configurations of the weight matrix Ω cause periodic or quasi-periodic attractors in the discrete time dynamics of the network [11, 55]. These types of networks are able to generate different types of oscillations which can be used in turn for generating target trajectories.

4.3.2 SO(2)-Networks

A special type of weight matrices Ω satisfying determinant $det\,\Omega = 1$ are the elements for the *special orthogonal group SO(2)*. They are associated with rotations in the plane and a standard representation of these elements is given in terms of $sin(\varphi)$ and $cos(\varphi)$ of the rotation angle φ. Thus, convenient weight matrices are of the form

$$\Omega = \begin{pmatrix} \omega_{11} & \omega_{12} \\ \omega_{21} & \omega_{22} \end{pmatrix} = \begin{pmatrix} cos(\varphi) & sin(\varphi) \\ -sin(\varphi) & cos(\varphi) \end{pmatrix} \tag{4.10}$$

The dynamics of such a two neuron network depends on one single parameter $-\pi \leq \varphi \leq \pi$ and is given by

$$a_1(t+1) := cos(\varphi)\ tanh(a_1(t)) + sin(\varphi)\ tanh(a_2(t)) \tag{4.11}$$

$$a_2(t+1) := -sin(\varphi)\ tanh(a_1(t)) + cos(\varphi)\ tanh(a_2(t)) \tag{4.12}$$

Networks with weight matrices of this type already have oscillating dynamics. However, since the gradient of the hyperbolic tangent transfer function is always less than one (except for $x = 0$), the amplitude of the oscillation levels off over time. To enable quasi-periodic attractors a second parameter $\alpha > 1$ is introduced which causes a scalable "push" of activation at each time step. The weight matrix of the network is now given by

$$\Omega = \begin{pmatrix} \omega_{11} & \omega_{12} \\ \omega_{21} & \omega_{22} \end{pmatrix} = \alpha \cdot \begin{pmatrix} cos(\varphi) & sin(\varphi) \\ -sin(\varphi) & cos(\varphi) \end{pmatrix} \tag{4.13}$$

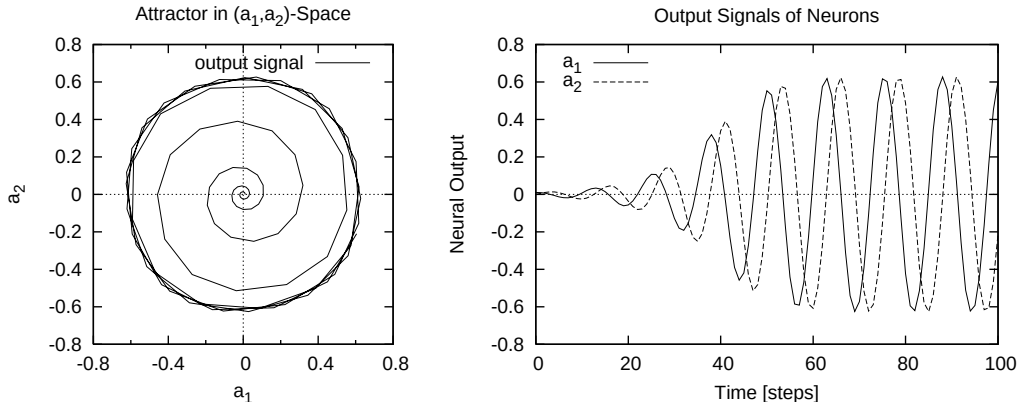

Figure 4.8: Example of a SO(2)-network output: Attractor in (a_1, a_2)-space (left), and output signals of neuron 1 and 2 (right) for $\alpha = 1.1$, $\varphi = 0.5$. Graphs show the initial phase up to reaching the quasi-attractor range within the first 100 time steps. The initial activation was set to $a_1 = 0.01$, $a_2 = 0.0$.

where α in general is chosen to be $\alpha = 1+\varepsilon$ with $0 < \varepsilon \ll 1$. Networks with a weight matrix of the type 4.13 are called *SO(2)-networks*. The frequency of oscillation changes with varying α and φ. Figure 4.8 shows an example of the dynamics of a SO(2)-network with a well-visible quasi-periodic-attractor.

4.3.3 Controller Topology

Section 4.3.2 demonstrates, that certain weight matrices of a two neuron network may generate quasi-periodic oscillations. These are used now for generating the joint's target trajectories. However, there is no reason to limit the weight matrices to the form of SO(2)-networks, since other weight matrices generate "stable" oscillations as well and may vary the attractor's shape. Hence, all four weights are going to be evolved independently in this approach. Each target goal position of a single joint is represented by the output of one standard additive neuron, which derivates its activation by two synapses coming from the two neurons oscillator and a bias term which represents the offset of the joint's oscillation. Thus, each joint trajectory is described by three parameters, ω_{j1}, ω_{j2} and θ_j, where ω_{ji} denotes the synaptic weight coming from neuron $i = 1, 2$ and θ_j the bias of joint j. Figure 4.9 shows the neural topology of the controllers network.

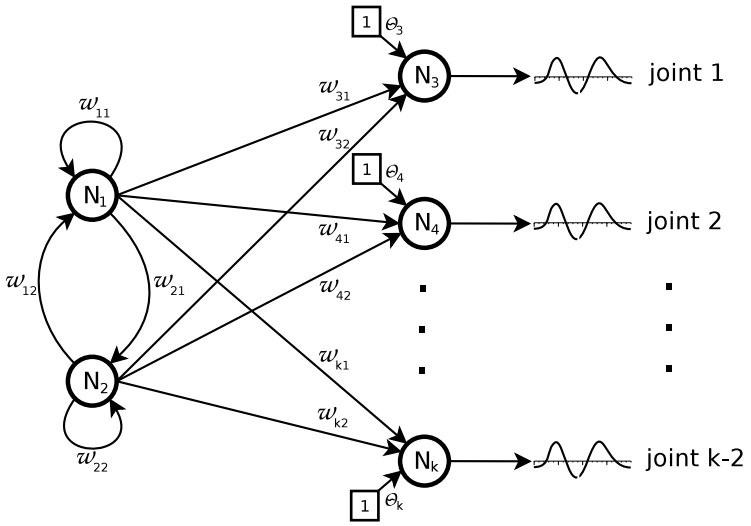

Figure 4.9: Topology of the neural net controller. Each joint's trajectory is given by a dedicated neuron, which derives its activation by the two oscillating neurons N_1, N_2 and a bias term θ_j.

4.3.4 Symmetry Assumption

Analog to the symmetry assumption in the cyclic function approach (see section 4.2.2), in the neural net approach some reasonable restrictions due to symmetric behavior may reduce the parameter search space significantly.

The phase shift of π between right and left sided joints is easily done by mirroring all target trajectories of the joints belonging to one body side. Consider two corresponded joints j (left side) and k (right side). The offset θ_j and θ_k works for both in same direction, thus

$$\theta_k = \theta_j \tag{4.14}$$

The trajectory's amplitude is mirrored along the x-axis, which corresponds to a 180° rotation about the origin in the (a_1, a_2)-space (compare figure 4.8, left). Hence it is set

$$\omega_{ki} = -\omega_{ji}, \quad i = 1, 2 \tag{4.15}$$

In this way, the parameter space is nearly halfed from 61 to 34 parameters.

4.3.5 Evolution Setup and Results

Similar to the evolution setup of the cyclic function approach, the primary object was to cover a long as possible distance within a given time. The fitness values of the individuals were determined again by processing episodes. To compare the results with those of the cyclic function approach, same basic setup and episodic process of the evolution (e.g. with initial phase) was applied - see section 4.2.3. The different or rather additional setups are described below.

Genotype description. A single individual is given by a double number parameter vector, denoting the synaptic weights and bias terms of the controllers network. A single weight ranges within $[-4.0 \ldots + 4.0]$, since this covers the most significant operating range of the hyperbolic tangens transfer function. The first generation consisted of individuals with (weak Gaussian noised) zero initialized parameter vector except the first four parameters denoting the neural oscillator's weights: For reasons of accelerating the evolution progress, the two neuron net was initialized so, that it already had oscillating dynamics. The chosen parameter were set to: $\omega_{11} = 1.1$, $\omega_{12} = 0.7$, $\omega_{21} = -0.7$, $\omega_{22} = 1.1$, which corresponds to a SO(2)-network with $\alpha \approx 1.3$ and $\varphi \approx 32°$.

Step length. Unlike to the cyclic function approach, in general best evolution results were found at step length $s = 0.01$, fine tuned individuals require also step lengths up to $s = 0.005s$. This may be caused by the non-harmonic target trajectories, which require a more precisely sample resolution.

Net update frequency. In addition to the simulated world's time, in the neural oscillator approach the neural net has also to be updated periodically. This could be either done by denoting the ratio between simulation's updates and net updates. Or the net update frequency is given absolute in world's simulation time, which has the advantage that changing of one parameter does not interact the other - e.g. the simulation resolution is independent from net update frequency. In the latter case, it is advisable to chose simulation's step length as a multiple of the net update time to ensure always same simulation time period between two net updates. In this approach, the net update frequency is set in simulation time. Best behavior and evolution results were found for $f_{netupdate} = 20Hz$ (or rather one net update per $0.05s$).

The further most important evolution parameters are listed in table 4.3. Figure 4.10 shows the graphs of the evolutions with best results.

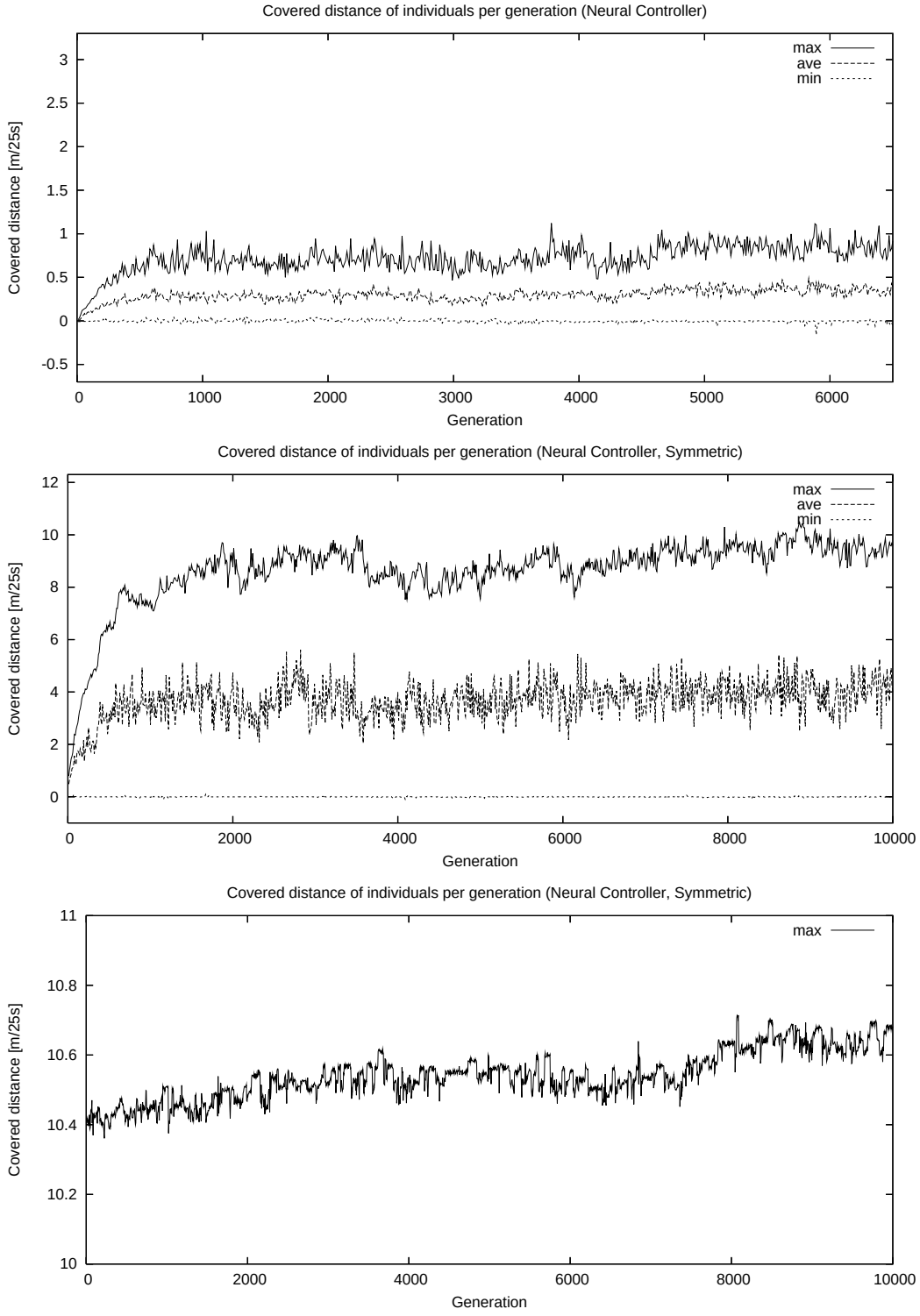

Figure 4.10: Fitness developing of evolution runs using the neural oscillator approach. Above: Evolution without symmetry assumption. Center: Evolution with symmetry assumption. Below: ”Fine tuning” of best individual with symmetry assumption.

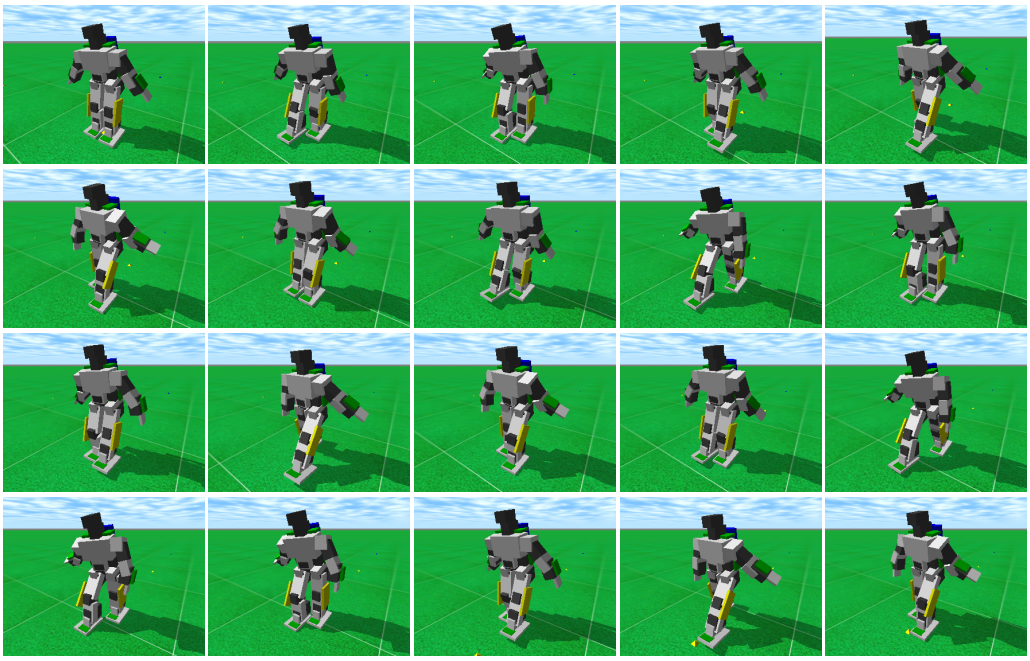

Figure 4.11: Evolution of Walking Pattern: Example of evolved walking pattern with neural oscillator approach. Pictures show start of walking and first steps. The displayed motion reaches a walking speed of about $0.45 m/s$.

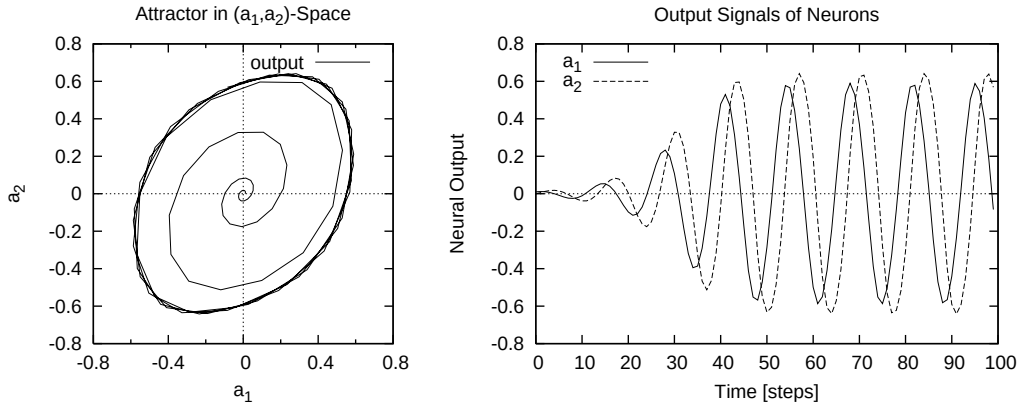

Figure 4.12: Dynamics of the displayed individual in figure 4.11. Attractor in (a_1, a_2)-space (left), and output signals of neuron 1 and 2 (right). Evolved synaptic weights of the neural oscillator: $\omega_{11} = 1.166865$, $\omega_{12} = 0.610873$, $\omega_{21} = -0.467230$, $\omega_{22} = 0.834088$. Graphs show the initial phase up to reaching the quasi-attractor range within the first 100 time steps. The initial activation is set to $a_1 = 0.01$, $a_2 = 0.0$.

Parameter	Value
Episode Duration	25s
Population Size	50
Selection Size	15
Padding Type	Copy
Mutation Probability for an Individual[3]	0.9
Mutation Probability for a Gene	0.1
Gaussian Variance when Mutating a Gene	0.01

Table 4.3: Summary of most important evolution parameters for the neural oscillator evolutions. [3]Note that best evolution results were found without(!) crossover, thus mutation rate was increased to 0.9

4.3.6 Sensor Coupling

Without any feedback, a robot pilots ”blind” through its environment. The robot could not react to obstacles, surface changings or gradients. Thus, stable and adaptive biped motions are not possible without any environmental feedback. The human as archetype of biped motion uses a lot of sensor informations to control its motion.

Due to limited time and scope of this thesis, sensor coupling experiments were only processed briefly within some problems of the students laboratory [33]. Further studies, especially for stabilizing biped motions were not accomplished. Nevertheless, some ideas, which belong to the background of exploring the neural oscillator approach, shall be outlined here.

While dealing with sensor feedback processing to solve a certain problem (e.g. stabilizing walking), two major questions get posed: Firstly, what kind of sensor information may be useful regarding the problem, and secondly, how should a certain sensor information be integrated into the motion generation to solve the problem? Both questions refer to biological, physical and mechatronic topics, which form an own field of research.

The synaptic architecture of the neural oscillator approach allows to connect any sensor information to any neurons of the controller’s neural net without having a specific sensor coupling model. The evolution may help finding optimal synaptic weights, to solve or optimize the given problem. The following two sections give

brief examples, how sensor information may be incorporated into the neural oscillator controller.

4.3.6.1 Harmonic Synchronization

Figure 4.13 shows a simple topology of coupling an external harmonic clock oscillator to the presented two neuron network. Within a certain range of frequency the neural oscillator is able to synchronize with the external oscillator. Figure 4.14 demonstrates an example: The external oscillator generates a sinusoidal shape with frequency of 5 periods per 100 time steps. The synaptic weight of coupling the oscillator was chosen to $\omega_{1s} = 0.2$. Without any coupling the neural net oscillates with about 8 periods per 100 time steps. When coupling, the neural oscillator synchronizes smoothly within 2 periods of the external oscillator to exact the same frequency.

Harmonic oscillating sensory sources are on hand when e.g. filtering body's acceleration sensors, that measure accelerations in the horizontal x/y-plane. Thus, it is conceivable using this type of sensor coupling to synchronize the neural oscillator with the robot's pendulum frequency.

4.3.6.2 Impulse Synchronization

Figure 4.15 shows a simple topology of coupling an external impulse generator to the two neuron network. With a certain setup of the weight matrix of the neural network, it is capable to synchronize with impulses given by the generator. Figure 4.16 demonstrates an example: The external generator induces line peaks of size 1.0. The frequency is chosen irregularly to demonstrate the stable synchronization to different frequencies. The weight matrix corresponds to a SO(2)-network with $\alpha = 0.9$. Thus, the neural oscillator's activation amplitude is damped over time without external impulses.

Discrete impulses are given e.g. by a discrete touch sensor placed on the foot, which generates an impulse when the foot touches down on the floor. A sensor coupling in that way could synchronize the neural oscillator with the foot's touching which may help stabilizing the motion on slightly rough floors.

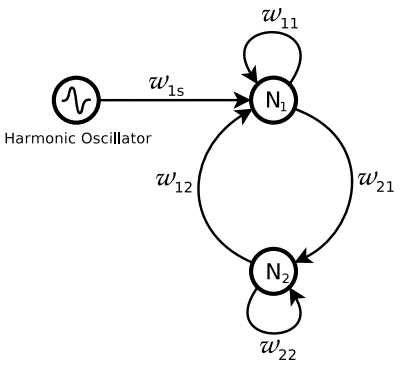

Figure 4.13: A two neuron network with a coupled external harmonic oscillator.

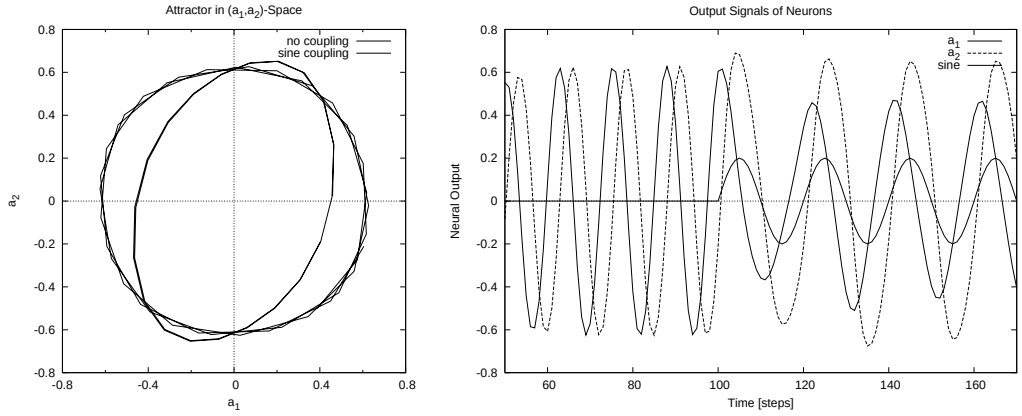

Figure 4.14: Example of a harmonic synchronization: Left: Attractor in (a_1, a_2)-space of the two neuron oscillator without and with coupling. Right: Output of the two neuron oscillator without and with coupling. The SO(2) oscillator parameters are: $\alpha = 1.1$, $\varphi = 0.5$, the chosen synaptic coupling of the external oscillator is $\omega_{1s} = 0.2$. The external oscillator generates a sinusoidal shape with frequency of 5 periods per 100 time steps. After coupling the two neuron oscillator synchronizes from 8 periods per 100 time steps to the external oscillator's frequency.

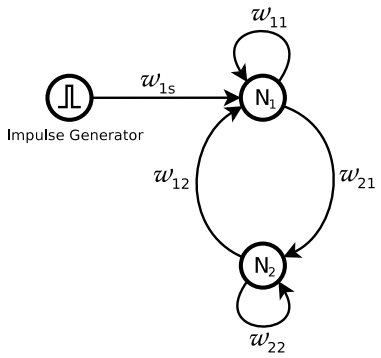

Figure 4.15: A two neuron network with a coupled external impulse generator.

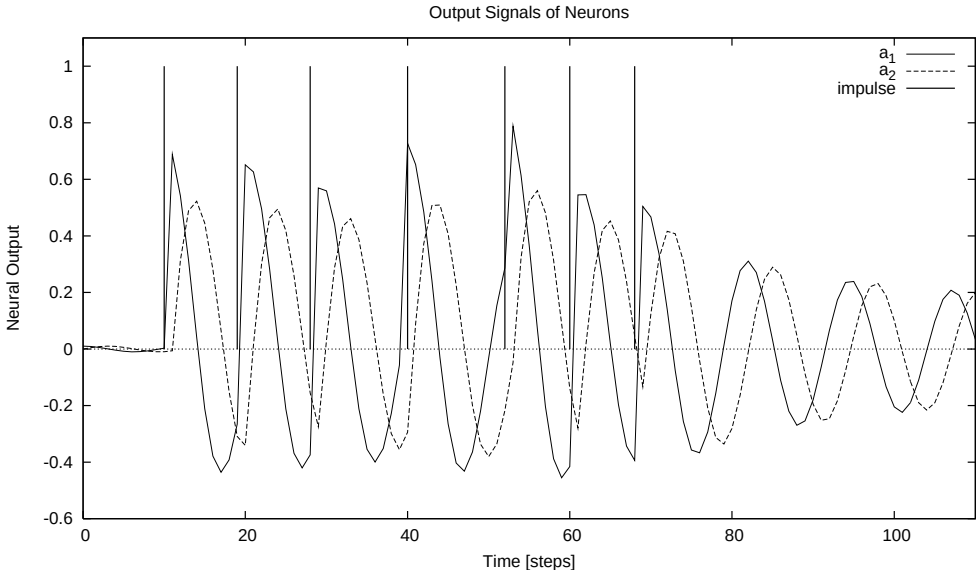

Figure 4.16: Example of an impulse synchronization: Output of the two neuron oscillator. The vertical lines indicate the impulses. The SO(2) oscillator parameters are: $\alpha = 0.9$, $\varphi = 0.5$, the chosen synaptic coupling of the external oscillator is set to $\omega_{1s} = 1.0$. The neural oscillator synchronizes with the irregularly clocked impulses of amplitude 1.0.

4.4 Comparing Joint Trajectories

Figure 4.17 displays joint trajectories of walking motions of an evolved N2-individual, an evolved neural oscillator individual and a human. The graphs display the trajectories of the right ankle, right knee and right hip joint in sagittal direction, which give a first well characterization of each motion.

The graphs of the Simloid controller driven motions show the target and the real joint trajectories. The motion data of the human were measured on an instrumented treadmill at the Locomotion Laboratory of University of Jena[3]. Regarding the body height relation between human (shoulder height $\approx$ 150cm) and Bioloid (shoulder height $\approx$ 34cm), the real walking speed of the human ($v_{walk} \approx 1.50m/s$) corresponds to a walking speed of $v_{walk} \approx 0.34m/s$. All graph intervals were bring into line.

The fullstep frequencies range from 0.67Hz (N2) over 1.14Hz (Human) up to 1.33Hz (Neural Oscillator). The low frequency of the N2 individual is compensated by the step width. This can be seen in the huge amplitude of the hip joint. Table 4.4 gives a brief analysis about offset and amplitude of the drawn trajectories.

Although not further funded, the table draws the percental relation between hip, knee and ankle amplitude. This value may characterize the relative use of the single joints. It is noticeable, that a human motion makes most of use of the knee joint (relative use of 46%, 65° working range at human in contrast to 30° working range at neural oscillator). The working range of the human knee joint with 65° is nearly twice the working range of the human hip joint. In contrast, the evolved individuals make more of use of the hip and the ankle joint (more than 60° hip joint's working range of neural oscillator in contrast to about 25° working range at human), meanwhile the knee joint's trajectory is characterized by lower amplitudes. The most extreme case is given in the N2 individual, which moves with nearly a stiff knees (relative amplitude 7%). This leads to somehow "stilt" walking individuals. These types of individuals make extensively use of the inverted pendulum frequency of the robot's body, which is a typical motion phenotype of this simulation's evolved individuals. Apparently for the applied evolution in the "ideal" world with constant influences and an obstacle free floor it is easier to cover longest distances by moving in this way. The disadvantage of this moving type is, that small obstacles on the

[3]Thanks to Susanne Lipfert and the Locomotion Laboratory (University Jena, Germany) [57]

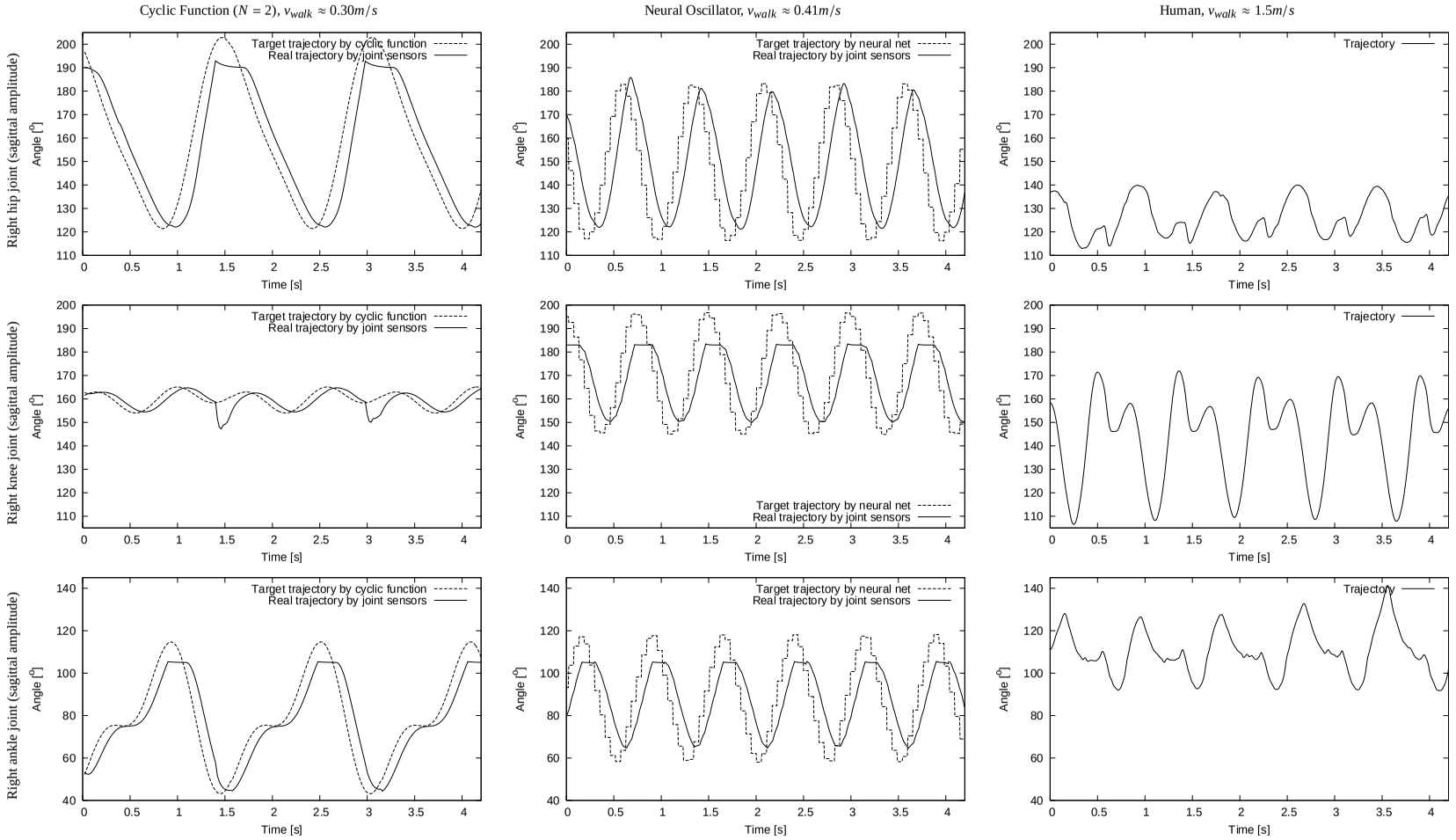

Figure 4.17: Comparison of joint trajectories for hip (upper row), knee (center row) and ankle (lower row) joint of the right leg. Graphs are taken from a N2 individual (left column), neural oscillator individual (center column) and human (right column).

Trajectory		N2		Neural Controller		Human	
Hip	Offset	135°		155.5°		132.5°	
	Amplitude	70°	(50%)	65°	(47%)	35°	(25%)
Knee	Offset	160°		166.5°		137.5°	
	Amplitude	10°	(7%)	33°	(24%)	65°	(46%)
Ankle	Offset	75°		85°		110°	
	Amplitude	60°	(43%)	40°	(29%)	40°	(29%)

Table 4.4: Overview about the offset and amplitude properties of the trajectory graphs shown in figure 4.17. The percentage is related to the sum of hip, knee and ankle amplitude and gives a (weak) characterization, how the joints are used during walking motion.

floor or small changes on the robot's body causing interferences and/or changes in the body's pendulum frequency will sensitively disequilibrate the walking motion.

Note that target trajectory of the neural oscillator differs partly a lot from the real trajectories in the case of knee and ankle joint. This is due to the joint's working range limits and leads to joint motions up to the joint stops, which are visible in the chopped off peaks. In general such a behavior should be avoided for reasons of mechanical stress.

Further differences can be seen in the shape of the trajectories: The human walking motion is characterized by two trajectory's peaks at the knee and ankle joint per one full step - one in the support phase, where the corresponding leg is carrying the body, and one in the swing phase, where the leg is repositioned to the start point of the support phase. Both bending motions are used for bring the foot into line with the floor while keeping the hip's height more or less constantly. Particulary a similar shape can be observed at the N2 individual (compare N2 ankle joint trajectory). The neural oscillator is due to its topology not capable to generate more complex shapes than single amplitude ones.

4.5 Short Excursion: Other Tasks

As mentioned in section 3.4.4.1, the presented simulation environment allows to plug easily different task modules to the evolution setup. This enables evolving different behaviors, without touching the underlying motion generation structure.

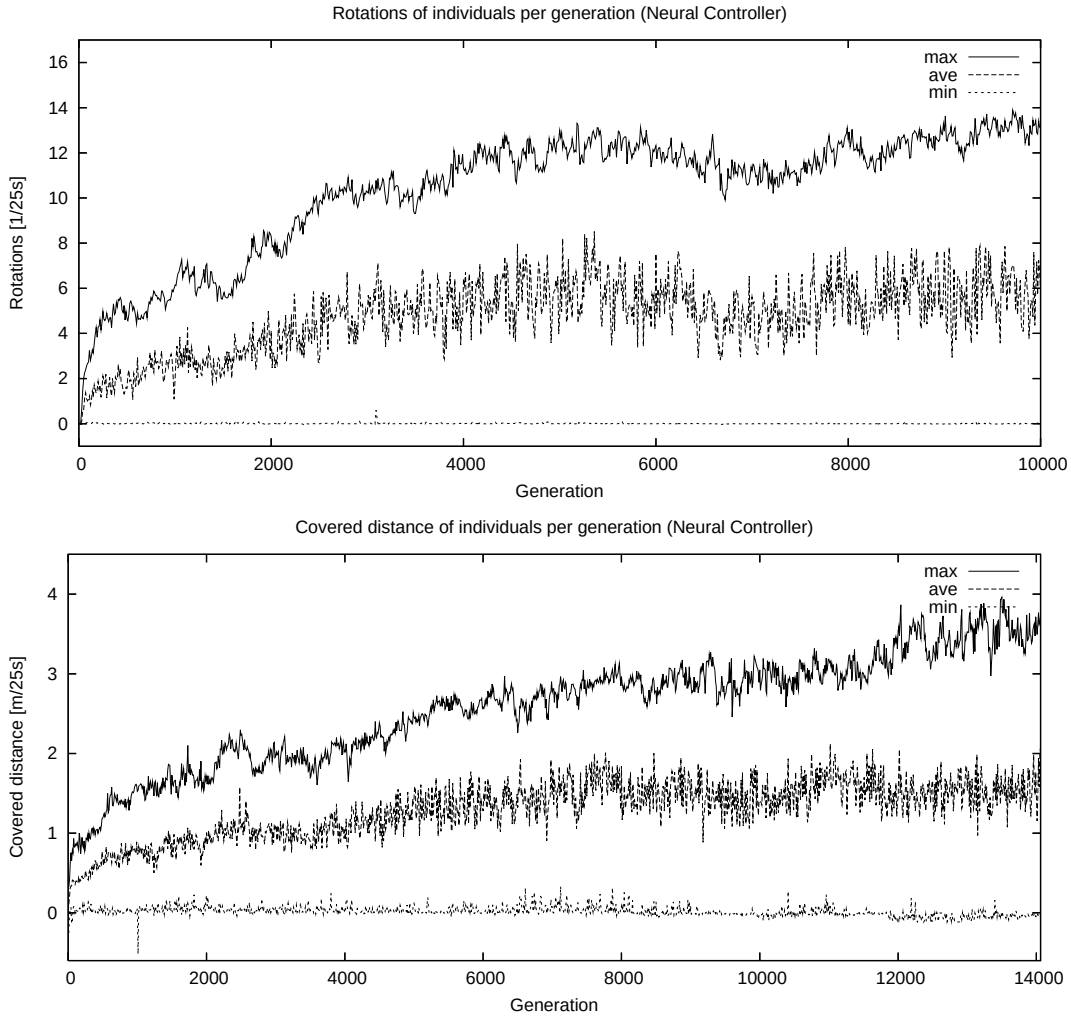

Figure 4.18: Fitness developing of evolution runs applying turning (above) and rightwards walking (below) task. The underlying motion generation is a neural oscillator architecture, without symmetry reductions.

As instance, two examples demonstrate the evolving of alternative tasks. As motion generation approach the neural oscillator is applied. Since the movements solving these tasks in general do not have symmetric characteristics between left and right sided joints, no symmetry assumption was applied.

Figure 4.18 (above) shows the fitness development of a turning task: The fitness value is given by the covered turning angle of an individual during one episode of 25s. The graph displays the number of complete rotations. An episode is aborted, if the individual falls (hip height 0.10m below the initial value), or if it gets off of its

initial position (hip has to be within a $0.20m$ circle around its initial position). As to see, within 12.000 generations individuals are able to turn up to more than 13 turns around its z-axis in one episode.

Figure 4.18 (below) shows the fitness development of a sidewards walking task: The task is defined analog to the forward walking task, except that the stated walking direction was set to rightwards. Thus, the fitness value is given by the covered distance along the right side axis (center of both feet centers). Within 14.000 generations, individuals covered a distance up to $4m$ in one episode.

Chapter 5

Knowledge Transfer to Real Hardware

The overall long-term objective of this work is to enhance the real robot's motion behavior. This includes questions about appropriate controlling and motion generation architectures as well as the retrieval and optimizing of motion pattern parameters. This chapter deals with the question, how the simulation and its results may help enhancing real robot's motions.

5.1 Types of Knowledge

The purpose of the simulation and its experiments is the acquisition of new knowledge. In literature, different types of knowledge are introduced [40]. The definitions of the individual knowledge types refer to human knowledge and due to their definition base it would be somehow arbitrary to apply this classification into the field of robotics. Nevertheless, with a brief statement it is reasonable to classify the knowledge outcome of the simulation experiments into different levels, because each level has a different transfer quality. Regarding this work's simulation, basically three levels of knowledge get touched: Knowledge about (concrete) motion parameter sets, about (potentially appropriate) controlling architectures and about (potentially appropriate) experimental setups.

A motion parameter set describes explicitly a certain motion pattern. This type is both the simplest and most concrete knowledge of all three levels. The transfer qualification of this knowledge type requires highest congruence between simula-

tion and real world behavior. Given a real world matching simulation, each evolved individual corresponds to knowledge about a possible solution in real world. In a slightly more abstract manner, it specifies the relation between the parameters and the shapes of the joint's trajectories.

A more abstract level of knowledge is given by knowledge about a task-applicable controlling architecture. It describes a general structure that is potentially able to accomplish a certain task, without giving a certain instance (parameter set) of the structure that performs the task. Within a given control architecture, several solutions may be found, defined by an individual parameter set. Its transfer quality requires less congruence between simulation and real world behavior, since the control architecture should be independent of a concrete robot behavior instance.

The most abstract level regarding the simulation's outcome is the knowledge about potentially appropriate experimental setups, that could generate new (lower level) knowledge. In literature this type of knowledge is also known as *meta-knowledge* [40] - which includes among other knowledge about how to gather new knowledge. Similar to the controlling-architectures, its transfer quality requires less congruence between simulation and real world behavior, since the experimental setup should be indepent of a concrete robot/world behavior instance.

5.2 Motion Export

Up to the time of the real robot experiments, the implemented controlling framework of the real robot only supported the keyframe-transition technique. However, to transfer and test any trajectory driven motion of the simulation, the target trajectories can be recorded into a keyframe-transition structure. This is done by the *keyframe exporter* (see figure 5.1) which is connected to the PID controllers of the joints. The exporter is able to linearly record a motion behavior, within a certain time span and with a configurable time resolution. Further it is able to extract a period (e.g. full step period) of a motion pattern in order to generate a cyclic keyframe-transition structure.

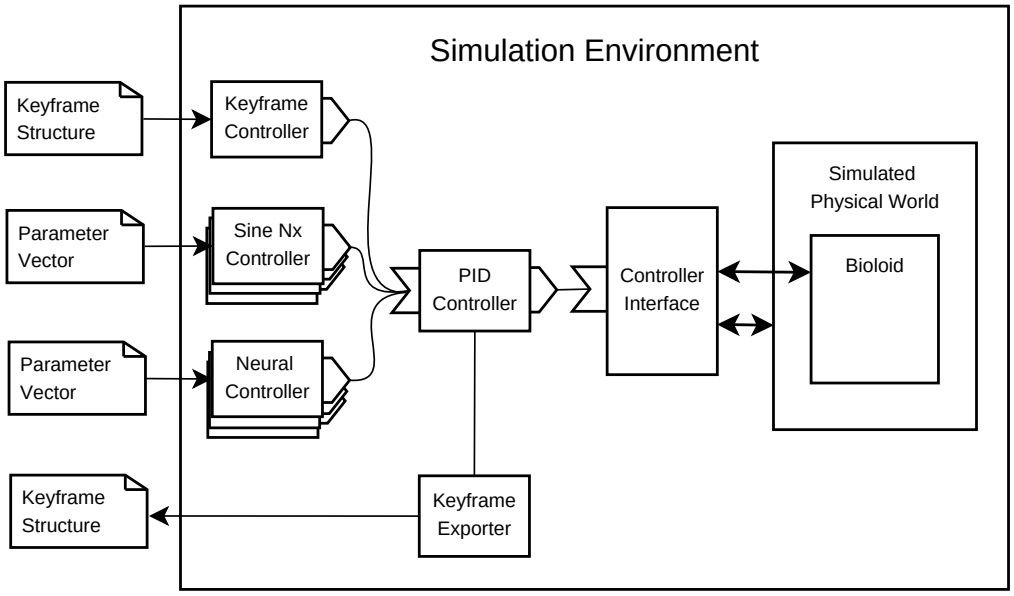

Figure 5.1: Exporting of simulation results: The keyframe exporter module tracks all target joint trajectories during a motion and records correspondend keyframes at a configurable time resolution.

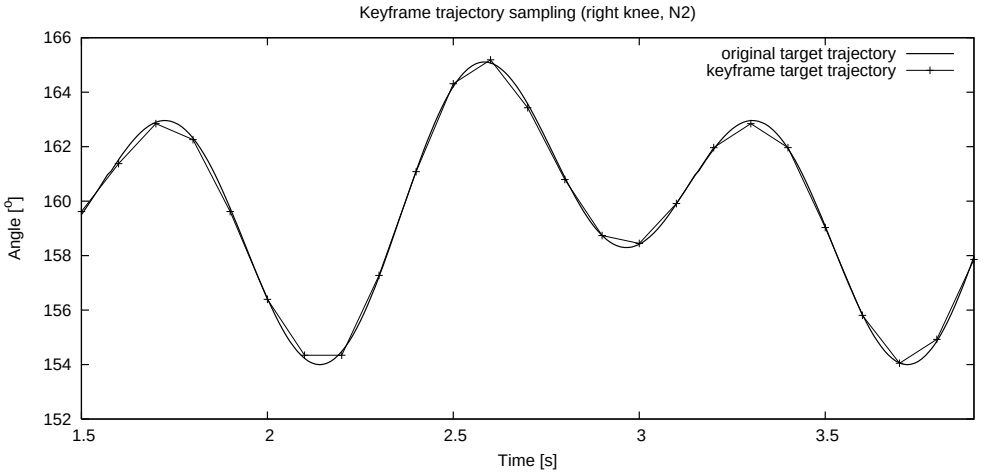

Figure 5.2: Example of sampling a N2 knee's target trajectory with a samling frequency of $10Hz$. Note that the target angles of the keyframes are quantized to the AX-12 servo motor's control graduation (approx. 0.29°).

5.3 Experiments and Results

Within the real robot experiments, motion behaviors generated by both, the cyclic function controller and the neural oscillator controller were transfered and tested at different time resolutions ($0.02s$...$0.1s$). To cut a long story short, none of the

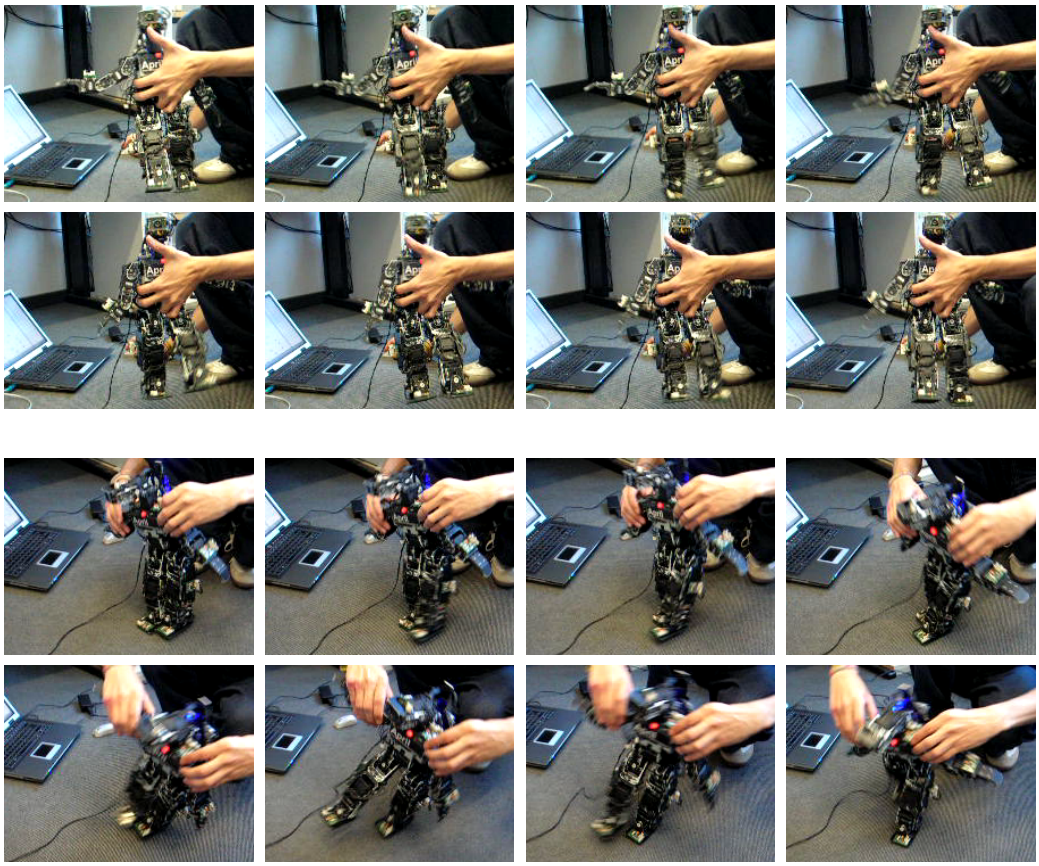

Figure 5.3: Transfer of motion pattern to hardware: The generated motion on the real robot is similar to the simulated one, as long as it acts free (above). The 'grounded' real robot needs manual help in contrast to its simulated counterpart (below).

transfered motions could generate a robust walking pattern. In general the robot was able to reproduce all transfered motions – as long as the robot moves freely (hold detached from ground), the "visible" characteristic is quite similar to the simulated one. Touching the ground changes the phenotype of the motion a lot.

We identified two major problems, causing serious gaps between simulation and real world behavior: One refers to the grave gears tolerances and the other to the non-linear motion characteristics of the servo motors. Further differences could be observed in the stiffness of the robot parts, in particular of the legs, and in the floor friction properties.

Manual modifications could slightly enhance the real robot motions: Motions, where feet move closely to one another tend to clink together due to the tolerances

and the robot begins to stumble, could be 'repaired' by slightly adjusting the offset of the hip joint trajectories.

5.4 Discussion

The transfer experiments showed, that the present version of simulation do not match real world behavior sufficiently for transfer (raw) parameters set in order to reproduce evolution results on the real robot. The existing gap is mainly caused by the simplified modeling of the robot, in particular the modeling of the servo motor joints. To exemplify the problem of the tolerances in general: It is not even possible to pose the robot stably standing on one foot with all joints fixed at a certain position. Thus, the underlying target system has some issues which make it potentially difficult to generate a robust biped walking motion.

A more promising way of knowledge transfer is given at the knowledge about controlling architectures: The simulation showed, that the proposed controllers are potentially appropriate for biped walking generation on real robots. Actually, a controlling framework allowing to apply the neural oscillator approach is going to be implemented on real robot.

The successfully applied experimental setups and evolution paramters constitute the (procedural) meta-knowledge about how to potentially obtain parameter solutions. This knowledge may help finding appropriate experimental setups for real hardware experiments. Furthermore, with a given similarity between real and simulated world, real world experiments could start evolution runs at already evolved (optimized) individuals of the simulation run and in that way shorten hardware experiments.

For enhancing the transfer results, two points should be worked out:

- Enhancing of the modeling, in particular the servo motor joints. This includes the question for an appropriate model of the servo motor's behavior, as well as appropriate experiments, to determine all relevant parameters of the model.

- Enhancing of the evolution results. The processed tasks of the evolution reward only the covered distance of an individual's run. After generating fast walking motions, the motions should now be stabilized by either enhancing the fitness function by additional stability criteria (e.g. by utilizing sensory

data) and/or by providing ”harder” environments to complicate the given problem (e.g. by rough floors, impacts, etc.).

Chapter 6

Conclusions

The subject of biped motion is a wide complex topic, that could not be handled in one single thesis. Within a small application range, this thesis exemplifies the use of physical simulation to study controller models. This section summarizes and discusses the work and gives a review about possible continuing topics that could carry on the processed work.

6.1 Summary and Discussion

Physical simulation is a promising method, to study and explore real world problems of complex biped robots. Especially when applying extensive machine learning or evolutionary algorithms it can save hardware stress, it may provide exhaustive state informations, enables parallelizing of experiments and permits unsupervised experiments in multiple realtime. Condition precedent to reusable results is an accurate modeling and continuous comparing between simulation and real world behavior.

The thesis on hand exemplifies a possible way of exploring biped motion generation using physical simulation. It presents a software environment, which simulates the introduced target system - the 19 DoF Bioloid robot. The modular environment is designed for exploring different motion generation approaches, as well as for applying artificial evolution to solve or optimize given motion problems.

Due to the scope of this thesis, the model of the robot used in the simulation was kept in a quite simplified level compared to the complexity of the real robot. However, the simulation is capable to reproduce real worlds behavior, which is shown

by the transfer of real robot motions to the simulated one with similar results (see section 3.4.3.3).

The thesis introduces two approaches of motion generation, that could be successfully applied for the simulated biped robot. Within the simulation environment the approaches were implemented as controller modules. It is shown, that both are applicable to generate walking patterns for the simulated biped robot. The work demonstrates, how the parameters for the given robot may be found and optimized with the appliance of evolutionary algorithms. Furthermore, the setup of the artificial evolution is able to optimize the parameters for other tasks than walking as well. Some results of the processed evolutions are summarized at the ”Simloid” website [7].

The implemented controllers were capable to generate a robust biped walking with a comparable high walking speed up to $0.51 m/s$. They do not use any sensory information. Particularly the properties and advantages of the neural oscillator approach are outlined and it is shown, how the neural oscillators may synchronize with sensory information.

In laboratory experiments, some evolved motions were transfered to the real robot. Due to divergence between simulated and real world behavior, none of the transfered motions could generate a robust biped walking pattern on the real robot. Nevertheless, it is outlined, how simulation may enhance real robot motions. In general, the presented approaches may be applied to any biped robot with the possibility of trajectory driven joints.

As another application, the simulation environment was used in a students laboratory [33]. The software allows to control the simulation by an external program. The applied communication protocol allows to use any programming language supporting this protocol to control the simulation. Within the laboratory, students had to solve different isolated tasks, e.g. balancing or stabilizing of the robot. Moreover, the implemented communication protocol allows to switch between the target trajectory driven approach and the torque driven approach to control the joints.

6.2 Outlook

The presented work constitutes only a first step, using simulation to explore and optimize different controller models of the Bioloid robot. There were several points,

which either touch ”unknown territory” or have to be improved for having a better outcome. Some of the most important points are reviewed in the following. Mainly they can be categorized into conceptual and technical matters.

6.2.1 Conceptual

Exploring Sensor Feedback. As mentioned above, the presented walking controllers do not incorporate any sensory information. Sensor feedback is essential for reacting on any environmental information. This topic includes for one thing the exploring of appropriate sensors (e.g. touch or acceleration sensors) and for another thing how to incorporate the sensor information into the motion generation. Section 4.3.6 proposes and exemplifies two concepts, how sensory data may be incorporated into the neural oscillator controller approach.

Stabilizing Individuals. The evolution setup and chosen fitness function is designed for getting fast individuals. More important than fast individuals – even for the gap between simulation and real world – would be to get and explore robust walking pattern. This means individuals, which are insensitive to environmental effects, such as roughness of the floor or small impacts caused e.g. by collisions with other robots. This could be done by enhancing episodes to be processed: A randomized environment with small obstacles, holes or roughness may help finding robust motion pattern. Sensor feedback would be essential as well as enhancing the fitness function, e.g. by adding additional sensor data based terms.

Exploring Alternative Neural Net Architectures. This work demonstrates, that even simple neural nets are powerful for generating biped walking motion pattern. However, the presented architecture is limited to single frequency oscillations. Analog to the Fourier series approach, the neural net topology may be enhanced enabling more complex wave shapes. As shown in figure 4.17, human walking trajectories are described by terms of at least double frequency, whereas the presented neural topology only enables single amplitude shapes.

6.2.2 Technical

Distributed Evolution. The presented simulation environment is single threaded. This means, one evolution run is limited to one process. Artificial evolution is good for parallelizing, since each episode is indepent from all others within one genera-

tion. Distributing the evolution among several processes will accelerate the whole run linearly with the number of processes, as long as the number of processes do not exceed the number of individuals per generation.

Modeling ODE Tolerance Joints. As already stated, one major reason of the gap between simulation and real world is the missing of tolerance effects of the servo motors in the simulation. Unlike simulation, real servo motors have tolerances mainly caused by the gears inside of each servo. The effect of the tolerance grows with leverage that effect a servo motor. Thus, the movement of e.g. the ankle joints is strongly influenced by these tolerances. Elaborating an appropriate model, describing all servo motors characteristics may enhance and validate the physical simulation a lot.

Scene Description Language Support. At present, the whole scene, including the environment as well as the robot, is described within the program source code (Scene Description Module, see section 3.4.2). Changing the environment's parameter or the setup of the robot requires a source code modification and a recompilation. From point of conceptual view, the scene description should be separated from program source code. This could be done by implementing a parser, supporting a scene description language. As reference, see [6].

6.3 Acknowledgments

The author would like to thank Hans-Dieter Burkhard, who enabled to write this thesis within its artificial intelligence laboratory. Special thanks to Manfred Hild for supervising me and my work, and for all the beneficial brainstorming during many coffee breaks. More and further thanks to Ralf Berger for a lot of helpful advises and the nice teamwork within our simulation team. Special thanks to Benjamin Werner who arranged and performed the real hardware experiments. Christian Thiele wrote the ”Motion Editor”, which was applied for the real robot experiments. The members of the Humanoid Team Humboldt assembled, maintained and enhanced the Bioloid robot kits. Finally I am grateful for my family – my parents, Jana and my little Karla for supporting me during all the work.

Appendix A

Evolution Setups and Parameters

Evolution	Fig. 4.5, above	Fig. 4.5, center	Fig 4.5, below
Controller Type	Cyclic Functions, symmetric		
	$N = 1$	$N = 2$	$N = 3$
Task	Forward (fitness=distance)		
#Parameters	31	51	71
Parameter Range	$-\pi \ldots + \pi$		
Step Length	$0.02s$		
Episode Duration	$25s$		
Initialization	Gaussian distributed, $mean = 0.0$, $\sigma = 0.01$		
Population Size	50		
Selection Size	15		
Crossover/Copy	Crossover		
Individual Mutation Probability	0.5		
Gene Mutation Probability	0.2		
Gene Mutation Type	Gaussian distributed, $\sigma = 0.05$		
Start Fitness	0.0	0.0	0.0
Best Fitness (Generation)	1.74 (7,928)	3.18 (20,970)	2.33 (23,551)
Processed Generations	20,272	55,066	24,800
Calculation Time	$8d\ 18h$	$20d\ 15h$	$11d\ 8h$

Table A.1: Parameters for bootstrap evolutions with cyclic function controller (Figure 4.5).

Evolution	Fig 4.6, center	Fig 4.6, below
Controller Type	Cyclic Functions, symmetric	
	$N = 2$	$N = 3$
Task	Forward (fitness=distance)	
#Parameters	51	71
Parameter Range	$-\pi \ldots + \pi$	
Step Length	$0.02s$	
Episode Duration	$25s$	
Initialization	Initialized by best of $N = 1$ (Gaussian noised with $\sigma = 0.000001$)	Initialized by best of fig. 4.6, center (Gaussian noised with $\sigma = 0.000001$)
Population Size	50	
Selection Size	15	
Crossover/Copy	Copy	
Individual Mutation Probability	0.95	
Gene Mutation Probability	0.1	0.05
Gene Mutation Type	Gaussian distributed, $\sigma = 0.01$	Gaussian distributed, $\sigma = 0.008$
Start Fitness	1.74 (without Gaussian noise)	7.43 (without Gaussian noise)
Best Fitness (Generation)	7.43 (44, 175)	8.07 (26, 791)
Processed Generations	50, 470	40, 001
Calculation Time	$22d\ 8h$	$14d\ 7h$

Table A.2: Parameters for incremental evolutions with cyclic function controller (Figure 4.6).

Evolution	Fig. 4.10, above	Fig. 4.10, center	Fig 4.10, below
Controller Type	Neural Oscillator	Neural Oscillator, symmetric	
Task	Forward (fitness=distance)		
#Parameters	61	34	34
Parameter Range	$-4.0 \ldots + 4.0$		
Step Length	$0.01s$	$0.01s$	0.005
Netupdate Time	$0.05s$		
Episode Duration	$25s$		
Initialization	Gaussian distributed, $mean = 0.0$, $\sigma = 0.01$		Best of Fig. 4.10,center (Gaussian noised with $\sigma = 0.000001$)
Population Size	50		
Selection Size	15	15	5
Crossover/Copy	Copy		
Individual Mutation Probability	0.9		
Gene Mutation Probability	0.1		
Gene Mutation Type	Gaussian distributed, $\sigma = 0.01$	Gaussian distributed, $\sigma = 0.01$	Gaussian distributed, $\sigma = 0.001$
Start Fitness	0.0	0.0	10.63
Best Fitness (Generation)	1.28 (5, 415)	10.63 (9, 039)	10.71 (8, 063)
Processed Generations	11, 407	13, 232	9, 764
Calculation Time	$15d$ $16h$	$12d$ $6h$	$34d$ $17h$

Table A.3: Parameters for evolutions with neural oscillator controller (Figure 4.10).

Evolution	Fig 4.18, above	Fig 4.18, below
Controller Type	Neural Oscillator	
Task	Turn (fitness=angle)	Rightwards (fitness=distance)
#Parameters	34	34
Parameter Range	$-4.0 \ldots + 4.0$	
Step Length	$0.01s$	
Netupdate Time	$0.05s$	
Episode Duration	$25s$	
Initialization	Gaussian distributed, $mean = 0.0$, $\sigma = 0.01$	
Population Size	50	
Selection Size	15	
Crossover/Copy	Copy	
Individual Mutation Probability	0.90	
Gene Mutation Probability	0.1	
Gene Mutation Type	Gaussian distributed, $\sigma = 0.01$	
Start Fitness	0.0	0.0
Best Fitness (Generation)	$566.61°$ $(10,332)$	4.32 $(13,865)$
Processed Generations	$17,593$	$18,062$
Calculation Time	$20d\ 7h$	$20d\ 7h$

Table A.4: Parameters for evolutions with neural oscillator controller (Figure 4.18).

Bibliography

[1] *Simulating Pathological Gait using the Enhanced Linear Inverted Pendulum Model*. Taku Komura, Akinori Nagano, Howard Leung, Yoshihisa Shinagawa, 2004.

[2] A. J. Ijspeert. *Locomotion, Vertebrate*, pages 649–654. MIT Press, 2 edition, 2002.

[3] A.J. Ijspeert and J.-M. Cabelguen. Gait transition from swimming to walking: Investigation of salamander locomotion control using nonlinear oscillators. Technical report, Swiss Federal Institute of Technology, 2002.

[4] Akinobu Fujii, Akio Ishiguro and Peter Eggenberger. Evolving a CPG controller for a biped robot with neuromodulation. In *Proceedings of the 5th International Conference on Climbing and Walking Robots*, pages 17–24, Paris, France, 2002.

[5] Alex M. Andrew. Understanding Intelligence, by Rolf Pfeifer and Christian Scheier, MIT Press, Cambridge, Mass., 1999, ISBN 0-262-16181-8. *Robotica*, 18(6):687–689, 2000.

[6] OSG Community. Homepage of the OpenSceneGraph project. http://www.openscenegraph.org.

[7] Daniel Hein, Ralf Berger. Homepage of Simloid Project http://www.robocup.de/AT-Humboldt/simloid.shtml.

[8] Dare A. Wells. *Theory and Problems of Lagrangian Dynamics*. McGraw-Hill, New York, 1967.

[9] G. T. Fallis. 1888. U. S. Patent No. 376588,
Available at http://www.tam.cornell.edu/.ruina/hplab.

[10] François Chapeau-Blondeau, Gilbert A. Chauvet. Stable, oscillatory, and chaotic regimes in the dynamics of small neural networks with delay. *Neural Networks*, 5(5):735–743, 1992.

[11] Frank Pasemann, Manfred Hild, Keyan Zahedi. SO(2)-Networks as Neural Oscillators. In *IWANN (1)*, pages 144–151, 2003.

[12] Gen Endo, Jun Nakanishi, Jun Morimoto, and Gordon Cheng. Experimental studies of a neural oscillator for biped locomotion with QRIO. In *IEEE 2005: International Conference on Robotics & Automation*, 2005.

[13] Hillel J. Chiel, Randall D. Beer, John C. Gallagher. Evolution and Analysis of Model CPGs for Walking: I. Dynamical Modules. *Journal of Computational Neuroscience*, 7(2):99–118, 1999.

[14] Humanoid Team Humboldt. Homepage of Humanoid Team Humboldt.
http://www.humanoidteamhumboldt.de.

[15] Jerry E. Pratt. *Exploiting inherent robustness and natural dynamics in the control of bipedal walking robots*. PhD thesis, 2000. Supervisor-Gill A. Pratt.

[16] Jin'ichi Yamaguchi, Eiji Soga, Sadatoshi Inoue, Atsuo Takanishi. Development of a Bipedal Humanoid Robot: Control Method of Whole Body Cooperative Dynamic Biped Walking. In *IEEE 1999: International Conference on Robotics & Automation*, pages 368–374, 1999.

[17] Joshua Clifford Bongard and Hod Lipson. Nonlinear System Identification Using Coevolution of Models and Tests. *IEEE Trans. Evolutionary Computation*, 9(4):361–384, 2005.

[18] K. Endo, T. Maeno, and H. Kitano. Co-evolution of morphology and walking pattern of biped humanoid robot using evolutionary computation - consideration of characteristic of the servomotors. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'2002)*, pages 787–792, 2002. Sponsor: Swiss National Science Foundation.

[19] K. J. Astrom and T. Hagglund. *PID Controllers: Theory, Design, and Tuning, 2nd Ed.* The Instrument, Systems, and Automation Society, Research Triangle Park, NC, 1995.

[20] K. Osuka. Dynamics based control of mechanical systems. In *IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pages 566–570, Piscataway, NJ, 2001. IEEE Press.

[21] Kaneko, K., Kanehiro, F., Kajita, S., Hirukawa, H., Kawasaki, T., Hirata, M., Akachi, K., and Isozumi, T. Humanoid Robot HRP-2. In *IEEE 2004: International Conference on Robotics & Automation*, pages 1083–1090, 2004.

[22] Kazuo Hirai, Masato Hirose, Yuji Haikawa, Toru Takenaka. The Development of Honda Humanoid Robot. In *ICRA*, pages 1321–1326, 1998.

[23] Kensuke Harada, Shuuji Kajita, Kenji Kaneko, and Hirohisa Hirukawa. ZMP Analysis for Arm/Leg Coordination. In *IEEE/RSJ 2003: Intl. Conference on Intelligent Robots and Systems*, 2003.

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

[25] Kyosuke Ono, Takasahi Furuichi, Ryutaro Takahashi. Self-Excited Walking of a Biped Mechanism with Feet. *I. J. Robotic Res.*, 23(1):55–68, 2004.

[26] M. Vukobratovic, B. and Borovac. Zero-Moment Point – Thirty five years of its life. *International Journal of Humanoid Robots*, 1(1):157–173, 2004.

[27] M. Wisse, A. L. Schwab, R. Q. van der Linde. A 3D passive dynamic biped with yaw and roll compensation. *Robotica*, 19(3):275–284, 2001.

[28] Martin Golubitsky, Ian Stewart, Pietro-Luciano Buono, J. J. Collins. A modular network for legged locomotion. *Phys. D*, 115(1-2):56–72, 1998.

[29] Martin Lötzsch and Joscha Bach and Hans-Dieter Burkhard and Matthias Jüngel. Designing Agent Behavior with the Extensible Agent Behavior Specification Language XABSL. In Daniel Polani, Brett Browning, Andrea Bonarini, and Kazuo Yoshida, editors, *RoboCup 2003: Robot Soccer World Cup VII*, volume 3020. Springer, 2004.

[30] Masaki Ogino. *Embodiment Approaches to Humanoid Behavior – Energy efficient walking and visuo-motor mapping*. PhD thesis, Osaka University, January 2005.

[31] Oliver Obst, Markus Rollmann. SPARK – A Generic Simulator for Physical Multiagent Simulations. *Computer Systems Science and Engineering*, 20(5), September 2005.

[32] Paul du Bois Reymond. Über die Fourierschen Reihen. *Nachrichten von der Königl. Gesellschaft der Wissenschaften und der Georg-Augusts-Universität zu Göttingen*, (21):571–582, 1873.

[33] Prof. Dr. H.-D. Burkhard. Homepage of Students Laboratory "Modern Approaches of Artificial Intelligence", HU Berlin 2006. http://www.ki.informatik.hu-berlin.de/lehre/ss06/mmki-prakt_html.

[34] Ralf Berger. Die Doppelpass-Architektur - Verhaltenssteuerung autonomer Agenten in dynamischen Umgebungen. Diploma thesis, Institut für Informatik, Humboldt Universität zu Berlin, 2006.

[35] Randall D. Beer, Hillel J. Chiel, John C. Gallagher. Evolution and Analysis of Model CPGs for Walking: II. General Principles and Individual Variability. *Journal of Computational Neuroscience*, 7(2):119–147, 1999.

[36] Rechenberg, I. . *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart, GER, 1973.

[37] Richard Reeve. *Generating walking behaviours in legged robots*. PhD thesis, University of Edinburgh, 1999.

[38] Robert Haschke, Jochen J. Steil, Helge Ritter. Controlling Oscillatory Behaviour of a Two Neuron Recurrent Neural Network Using Inputs. In *Proc. of the Int. Conf. on Artificial Neural Networks (ICANN)*, Wien, Austria, 2001.

[39] RoboCup. Homepage of Robocup Federation. 1997. http://www.robocup.org.

[40] Roland Arbinger. *Psychologie des Problemlösens*. Primus Verlag, Darmstadt, 1997.

[41] Russel Smith. *Open Dynamics Engine v0.5 User Guide*. http://www.ode.org, 2006.

[42] S. H. Collins, M. Wisse, and A. Ruina. A 3-D passive-dynamic walking robot with two legs and knees. *The International Journal of Robotics Research*, 20(7):607–615, 2001.

[43] S. Hashimoto, S. Narita, H. Kasahara et al. Humanoid Robots in Waseda University–Hadaly-2 and WABIAN. *Auton. Robots*, 12(1):25–38, 2002.

[44] Yasuo Kuniyoshi Akihiko Nagakubo Seiichi Miyakoshi, Gentaro Taga. Three Dimensional Bipedal Stepping Motion Using Neural Oscillators — Towards Humanoid Motion in the Real World. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 84–89, Piscataway, NJ, 1998. IEEE Computer Society.

[45] Shuuji Kajita and Kazuo Tani. Adaptive Gait Control of a Biped Robot Based on Realtime Sensing of the Ground Profile. *Auton. Robots*, 4(3):297–305, 1997.

[46] Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, K. Yokoi, H. Hirukawa. The 3D Linear Inverted Pendulum Mode: A Simple Modeling for a Biped Walking Pattern Generation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 239–246, 2001.

[47] Thomas A. McMahon Simon Mochon. Ballistic Walking. *J. Biomechanics*, 13(1):49–57, 1980.

[48] Russell Smith. Homepage of Open Dynamics Engine project. http://www.ode.org.

[49] Squin, Carlo H. SDL: Scene description language.

[50] Sten Grillner. Neurobiological Bases of Rhythmic Motor Acts in Vertebrates. *Science*, 228(4696):143–149, 1985.

[51] Sten Grillner. Neural Networks for Vertebrate Locomotion. *j-SCI-AMER*, 274(1):48–53 (Intl. ed.), Jan 1996.

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

[53] Tad McGeer. Passive Walking with Knees. In *IEEE 1990: International Conference on Robotics & Automation*, pages 1640–1645, 1990.

[54] Thomas A. McMahon. *Muscles, Reflexes, and Locomotion*. Princeton University Press, Princeton, New Jersey, 1984.

[55] Thompson, John M. T., Stewart, H. Bruce. *Nonlinear Dynamics and Chaos*. John Wiley & Sons, Chichester, 2nd edition edition, 2002.

[56] Toyota Motor Corporation. Homepage of Toyota Motor Corporation. http://www.toyota.co.jp/jp/special/robot/.

[57] University of Jena, Institute of Sport Science. Homepage of Locomotion Laboratory.
http://www.lauflabor.de.

[58] Yoshihiro Kuroki, Masahiro Fujita, Tatsuzo Ishida, Ken'ichiro Nagasaka, Jin'ichi Yamaguchi. A small biped entertainment robot exploring attractive applications. In *ICRA*, pages 471–476, 2003.