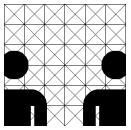
Diploma Thesis
Course of Studies Computer Science

Development of Bio-inspired Locomotion Using Modular Robotic Simulation and Control System

at Research Cluster of
Technical Aspects of Multimodal Systems,
University of Hamburg

submitted by
Dennis Krupke
1. October 2013



supervised by
Prof. Dr. Jianwei Zhang
Prof. Dr. Houxiang Zhang

Zusammenfassung

Zur Erzeugung von Fortbewegungsmustern modularer Roboter wurde ein komplett integriertes Simulationssystem entwickelt.
Mit Hilfe einer grafischen Benutzungsschnittstelle ist es möglich mit Simulationen zur Laufzeit zu interagieren und diese zu überwachen. Durch die Möglichkeit das System auch ohne grafische Benutzungsschnittstelle (GUI) zu starten, können Langzeitoptimierungen als Hintergrundprozesse gestartet werden. Das System ermöglicht Benutzern verschiedene Arten der Fortbewegung mit Hilfe einer standardisierten Programmierschnittstelle für Algorithmen, die Aktuatoren ansteuern, untereinander zu vergleichen. Aufgezeichnete Daten vergangener Simulationen können untersucht und bei Bedarf zu Dateien mit durch Tabulatoren separierten Werten (*.tsv) exportiert werden.
Das vorgestellte System ist als Rahmenwerk für Menschen mit unterschiedlichem Kenntnisstand über Fortbewegungsprinzipien in modularer Robotik zu verstehen. Die Hauptfunktionen sind die Entwicklung, Analyse und Optimierung von Fortbewegungalgorithmen für modulare Roboter in unterschiedlichen Konfigurationen. Als besonders nützlich hat sich die Möglichkeit erwiesen komplexe Steuerungsparameter zu optimieren, deren Bedeutung und Wirkung nur schwierig zu verstehen sind und deren Korrelationen mit dem Verhalten des zugehörigen Algorithmus und anderen Parametern nicht klar sind. Die Anwendung des Systems zur Optimierung von Steuerungsparametern ist für zwei unterschiedliche Fälle gezeigt. Um die Nützlichkeit des vorgestellten Systems zu demonstrieren, werden in diesen Experimenten zwei verschiedene Steuerungsalgorithmen, die unter Zuhilfenahme abstrakter Basisklassen implementiert wurden, automatisch verbessert.
Das präsentierte System zeichnet sich durch seine Exklusivität in Bedienbarkeit, Funktionsumfang und Nutzen für die Forschung im Bereich Fortbewegung modularer Roboter aus. Meines Wissens gibt es im Bereich modularer Roboter keine vergleichbare Arbeit.

Abstract

This work presents an integrated simulation system for development of modular robotic locomotion patterns.
Using graphical configuration interfaces users can create robots, attach sensors, create and assign actuation algorithms, build an environment and optionally set up an optimization mode. The graphical control interface allows to interact with the simulation at runtime and to supervise it. With the possibility to run the system without graphical user interface (GUI), long-term optimizations can be performed as background processes. The system enables users to compare different locomotion patterns with the help of a standardized programming interface for actuation algorithms. Recorded data from simulations can be inspected and, if needed, exported to tab-separated value files (*.tsv) for further usage.
The proposed system can be regarded as a framework for people at different levels of knowledge about modular robot locomotion principles. Its main functions are development, analysis and optimization of locomotion algorithms for different configurations of modular robots. Great benefit arises from the possibility to optimize even not well understood and complex control parameters, whose correlations with the behaviour of the associated control algorithm and other parameters are not clear. An application of the system for control parameter optimization is shown in two different experiments. In these experiments two control algorithms implemented using abstract base interfaces of the system, are automatically improved to demonstrate advantages of the system.
For the best of my knowledge the presented system offers a combination of usability, features and benefits for research in the field modular robotic locomotion that exceeds other systems.

Contents

List of Figures

List of Tables

Introduction

1

Modular robots are a marvellous platform for research and education. They are low cost and their structure is easy to understand. Because of their modular and hyper redundant design it is possible to recombine several modules to new robots with new capabilities. In this way they offer easy and fast prototyping of new topologies and control strategies. Using arbitrary sensors and different methods of sensor fusion and integration it is possible to achieve intelligent locomotion behaviour with simple hard- and software. Principles of efficient locomotion can be investigated as well as mechanisms to adapt to terrains' difficulties autonomously. To design algorithms that produce locomotion efficiently, a platform is needed that allows to implement and compare different concepts in hard- and software. In the following a very flexible system is introduced. Figure 1.1 shows the GUI in application with a real robot. It is explained how to use it to design and optimize new ideas of efficient and intelligent locomotion in modular robotics.

The intention was to build a system that allows inexperienced users as well as experts to create simulations of modular robot locomotion. After a simulation set-up has been created the user will be able to optimize and evaluate his results in an easy but detailed way. This makes it applicable for educational and scientific purposes.

In the following the motivation of the proposed work is given. It is followed by an description of the structure of this thesis.

1.1 Motivation

The main idea of this work is to create a system that simplifies the process of locomotion development in modular robotics. One of my objectives was to make it suitable for education and research. Students need a system that comes with everything necessary for programming, running and evaluating real or simulated robot locomotion. Using such a system helps to get in contact with the field of modular robots. Scientists need freedom of action to implement their own ideas with as few limitations as possible. But they also want to have a system that can be regarded as closed solution. Re-use of already configured parts, as well as support in time consuming subtasks is needed to work efficiently.

My intention was to build a system that allows to configure different set-ups, to control and supervise simulations or real robot control loops and to optimize control algorithms. The requirement was to create a general interface for modular robotic control and locomotion algorithms to enable users to implement their own ideas in hard- and software.

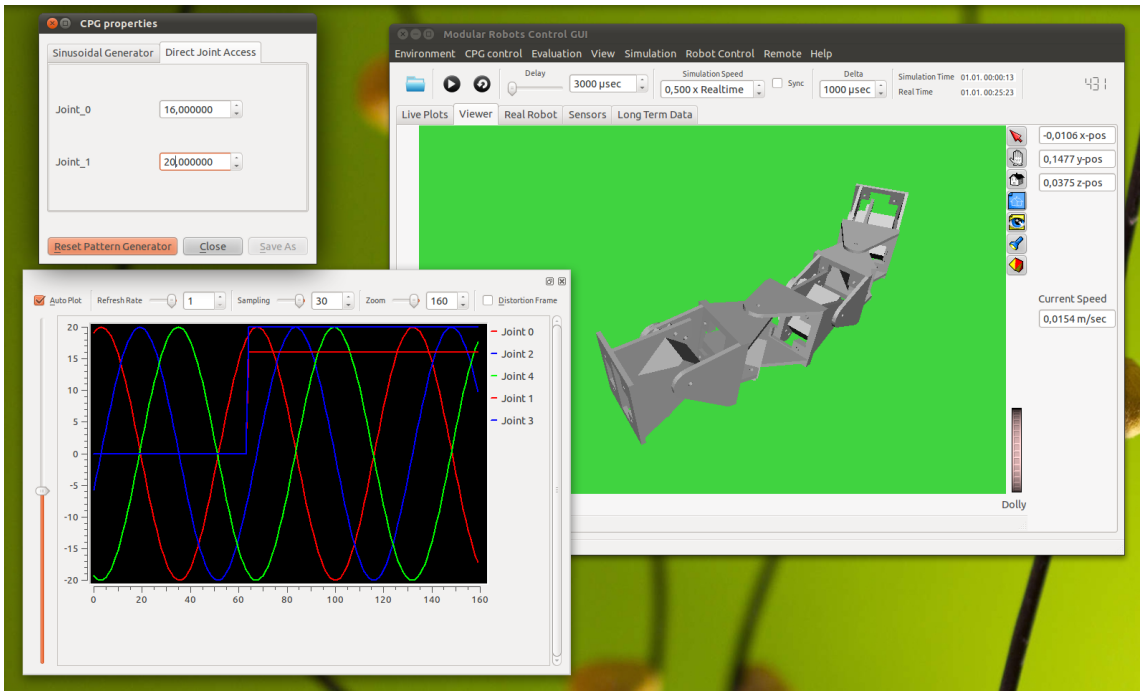


Figure 1.1: Example of the proposed system. The GUI of the control system is shown while operating a real robot. The simulated robot shows the desired state of the real robot. In the plot window the desired angles of the joints are displayed versus time. The small window in the upper left corner is used to change important control parameters.

Comparisons of different locomotion strategies are simplified when using the same simulation or control configuration except concrete composition of locomotion algorithms. For evaluation purposes data handling is needed to record and visualize processes of simulation and control. Thus, the current state of robots and algorithms actuating their joints should be recorded and visualized at runtime. Implementations of some heuristic optimization methods would allow to optimize the parameters determining characteristics of a certain actuation algorithm. It should also be possible to integrate different robot models by implementing arbitrary robot control libraries and importing corresponding 3d models for the simulated robots. Such a system can be regarded as a closed solution for modular robot science and can be very useful as an assisting tool in creating intelligent modular robots with autonomous polymorphic locomotion methods.

Letting modular robots in chain-like configuration move, inspired by the movements of real animals like worms, caterpillars and snakes, can be the beginning in the process of creating efficient locomotion patterns. To reduce the complexity of the locomotion generation problem, topologies of robots used in this work are all chain-like but with different numbers of modules and arbitrarily oriented joints[1]. This work does not claim that biology offers most efficient and best ways to move artificial creatures, but analysis of

[1] Orientations of joints can be pitching or yawing.

inventions from the evolution can help to find some ideas for designing artificial locomotion algorithms.

1.2 Structure of this work

This thesis presents a novel software system for research and education in the field of locomotion of modular robots in a chainlike configuration. The usefulness is demonstrated by its successful application for the optimization of control parameters of locomotion algorithms. The thesis is structured as follows:
The second chapter introduces locomotion of limbless cylindrical animals and gives an overview of creeping animals' kinematics research. These ideas and some principles can be taken into account when designing algorithms to move modular robots on different terrain. Additional benefits arise from theoretical formulations of simplified models for evaluating and rating the results from optimization of control algorithms.
Chapter three provides a description of the history and state-of-the-art of chain-like modular robots. Some basics on modular robot actuation are explained to transfer the knowledge about animal kinematics into algorithms moving modular robots. Sinusoidal generators, central pattern generators (CPGs) and adaptive mechanisms are explained. After describing the control parameter problem two different optimization methods are introduced which are used in the proposed system to improve the locomotion of simulated modular robots. The chapter closes with a comparison of related work.
Chapter four presents the main components of the proposed system that allows to implement, evaluate, compare and optimize arbitrary locomotion algorithms. After an explanation of its components and capabilities the usage is explained. The presented system is a framework for research and education and combines flexibility, extendibility and easy usage.
Generalized programming interfaces are described in chapter six. They allow e.g. to implement any desired algorithm to control groups of joints of modular robots. Algorithms using sensor feedback are also taken into account.
Chapter six presents two different experiments and their results from the application of two different previously explained optimization methods from chapter three. The results from the experiments show the usefulness of the proposed system. Both methods calculated good solutions for parameters which determine the characteristics and behaviour of the algorithms that generate locomotion patterns.
Chapter eight, the last chapter contains a discussion of the results of both optimization methods in different cases of optimization. The thesis is evaluated with the help of the scientific application of the system for the optimization of locomotion patterns. The results show that the proposed system is useful for locomotion pattern generation and optimization in research. For the best of my knowledge there is no other system that offers a similar amount of functionality and usability than presented in this thesis. For future work ideas to improve the system are explained.
In the appendix are a few important abstract base interfaces of the system. In addition there are some examples for configuration files and data files. On the DVD the source code system is stored as well as a *howto*-document that contains hints about installation and usage of the software. In addition on the DVD an automatically generated documentation

is stored. It has been created using the *Doxygen* system. Additionally several videos are stored that show real and simulated modular robots that are controlled by using the proposed system.

1.3 Summary

This chapter introduced the thesis and the motivation to create the proposed system. A flexible system that allows to implement control algorithms for modular robots and provides the user with useful functions for development and optimization at the same time is introduced. After this introduction the structure of the thesis was explained. The next chapter contains basic knowledge about creeping animal locomotion. This knowledge helps to design bio-inspired locomotion patterns and to understand how these animals move.

Principles of Creeping Motion

2

This chapter imparts basic knowledge about animal locomotion. Modular robot research benefits from this knowledge since it helps to understand locomotion principles of longitudinal animals in order to create own locomotion patterns for robotics.
After a description of the motivation of bio-inspired locomotion, section 2.2 clarifies terms of kinematics that are used in later sections. Section 2.3 presents different models of crawling. These models help to understand which physical terms are important in locomotion of creeping animals in order to design efficient locomotion of robots with similar structure. They are also useful to estimate the energy consumption of different movements. This helps to rate different possibilities of applying locomotion patters. In section 2.4 and section 2.5 deeper insights from biology are given. Research of caterpillars and snakes helps to apply initial values to parameters that determine the characteristics of locomotion patterns. In the end of this chapter, a conclusion is given in section 2.6 that contains a summary of animal kinematics.

2.1 Introduction

Creeping animals with longitudinal bodies show off amazing capabilities in locomotion. Caterpillars and snakes can easily pass unstructured terrain. They feature impressing techniques to overcome arbitrary obstacles. Many differences lie in properties of their bodies. Caterpillars, worms and snakes have soft and flexible bodies that can easily adapt to uneven terrain, most robots however consist of rigid body parts. Longitudinal muscles in addition to very flexible bones provide snakes with amazing locomotion capabilities. Animals are not supposed to offer the best locomotion patterns for artificial creatures like a robotic snake or caterpillar robot but they can express some helpful principles. By using these principles guidelines for designing locomotion algorithms for robots in chain-like configurations can be established.

2.2 Locomotion basics

From the kinematics point of view in creeping animal locomotion there are some definitions that need to be mentioned in order to understand the following results from research in biology.

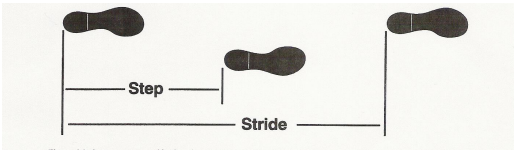


Figure 2.1: Illustration of step and stride. (Alexander [2])

2.2.1 Definitions and criteria of comparison

To compare and rate different locomotion techniques under different aspects definitions of several measures are necessary. This paragraph presents some basic measures that are used in this chapter and helps to understand the following models of locomotion.

- *Acceleration*

$$a = \lim_{\Delta t \to 0} \frac{\Delta v}{\Delta t} = \frac{dv}{dt} \tag{2.1}$$

  Acceleration involves the need to overcome inertia of the moving body and friction with the ground. From energetics point of view this is very expensive. For hunting fast acceleration is more important than high speed.

- *Center of gravity (COG)*
  The COG is a geometric property of any creature or object. It describes the average location of the weight of an object and can be used to review stability of poses.

- *Duty factor*
  The duty factor of a certain gait is the percentage of the total cycle which a given foot is on the ground. Depending on duty factors walking and running can be distinguished and current total friction can be estimated. Normally gaits with more than 50% are considered a *walk*, while those with less than 50% are considered a run.

- *Economy of energy*
  The oxygen consumption of animals was measured under different circumstances. Animals need energy for growth and reproduction, so they need to save energy. That means energy saving leads to surviving in nature. In robotics energy-saving means a longer working time, because of the limitation by batteries in mobile applications.

- *Froude number*

$$Fr = \frac{\text{centripetal force}}{\text{gravitational force}} = \frac{mv^2/l}{mg} = \frac{v^2}{gl} \tag{2.2}$$

  A dimensionless number comparing inertia and gravitational forces. Initially it is used in hydrodynamic systems to compare objects of different sizes. In case of gait patterns where walking is modelled as inverted pendulum it is defined as ratio of centripetal force to gravitational force. Alexander [2] used Froude numbers to rate general movement capabilities of different animals.

- *Mechanical cost of transport*
  Required work to move the mass of creatures a certain distance.

- *Metabolic cost of transport*
  Amount of metabolic energy used to move one unit mass of the creature one unit of distance
- *Speed*
  $$v = \lim_{\Delta t \to 0} \frac{x(t + \Delta t) - x(t)}{\Delta t} = \frac{dx}{dt} \tag{2.3}$$
  High speed means much higher energy consumption. Natural creatures choose high speed for hunting or for running away from a predator. To save energy most animals do not move with high speed if they do not need to.
- *Stability*
  Creatures that move very slowly are possibly very economical with their energy. But they have a lack in stability which can only be compensated by the choice of a good gait. A lack in stability can be regarded as consequence of missing stabilizing forces. In freely rolling wheels, these forces prevent from overturning when rolling with appropriate speeds. Generally creatures are in stable poses when the projection of their center of gravity lies within a plane created by at least three supporting points of contact with the ground.
- *Stride*
  One stride is a complete cycle of movement.
  - *Stride length* is the distance moved in a single stride.
  - *Stride frequency* describes the number of strides taken in one unit of time.

2.2.2 Optimization

Locomotion modes of creatures always represent trade-offs. It is impossible to have the lowest energy consumption possible while moving at highest speed. Depending on desired results there are several optimal set-ups with different focus:

- highest speed
- largest acceleration
- best energy saving
- best stability
- best efficiency
- most smooth locomotion

Before an optimization can be started it has to be chosen carefully among these or similar set-ups from the list above to produce results according to the resulting needs.

Evolution in nature does not create best imaginable structures for single special purposes, but it optimizes existing properties by refining following generations. Not only the design of the physical structure but also the control of propulsive body parts determines efficiency of locomotion. In this way muscle actuation patterns affect the amount of consumed energy and the resulting speed of movements.

2.2.3 Gaits

Gaits are timing sequences for lifting legs and placing them on the ground. Each creature needs distinct gaits for different purposes. An energy saving movement pattern should be used in *normal* situations if it is desirable to move from A to B without any constraints like moving as fast as possible. There are movement patterns that provide best propulsion rate, others that are energy saving and some for special purposes e.g. with best stability conditions for difficult terrain.
In general, gaits can be understood as solutions for optimization problems. These optimization problems can be formulated as a desired movement, e.g. to move fast and statically stable from A to B. For real world application it is not important to get the best solution, but to reach the goal position B. Each creature needs a set of gaits to move according to its needed applications. Adapting to irregular terrain, escaping very fast from predators or hunting while moving as silent as possible are tasks in nature that need proper gaits.
Kinematically gaits can be differentiated by their duty factors. Generally gates with duty factors bigger than 0.5 are described as walks, while smaller duty factors represent runs. Proper usage of different gaits requires smooth gait transitions. Switching from one to another gait needs to be fast and must conserve stable states. Often there are some stages in motion where stability criteria are not fulfilled when switching to another gait.

2.3 Principles of crawling

In the following, some simple models of crawling are described. They are useful to understand principles of creeping motion and to do some rough estimates on energy costs.

2.3.1 Two-anchor crawling

A very simple model for crawling is described by Alexander [2]. It uses an artificial animal that is able to lengthen and shorten its body. Protrusion is created by applying force to the ground that is bigger than the frictional force in the opposite direction of travel and smaller than the frictional force in direction of travel. In figure 2.2 A the side with contact to the ground has bristles. These are directed to a position that allows the creature to propulse itself without slipping back. To move forward it stretches to the front and after that it shortens its body without sliding back at the front part.
Shortening and lengthening by λ in one cycle means, that the movement has the stride length λ. The mass of the animal is m and the gravitational acceleration g. The coefficient of friction with the ground while moving forward is called $\mu_{forward}$ and for sliding backward μ_{back}. The coefficient of friction for backward sliding is larger because of the bristles. If not it could not move forward, because no propulsion in desired direction of travel can be generated. The animals weight is mg and leads to the frictional force resisting its forward motion $\mu_{forward} \cdot mg$. In each stride work of $\mu_{forward} \cdot mg\lambda$ has to be done. The mechanical cost of transport T is the work per unit mass and per unit distance.

$$T_{friction} = \mu_{forward} g. \tag{2.4}$$

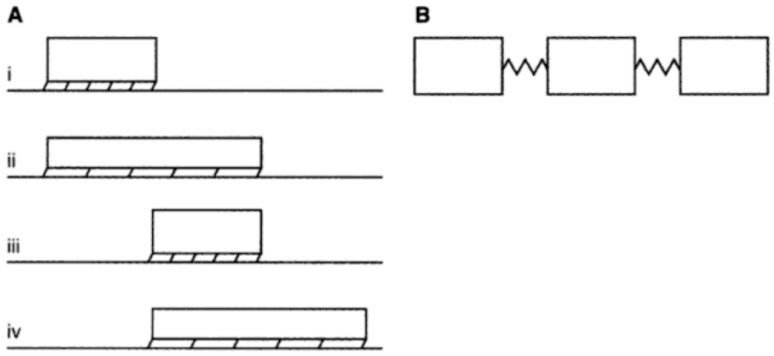


Figure 2.2: Two different illustrations of the principle of two-anchor crawling. (A) Shows an elastic body that creates static friction against the direction of travel with the help of directed bristols. A(i) represents the beginning of one cycle of movement where the creature is resting. Stage (ii) shows the creature in stretched state. The directed bristols on the downside allow the creature to prolong forward without slipping backwards. (iii) Now the creature has already shortened its body again. Again the bristols prevent slippage. In (iv) again lengthening of the body occurs. (B) Shows a body with three segments and elastic connections in between that is able to move forward or backward without additional mechanisms. (Alexander [2])

But this works only at low speeds. Repeated acceleration and stopping means producing and losing the kinetic energy. While moving fast a lot of energy gets wasted in stopping the animals body.
Small modifications of the artificial creature lead to a three-segment crawler, shown on figure 2.2 B, with a kind of springs between the three segments for shortening and enlarging the body. While this creature moves with constant velocity v the segment in the middle moves with velocity v too. But the first and last part are stationary half the time so they have to move with $2v$ for the other half. The amount of kinetic energy that is gained and lost in each stride λ is $\frac{1}{2}\left(\frac{2m}{3}\right)(2v)^2 = \frac{4mv^2}{3}$. Now inertial cost of transport can be calculated.

$$T_{inertia} = \frac{4v^2}{3\lambda}. \tag{2.5}$$

At high speed the inertial cost is higher than the frictional cost and at low speed it is the other way round. Both are equal in case of $\frac{v^2}{\lambda g} = 0.75\mu_{forward}$. $\frac{v^2}{\lambda g}$ is a Froude number. Maggots crawl in a similar way advancing 0.15-0.25 body lengths per stride.

2.3.2 Crawling by peristalsis

Another technique of crawling is comparable to movements of an earthworm. Segments of soft-bodied creatures are shortening and lengthening repetitively. Waves travel backwards along the body. Each segment moves forward during the phase of lengthening and remains in place when shortening. Segments that are directly behind the last lengthening segment are prevented from sliding back. By doing this segments in front of the shortening parts can be pushed forward. In the same way the segments behind the shortening parts are

pulled forward.
The coefficients of friction while sliding are called $\mu_{forward}$ and μ_{back}. The fraction of segments moving forward at any time is denoted with q. To provide protrusion frictional force ($(1-q)mg\mu_{back}$), preventing shortening segments from sliding back has to be greater than frictional force of elongating segments ($qmg\mu_{forward}$). That means:

$$q < \frac{\mu_{back}}{\mu_{back} + \mu_{forward}} \tag{2.6}$$

When locomotion is performed slowly enough, inertial forces can be neglected. Frictional cost of transport is similar to the two-anchor crawling model (Equation 2.1) described above.

$$T_{inertia} = \frac{v^2}{2q^2\lambda} \tag{2.7}$$

The frictional and inertial cost of transport are equal when the Froude number based on stride length, $\frac{v^2}{\lambda g}$, equals $2\mu_{forward} \cdot q^2$.

2.3.3 Serpentine crawling

Snakes move with the help of bending waves. These waves are travelling backward along the body to push the animal forward. A rough simplification of crawling snakes in figure 2.3 shows the body of a snake in zigzag. To denote the frictional coefficient with the

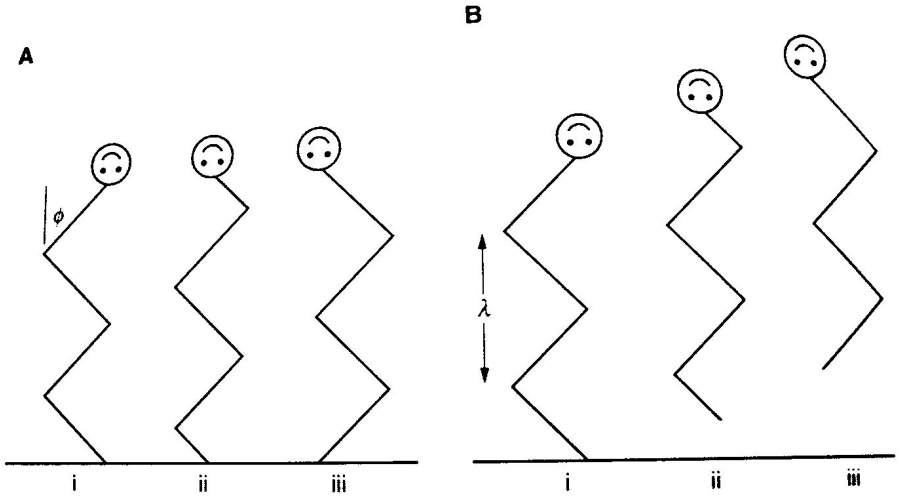


Figure 2.3: Illustration of the principle of serpentine movement. Maximum bending angle ϕ and stride length λ determine frictional and inertial forces in serpentine movement. (Alexander [2])

ground μ_{axial} is used for moving along the body axis and $\mu_{transverse}$ for moving sideways. After dividing the body into segments, each of mass δm, we can distinct resulting forces $\delta mg \cdot \mu_{axial}$ and $\delta mg \cdot \mu_{transverse}$. Both are involved in creating propulsion of the whole

body with respect to the ground. To move along the body axis the forward component of the available transverse frictional force must exceed the backward component of the axial frictional force.

$$\mu_{transverse} \cdot \sin\phi \;>\; \mu_{axial} \cdot \cos\phi \tag{2.8}$$

$$\tan\phi \;>\; \frac{\mu_{axial}}{\mu_{transverse}}$$

Stride length λ equals wavelength of body-waves. To travel distances of λ, snakes have to slide $\frac{\lambda}{\cos\phi}$ in zigzag path. In a single stride frictional forces from ground and snakes' body can be calculated by evaluating $\frac{\mu_{axial} \cdot mg\lambda}{\cos\phi}$.
Crawling at speed of v means that each segment slides with speed $\frac{v}{\cos\phi}$ at angles of about $\pm\phi$ in direction of travel. Velocity has a transverse component $\pm v\tan\phi$. Each time a segment yaws to the other direction this transverse component is lost and regained afterwards. Kinetic energy, calculated by $\frac{1}{2}\delta m v^2\tan^2\phi$, gets lost and has to be recreated when moving goes on. This means for each stride that this happens two times for each segment. Inertial work for the whole body can be calculated by evaluating $mv^2\tan^2\phi$.
The frictional cost of transport

$$T_{friction} = \frac{\mu_{axial} \cdot g}{\cos\phi}. \tag{2.9}$$

The inertial cost of transport

$$T_{inertia} = \frac{v^2 tan^2\phi}{\lambda}. \tag{2.10}$$

At high speed inertial cost is much larger than frictional cost. At low speed frictional cost grows and becomes much more important in contrast to the inertial cost of transport.

2.3.4 Summary of crawling methods

In this section three different simplified models of creeping motion have been introduced. They are all very useful for establishing first estimations of requirements for efficient locomotion techniques of limbless robots. Dynamic calculation of forces is needed to work on efficient locomotion patterns and to implement functions that rate these patterns (see chapter 3.4.2)
From the kinematics point of view we need friction to generate propulsion. But frictional forces in the opposite direction of travel must be greater than friction in direction of travel. Otherwise no locomotion at all or locomotion in the wrong direction will occur. The faster creatures move the more inertial forces are of importance and the more frictional forces can be neglected. Smooth movement patterns using smooth state transitions can be very energy efficient because they tend to maintain kinetic energy. It seems that the general key to fast and energy efficient locomotion techniques is to create smooth movement patterns. Several real animals demonstrate us how this can be done while maintaining high degrees of stability. Snakes, caterpillars and worms all use smooth locomotion modes. Though energy efficiency varies among different gaits of these creatures.

2.4 Caterpillar

Caterpillars make maximum use of the power that is available from real-time muscle contraction. Even if they move slow in comparison to adult insects they show off amazing locomotion capabilities in unstructured terrain and climbing tasks. The way how they move is not the most energy efficient way but it is not easy to outperform the simplicity and stability of their locomotion. That is the reason why it suits so well to modular robotic tasks. This section summarizes research of analysis of real caterpillar locomotion .

2.4.1 Caterpillar's gaits

Generally caterpillars are able to perform different gaits for locomotion. The next paragraphs describe three main gaits, shown in figure 2.4. Depending on current circumstances caterpillars choose one of these gaits.

Forward moving

Forward movement is performed by using the principle of two-anchor crawling as described in section 2.3.1, but with three anchors. Anchors of the artificial model can be assigned to *true legs* on body segments 1-3, *prolegs* on segments 6-9 and the *clasper* on segment 13 of figure 2.4. Locomotion is produced by waves of contraction of animal's body, followed by relaxation that runs along the body from tail to head. These body waves are capable of producing propulsion with the help of travelling waves. A travelling wave lets each segment be raised from the ground one after the other. After segments are stretched forward into their neighbours, they are lowered back to the ground. At least three segments are moving at all times. Brackenbury [8] observed that each foot is airborne for only 35% of a stride to maximize stability.

Retreating

From kinematics point of view the gait retreating is comparable to moving forward. It is used in case of threats by predators and produces very fast waves in reverse that arch up the whole body. Then no leg has contact to ground except claspers at terminal segment. Claspers work as an anchor until the relaxation phase of body waves lower the legs to ground. If the relaxation reaches claspers they detach at current position and reattach further back (Brackenbury [8]). Technically this movement is a kind of *reverse gallop*.

Backward roll

The third gait, displayed in figure 2.4, is the backward roll. Caterpillars use it as an escape strategy. It is the fastest gait of caterpillars but it is hard to control the direction of movement very precisely. Movement is produced through bringing the body in an unstable position that results in rolling. This gait uses the same muscles as normal movement but much more quickly to produce the needed kinetic energy. It is used as an escape strategy,

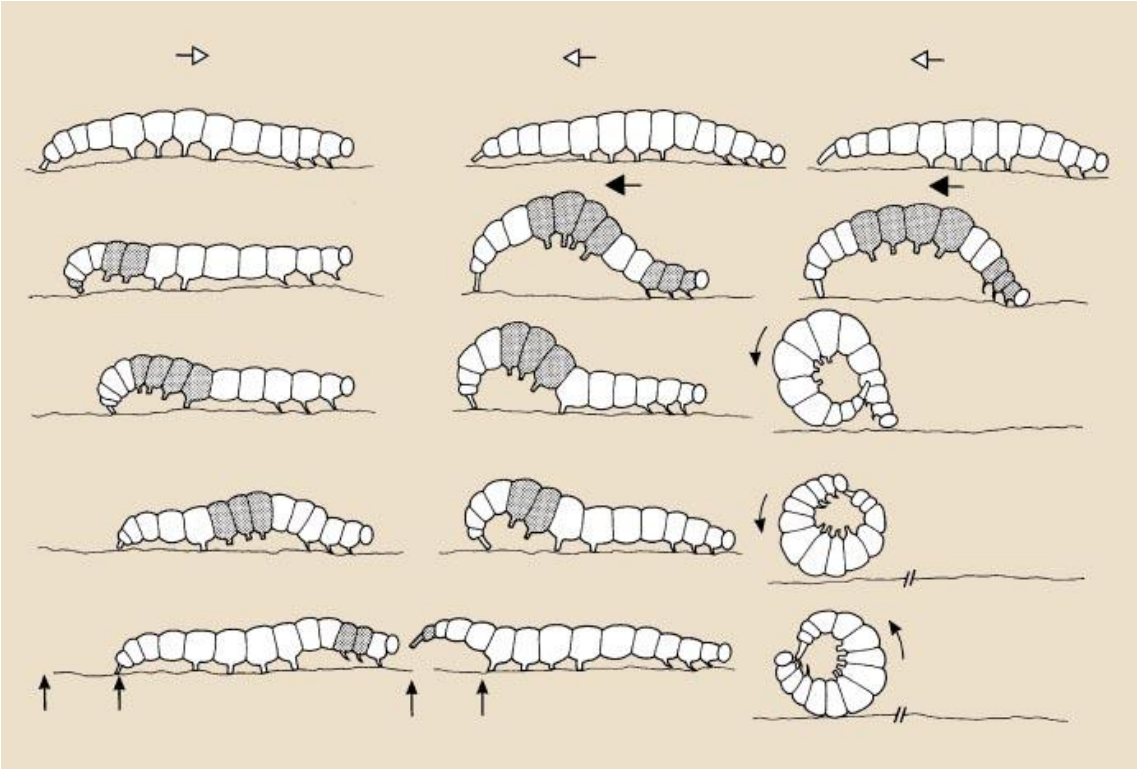


Figure 2.4: Caterpillar's movement: left: *forward movement*; middle: *retreating*; right: *rolling* (Brackenbury [8])

not for general locomotion. It works on flat terrain with up to five revolutions at a speed of about $39 \pm 3.6 \frac{cm}{s}$. In contrast to that the forward moving gait reaches only 10 $\frac{mm}{s}$.

2.4.2 Some differences in real caterpillars

Comparisons of different caterpillars show that there are quite big differences in properties of locomotion patterns. In table 2.1 some differences in averages of kinematic properties of four different caterpillars are summarized.

Gipsy moth caterpillars crawl much more slowly than adult insects of similar mass, and take much shorter strides. Casey [9] describes that they are able only to speed up to 0.03 $\frac{m}{s}$. Stride lengths up to 0.008 m were recorded. Metabolic cost of transport is very high. In comparison to running arthropods of same mass it is 4.5 times higher.
In case of *Pleuroptya caterpillar* speed, stride frequency and stride length are $1.0 \pm 0.2 \frac{cm}{s}, 1.7 \pm 0.2 Hz$ and $0.6 cm$ during normal forward walking. Like most caterpillars *Pleuroptya* uses a single locomotory wave. But there is another, called *Tyria jacobaea*, that generates about 1.32 simultaneous waves (Brackenbury [8]). Its duty factor is 66,5% at a stride frequency of 2.9Hz.
Berrigan and Pepin [6] studied crawling of larvae of dipteran fly. Larvae of masses 0.5-220 mg crawl at speeds of 0.7-10 $\frac{mm}{s}$ with stride frequencies of 0.6-2.8 Hz. In comparison to

adult insects of similar mass ,which typically reach speeds of 70-100 $\frac{mm}{s}$ with stride frequencies of 10 Hz, this is quite slow. Berrigan and Pepin [6] measured oxygen consumption of dipteran larvae to calculate the metabolic cost of transport. It was ten times as high as assumed for running adult insects of same mass. From the point of energy efficiency this result is very poor.

	Cucullia verbasci	Cacoecimorpha pronubana	Pleuroptya ruralis	Tyria jacobaea
Body length [cm]	1.2	1.7	2.4	2.6
Stride frequency $[s^{-1}]$	2.0 ± 0.2	2.6 ± 0.3	1.7 ± 0.2	1.8 ± 0.3
Stride length [% body length]	0.26 ± 0.07	0.24 ± 0.06	0.25 ± 0.04	0.16 ± 0.02
Speed $\left[\frac{cm}{s}\right]$	0.62	1.06	1.02	0.75
Proleg stride duration [ms][1]	620		360	450
Proleg stride duration [ms][2]			200	
Proleg stride duration [ms][3]			140	
Proleg airtime in segment 6 [ms]	620		240	240
Proleg airtime in segment 7 [ms]	300		210	200
Proleg airtime in segment 8 [ms]	180		270	240
Proleg airtime in segment 9 [ms]	60		240	220
∅ Number of prolegs-in-the-air[4]	1.87		2.67	1.96
∅ Number of prolegs-in-the-air[5]	1.38		1.58	1.26

Table 2.1: Comparison of caterpillar kinematics parameters in forward walking of four different caterpillars. These values can be used to adjust locomotion patterns for artificial creatures according to real animal dynamics. (Gans [20], Brackenbury [8, 7], Belanger and Trimmer [4], Trimmer and Issberner [50])

2.4.3 Proleg movement

Prolegs of caterpillars, shown in figure 2.5 at Abd3-Abd6 are very important to avoid falling or rolling sideways while traversing small branches. For animals with cylindrical bodies it is very difficult to move horizontally without overturning. But both legs of one pair of prolegs are very close to each other, which prohibits walking stable on those legs. That is the reason caterpillars have strong gripping systems, allowing to move in any orientation on different substrates. Unfortunately gripping the ground in this way coincides with the loss of kinetic energy.

Role of prolegs in forward walking

In caterpillar kinematics prolegs play the most important role because they are the elements of caterpillars' body that exert forces to the ground. They move in a passive way

[1] while forward walking
[2] while reverse walking
[3] while reverse gallop
[4] while proleg stride
[5] in a whole stride

because they are connected to the bodysegments. One pair is placed at each of the segments 6-9 shown at figure 2.4 or at Abd3-6 in figure 2.5. Thus movements of segment cause movements of pairs of prolegs. Each pair of prolegs moves in phase. To provide enough friction with the ground that prevents from slippage prolegs use a passive grasping system. Gripping is performed by a kind of suckers. By bending segments of the body prolegs are lifted from the ground and later placed, a bit further in direction of movement, back on the ground again (Belanger and Trimmer [4]).

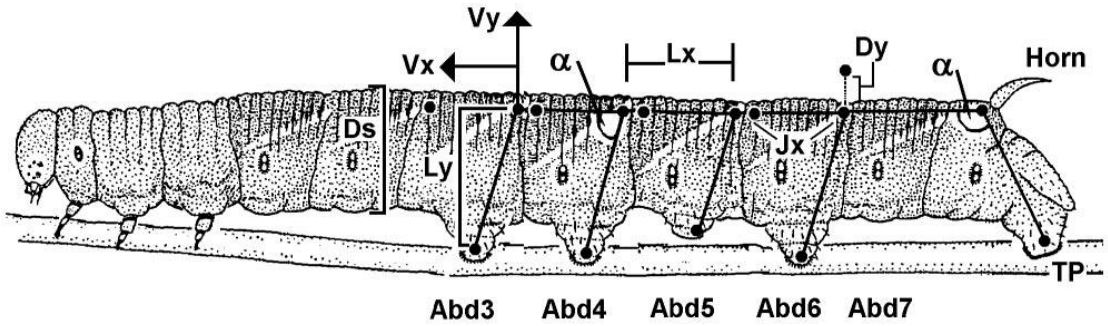


Figure 2.5: Physiology of *Manduca sexta* Larvae. Prolegs at Abd3 to Abd6 and the terminal proleg (TP) at the last segment build the locomotory system. While the terminal proleg uses active grasping to anchor the body of the caterpillar all other prolegs use passive attachment for locomotion. (Trimmer and Issberner [50])

In forward walking of *Cucullia verbasci* for a short time no proleg has contact to the ground just before before half of each stride is reached. At this point terminal prolegs[1] have to anchor to prevent backward-slippage. After that, when caterpillars start placing the prolegs back to the ground (from back to forth), the head is lifted from ground. Body segments stretch in direction of travel and one proleg after the other in reverse order reaches the ground again.
Another forward walking pattern is used by *Pleuroptya ruralis*. In the beginning of one movement cycle the terminal proleg anchors, then prolegs are lifted from back to forth to be placed a bit further in the direction of movement. After half of the stride all segments having prolegs are lifted to carry the current body wave to the head that lifts last. Because of the smaller amplitude of the body wave the length of the steps is shorter than in locomotion of *Cucullia verbasci*. Durations of single strides of the proleg movement are close to be only half-length in comparison to Cucullia verbasci. This results in much higher speed. Each pair of prolegs is nearly the same time off the ground and phase differences between neighbouring segments are nearly the same.
Tyria jacobeae performs quite stable walking. At any time at least one proleg is placed on the ground. Only ≈ 1.96 pairs of prolegs are in the air during the stride of the prolegs. Locomotion is generated by 1.32 body-waves. This is the upper limit that fulfils the stability condition. Overlapping of two body-waves occur by movements of head and tail that are partially simultaneous with the movement of prolegs.

[1] Terminal proleg (TP) is shown in 2.5 at back end of the caterpillar.

Physiology and proleg kinematics of Manduca sexta larvae

Trimmer and Issberner [50] clearly separate crawling of caterpillars from that of worms or molluscs. Series of steps taken by caterpillars are comparable to walking or running animals with stiff skeletons. They found out that the compression and extension of each segment are similar to harmonic oscillations in a spring. Some part of these movements were caused by folding the body wall between segments. Without fixed joints, soft-bodied animals are able to move in ways that are difficult for articulated creatures (Trimmer and Issberner [50]) or even artificial animals. Caterpillar locomotion uses hydrostatics and abdominal appendages (prolegs). Crawling can be regarded as directed two-sided from back to stomach wavelike bending (*bilateral anterograde dorsoventral undulations*) with at least two pairs of prolegs in continuous contact with the substrate. In this description terminal prolegs (TP) are included. It is necessary that at least two of five pairs of prolegs are grabbing substrate at any time to provide stability. Having a look at the duty factor induces walking gaits instead of running gaits in caterpillar locomotion. It is not possible to cycle the whole body between kinetic and gravitational potential energy as running animals do. But it seems that a mechanism to store and recover elastic energy exists. Unlike worms, caterpillars do not have circular muscles and there are no septa dividing the hemocoel. Trimmer and Issberner [50] suggested that caterpillars power their movements through longitudinal shortening rather than constriction and elongation used by worms.
Each segment contains about 70 distinct muscles, each controlled by one or occasionally two motoneurons without inhibitory motor units. Thus, most movements can be controlled by only few hundred motoneurons. The abdomen makes 75% of Manduca's body and provides principal means of locomotion. Abd3 to Abd6 are very similar and each have one pair of appendages (prolegs) that grip passively and can be actively unhooked and retracted. The terminal segment (TS) has one pair of specialized appendages, called terminal prolegs (TPs), with different muscles and innervation than normal abdominal prolegs feature.

Trimmer and Issberner [50] tried to describe the energy exchange in *Manduca*. Figure 2.5 shows necessary components needed to calculate relevant forces. D_y is associated with *gravitational potential energy*:

$$E_p = Mgh, \tag{2.11}$$

V_x and V_y determine *kinetic energy*:

$$E_k = \frac{MV^2}{2} \text{ and} \tag{2.12}$$

Elastic potential energy[2]:

$$E_s = \frac{Kx^2}{2} \tag{2.13}$$

In *Manduca* each segment can be described as a spring, moving along x-axis with E_s proportional to L_x, the range of the spring-like segment[3]. Movements of each body segment

[2] K is a spring constant

[3] This does not mean that a spring is a physical structure in *Manduca*, but that there is something that behaves spring-like.

can be regarded as a spring that is itself moving forward periodically. Hence differences in velocity between neighbouring segment borders provide simplified measures of spring properties of current segments.

Thoracic legs, in the first three segments directly after the head of caterpillars, are only used to steady anterior segments and are not involved in generating the propulsion itself. So kinematic analysis concentrates on abdominal segments. As shown in figure 2.6 each step in proleg-bearing segments consists of two phases, *stance phase* and *swing phase*. During stance phase prolegs are attached to substrate by cuticular hooks (crochets). In

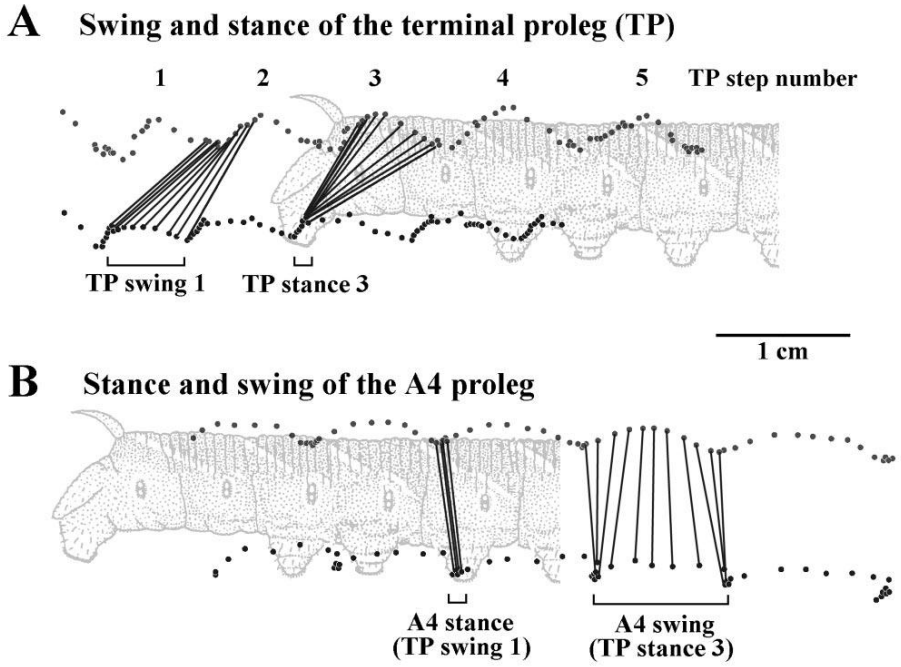


Figure 2.6: Swing and stance phase of *Manduca sexta* Larvae. (A) Different stages of swing and stance phases of the terminal proleg. Differences in motion of underside and backside of the skin, which occurs in alternating order, can be determined. (B) In contrast to the terminal prolegs common prolegs show different behaviour that can be described as folding and stretching movement of concertinas. (Trimmer and Issberner [50])

swing phase crochets were unhooked from substrate, raised and moved forward.
About three segments are in different stages of the swing phase at any time. Onset of swing phase in Abd3 coincides with onset of stance phase in Abd6. Average crawling velocity is $0.28 \pm 0.03 \frac{cm}{s}$. Average step period for prolegs in Abd4 is $2.91 \pm 0.09s$. Duty factor of prolegs in Abd4 is 0.53 ± 0.025 meanwhile duty factor in swing phase is 0.223. But terminal prolegs (TP), with a duty factor of 0.41 ± 0.029, show off very short swing phases of about 0.003. In midbody segments the waves of vertical displacement move forward with phase differences of $30 - 35°$ (about 0.24 to 0.29s) between each of direct neighbour segments Abd6 to Abd3 (Figure 2.7).

For each segment maximum shortening corresponds approximately to peak velocity in

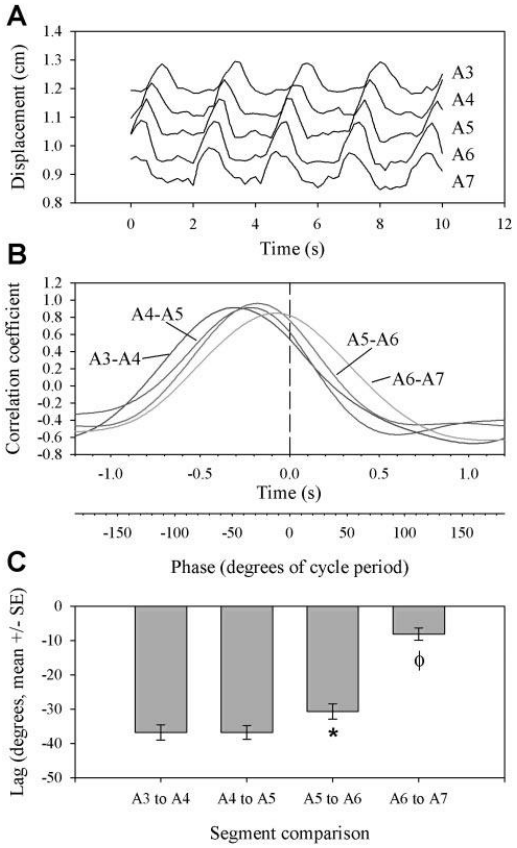


Figure 2.7: Phase Lag of *Manduca sexta* Larvae between neighbouring segments. (A) Displacement in centimeters of reference points representing body segments versus time in seconds is shown. Similarities but with little time shift can be seen. (B) Correlation coefficients of movements of segments with sinusoidal curves is quite strong and justify the usage of sinusoidal curves in locomotion generation of artificial creatures. (C) Average phase lags between neighbouring segments are described. (Trimmer and Issberner [50])

the middle of proleg swing phases. Changes in L_x were approximately in phase (within 11°) and negatively correlated with V_x. Resulting variations in length could involve body wall deformation and folding of the intersegmental membrane. Moving segments collapse leading edges into next anterior segments. This was confirmed by slow motion video. The main difference of the movement of the TP and midbody prolegs is, that in stance phase terminal segments continue moving. To achieve this, TPs rotate about 30°. In swing phase TPs rotate in direction against rotation in stance phase to produce cyclic behaviour. Duty factor of midbody prolegs is typical for walking gaits. But changes in vertical displacement and horizontal velocity are in phase like in case of running animals. This kind of motion agrees with interchange of kinetic and potential energy through storage and release of elastic energy.

2.4.4 Attaching to the ground

Caterpillars use special passive attachment systems providing stability while walking and climbing in arbitrary orientations. Figure 2.8 shows them in detail. There are crochets

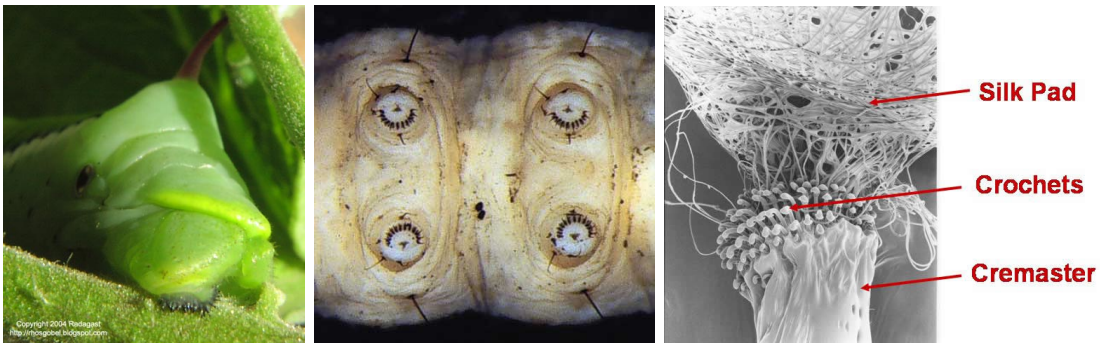


Figure 2.8: Attachment system of caterpillars. Left: Crochets at the 5-th hind end of manduca [31]; Center: Bottom-view of prolegs with cremaster; Right: Crochets of the 4-th abdominal segment attached on a silk pad [43].

at the tip of each proleg. These can be engaged or disengaged during crawling to grip substrate. In horizontal movement it prevents from rolling sideways. For creatures with cylindrical bodies this is very important to maintain stability when in locomotion. With left and right legs positioned near by near these creatures are at risk while moving e.g. on a horizontal branch to fall down. Attachment is so strong that crochets tear from prolegs before they loosen from substrate. Disengaging is caused by retractor muscles. prolegs are operated directly by a small number of motoneurons (one or two for each of the six proleg muscles). Sensory hairs (*planta hairs*) near the distal tip of the leg cause prolegs to retract. Muscles that are mainly used for retraction are the *principal planta retractor muscle* (PPRM) and the *accessory planta retractor muscle* (APRM). Belanger and Trimmer [5] have discovered that activity of these muscles is extremely stereotyped. Retraction of planta causes crochets to disengage from the substrate, while contact by planta with substrate causes them to hook into the surface. Prolegs have no extensor-muscles and are operated via combination of hydrostatic pressure and activity of intrinsic muscles. In mode of crawling prolegs work as claspers and the abdominal parts build locomotors.

2.5 Snake

Snakes offer amazing locomotion and climbing techniques. The black mamba, the fastest snake in the world, reaches speeds up to 20 $\frac{km}{h}$. This is quite impressive for a limbless animal and its' size. Even without any attachment system, snakes are marvellous climbers. They have a wide range of sensor- and force-feedbacked locomotion modes to overcome obstacles in elegant ways. The following sections describe the physiology of snakes and one of their simplest ways to generate propulsion, the *rectilinear movement*.

2.5.1 Physiology of snakes

Snake locomotion differs from that of invertebrates. As shown in figure 2.9 their physiology is different to that of worms and caterpillars. Indeed they have similar shape. The

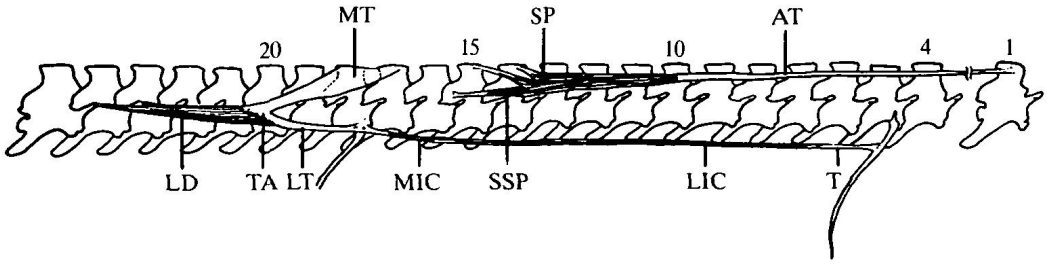


Figure 2.9: Anatomy of snakes. Backbone of snakes with position most important muscles and tendons with respect to enumeration of vertebrae is shown. Semispinalis (SP) and longissimus dorsi (LD) are very important muscles for locomotion of snakes. (Jayne [33])

main difference lies in rigid body skeletons, divided by vertebrae, which allows distinction between single segments easily. The number of vertebrae ranges from about 160 to 400 depending on the species. The size of snakes differs by species as well. Neighboured vertebrae are interconnected through muscles and tendons. Boid and colubroid snakes show different patterns of muscle interconnections. There is a major dichotomy in arrangement of muscles of primitive (boid) versus advanced (colubroid) snakes. Colubroids have different muscles according to their specialization for locomotion or constriction. Main differences are caused by different interconnection patterns of tendons between semispinalis (SP) and longissimus dorsi (LD) (see figure 2.9). Another difference is the number of vertebrae spanned by axial muscle segments and the relative proportion of tendon to contractile tissue within single muscles. Most snakes are able to switch their locomotion modes in response to resistive forces generated by interaction of snakes' bodies and current substrates. For snakes it is possible to use more than one locomotion mode at the same time at different body sections.

2.5.2 Rectilinear movement

From the control point of view, one of the easiest locomotion techniques of big snakes is rectilinear movement. It is quite similar to locomotion of caterpillars but with lower amplitudes. Rectilinear movement can be described as lateral bending of elongated body of snakes in a vertical plane. Locomotion is produced through rib movements and forms straight lines. Ribs are used to lift and to move forward successive sections of the ventral body surface as shown in figure 2.10. Travelling waves are passed posteriorly through the body while ribs are used almost like legs. There are cyclically anchoring parts of the skin at the ground using scales. By pushing the backbone forward and finally releasing the scales snakes are able to move forward [38, 20].
This technique is often used by big snakes, like boids. It is the slowest locomotion mode

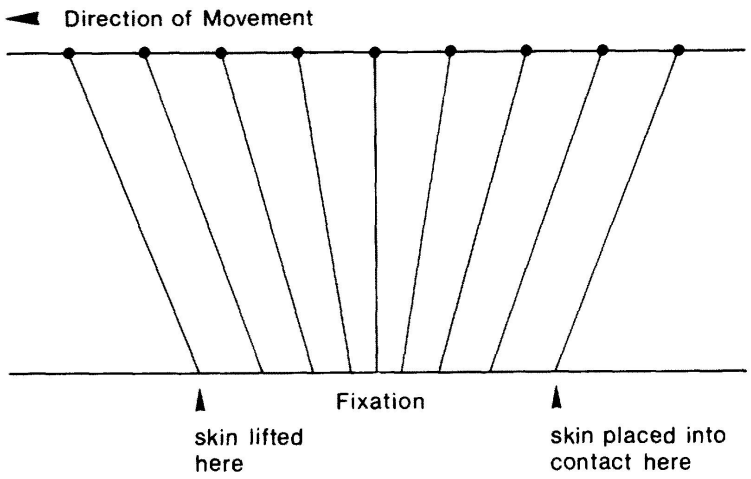


Figure 2.10: Principle of rectilinear movement. Rectilinear movement is based on coordinated shortening, lifting, lengthening and lowering of the skin at the downside of snakes' body. (Gans [20])

offered by snakes. Thus, the amount of frictional forces is much higher than the inertial component. Snakes utilize this gait to move in a straight line over the ground. In combination with fixed lateral bending in horizontal plane it is also possible to move along desired paths. Several bunching and fixation sites operate simultaneously to achieve continuous movement of the axial mass as the skin cycles. The middle of the back moves at constant rate.

2.6 Conclusion - Locomotion capabilities of caterpillars and snakes

Evolution attempts to improve physical and neural structures of certain kinds of creatures to guarantee the possibility of surviving generations. That does not mean that it creates generally perfect designs. This is not possible because every next step in evolution is based on past stages that can only be improved. So there is the need to examine what different creatures can perform extraordinary well and not to forget the backgrounds of evolution of these animals.

2.6.1 Habitats and domains of caterpillars and snakes

A dominating part of caterpillar's life is moving to places providing shelter and food. In fact they need to be capable of traversing unstructured and rough terrain with different properties of friction. Supported by kinematically effective and stable locomotion mechanisms, caterpillars are able to grow until they pupate and change their outer appearance. Caterpillars need escape strategies to get away from predators. Climbing is very needful for the caterpillar to reach leaves of plants to hide under their surface. To fulfil the requirements of a caterpillars life it uses an effective mechanism to attach to the ground. Like

that it enables the caterpillar to climb up food plants and to pupate. To move kinematically stable, caterpillars not only use well designed locomotion patterns but also benefit from attachment systems, located at the end of the prolegs, for grasping the ground in a passive way.

Indeed, snakes live in polymorphic environments. Typical environments are trees, mountains, water, desert, forest, caves and today there are also ruins, that snakes inhabit. In most cases it is irregular and difficult to traverse. Often there are small gaps, e.g. between a bunch of stones and rocks, large distances between branches in a tree. Additionally, snakes are limbless animals comparable to an articulated chain of hyper-redundant modules. They have to hunt and hide to survive. Snakes generate propulsion with the help of different gaits depending on the current situation. All gaits underlie the principle that a moving body propulses when generated forces in the opposite direction of travel are higher than friction in the moving direction. That also means that snakes try to produce high friction in the opposite direction of travel to be able to move. In general, snakes use travelling waves to produce rhythmic locomotion comparable to caterpillar movement or utilize force feedback to traverse narrow tunnels or to climb.

2.6.2 Summary of animal kinematics

To increase speed caterpillars increase the stride frequency. But in caterpillar movement it is not possible to increase the stride frequency beyond the speed limitation of muscular contractions or the segments will not have enough time to complete their cycle. In this case it would result in arching up the body, which leads to increased instability. Increasing stride length is another technique for increasing speed. But there is also a limitation, depending on leg length and maximum bending angle of the body. In case of caterpillars it means, that there are maximum bending angles between two neighbouring segments. When formulating caterpillar locomotion with the help of travelling waves this property can be described as *amplitude*. In forward walking proleg movement is limited, especially by stability conditions and physical properties of the body. Further improvements in efficiency of the movement can be done by manipulation of the duty factor. That means how many pairs of feet are simultaneously on the ground during a whole stride. A higher duty factor equals more contact with the ground, higher energy consumption because of the friction and slower movement, but more stable walking. It is good to find a duty factor as small as possible to increase the energy efficiency. But choosing too big duty factors would reduce the stability much. In mathematical models, describing this kind of locomotion, it can be denoted as *phase-difference* between neighbouring segments.

To save energy it is reasonable to use very smooth actuation patterns to make use of kinetic energy. *Cucullia verbasci* moves very inefficiently. One stride consists of arching up the body, after that follows a short break and finally a stretching of the body in direction of travel. This results in very high cost of energy.
In comparison with animals having jointed stiff skeletons, from the kinematics point of view, caterpillars move not very efficiently. Their benefit lies in simple kinematic control. It seems that the locomotion is not influenced by sensory feedback from interaction with the ground. Caterpillar locomotion shows no significant differences in kinematics concerning the orientation. Strong attachment systems are needed to realize this. Caterpillars'

prolegs need to be unhooked actively, while gripping the substrate works in a passive way. The final issue that affects speed, stability and energy consumption is the length of the body wave in contrast to the length of the body. It is determined by phase difference between neighbouring segments. To produce very smooth patterns of movement, phase differences between two neighbouring segments should be the same because normally each segment has same muscles. Equation 2.14 can be used to calculate the number of body waves. The equation uses degrees to calculate the result. Phase difference multiplied with the number of segments (M) results in the current number of body waves (k).

$$k = \frac{\Delta\phi \times M}{360} \tag{2.14}$$

Habib [24] describes that two body waves, in vertical-plane applied to a body with at least five segments, lead to stable movement. In this case the COG is at the same height all the time. Regarding only proleg movement there are three modules that can simulate four pair of prolegs, corresponding to the mid body segments. This mean, without having a terminal proleg the number of complete waves needs to be smaller than two. This can be achieved by smaller upper limits in phase difference. Alexander [2] points out that worms are too small and too slow to get influenced by inertia significantly. Because of the similarity in mass this should count for caterpillars, too. Building an artificial caterpillar that is much heavier than a real one implies to take inertia into account. In most efficient locomotion of caterpillars the head moves almost continuously as a result of the hydraulic pressure inside the body. Because of the high duty factor of the prolegs in caterpillar locomotion it is not possible to produce momentum to save energy. Instead each body segment must be accelerated from rest again, when the next body wave occurs. In comparison to stride lengths of vertebrates and insects of similar mass with solid skeletons, the stride length of caterpillars movements is only 25 to 30%. Even if the locomotion is low in energy efficiency, it provides a lot of advantages in traversing unstructured and rough environments, especially if combined with attachment systems.

In comparison to other chain-like animals with hydrostatic skeletons snakes move much faster. Despite of differences in size, stoutness, proportion of body length to number of vertebrae and proportion of body to tail length there are similarities in waveform and timing of muscle activities relative to vertebral flexion. Increased recruitment of muscles can compensate the increased number of body segments that needs to be used for locomotion. Rectilinear movement is very slow but useful to traverse some obstacles like small gaps or holes.

2.7 Summary

This chapter summarized theoretical foundations from animal research. Important terms of animal kinematics were explained in order to enable the design of efficient locomotion patterns for modular robots. Not only for designing new algorithms, but also for rating existing methods insights from this chapter are useful. The next chapter presents famous modular robots that benefit from principles of animal locomotion, applies knowledge from the previous chapter to modular robot kinematics and describes methods to generate and optimize similar locomotion patterns.

Locomotion of Modular Robots 3

The first section of this chapter introduces modular robots in chain-like configuration and describes famous modular robots from their first occurrence to the current state-of-the-art. At the end of the first section the simulated robots, used in experiments (described in section 6) of this work, are described. Section 3.2 presents a summary of important principles regarding the kinematics of modular robots. In section 3.3 common methods to generate locomotion patterns for modular robots are explained. Methods to optimize these locomotion patterns are presented in section 3.4. These optimization methods are integrated into the proposed system and were used in the experiments (described in chapter 6). Section 3.5 summarizes related work. The chapter ends with a summary of modular robot locomotion.

3.1 Modular robots

Inspired by amazing locomotion capabilities of worms, caterpillars and snakes, research of modular robots became popular since the 1980s. Very often the idea was to build up robots based on modules having all the same base structure. For certain tasks one single module is useless, but the combination of many of them creates structures that are able to fulfil these tasks.

3.1.1 Classification

Using different combinations of joints and elements that exert forces to the ground locomotion can be generated in many ways. Hirose and Yamada [27][29][30] present a classification of different kinds of chainlike modular robots.

1. active bending joint type
2. active bending and elongating joint type
3. active bending joint and active wheel type
4. passive bending joint and active wheel type
5. active bending joint and active crawler type

The way the locomotion is generated determines its efficiency in certain environments. This work focusses on the first category of limbless modular robots in chain-like configurations using active bending by powered joints. In principle these robots consist of rigid

links connected by 1-DOF rotating joints which are powered by servo motors. Joints in these robots can be oriented as pitching- or yawing-joint. Using both orientations together, modular robots can operate freely in three-dimensional environments. Some prototypes used at our department are explained by Zhang et al. [56]. Control of the servos is performed by micro-controller boards. More recent models used in our experiments have blue-tooth and separate touch sensor modules to realize wireless and intelligent control.

3.1.2 Application

There are several applications of modular robots in industry, surveillance, research and education. In industry they can be used in pipelines and sewers for tasks of inspection. Whereas in surveillance they are useful to gather information in unstructured terrain like collapsed buildings. In this way survivors can be found or useful knowledge about the current state of disaster sites can be gathered. For research they are of special interest. Biologically inspired control like CPGs (described later in section 3.3.2) can be studied as well as locomotion techniques seen in real animals can be applied. Biologists can test their hypothesises using robots and researchers related to robotics can use the results from biologists to create control strategies. Because of possible low cost architecture modular robots also suit very well as educational platform. Application of control algorithms can be exercised as well as the design of electrical circuits and programming of micro-controllers.

3.1.3 From early beginning to state-of-the-art

Shigeo Hirose and his team developed different prototypes of modular robots. In the early beginning, huge train-like machines shown in figure 3.1, like KORYU I (KR-I 1985-1992), were built. With the help of manipulators mounted on the top of some modules, it was

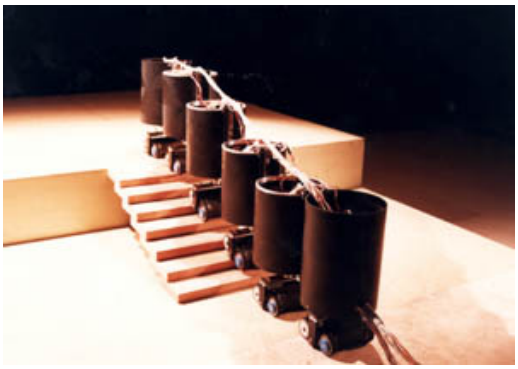


Figure 3.1: KORYU I is one of the first modular robots which redundant structure. By connecting several segments to a large chain it was able to climb stairs. In contrast single modules fail in this task. (Hirose [28, 29], Hirose and Morishima [30])

designed to fulfil assisting tasks in industry. The redundant structure using wheels at each module, provides the robot with the ability to overcome stairs. Indeed single modules were not able to climb stairs but the combination of several modules in a chain was able to

fulfil this task. Later, smaller robots were built. Some of them were able to imitate the locomotion of limbless animals while others utilize artificial mechanism like wheels or tracks.

ACM-R5 was presented at EXPO2005. It can be regarded as a snake-like robot and provides several features for usage in special terrain. The housing offers dust sealing and waterproofing to allow usage under any severe condition. It is composed of several rigid modules with small passive wheels. In this way it allows the robot to move smoothly on surfaces as well as swimming utilizing undulative movements in water. ACM-R5 is designed for inspection and search operations in underwater environments.

Figure 3.2: ACM-R5H is an amphibious snake robot. Efficient locomotion on land by using wheels is provided as well as swimming and diving in water. In both cases locomotion is generated by wavelike motions of the body. (HiBot [26])

Five versions of Soryu robot have been build in the last 20 years. Soryu-4, shown in middle of figure 3.3, is a modular robot developed to move in narrow and unstructured environments. It consists of three dust- and water-proof modules covered with tracks. These can adapt their posture, using special joints with high degrees-of-freedom, in order to overcome obstacles such as stairs, etc.. Joints allow to rotate single modules along the longitudinal axis to pass obstacles. Usage of different sensors and locomotion capabilities makes it applicable to urban, industrial, outdoor or unstructured scenarios, like search and rescue missions or remote inspection in potentially hazardous sites.

Figure 3.3: Series of Soryu robots consist of three modules driven by tracks. They are designed for tasks in rough terrain. Single modules can be autonomously disconnected or connected. These special joints are powerful enough to rotate or lift connected modules. (HiBot [26])

Johann Borenstein and his team created the four-inch[1] Omnitread, OT-4, that weighs around four kg and the eight-inch version, OT-8 with weight of nearly twelve kg. Joints are actuated using pneumatic bellows which are powerful enough to lift half its body off the ground. Propulsion is generated with the help of moving tracks at all sides of each module. Thereby OT-4 provides enough flexibility to manoeuvre in difficult terrain. They are intended to be used as inspector bots for hazardous environments.

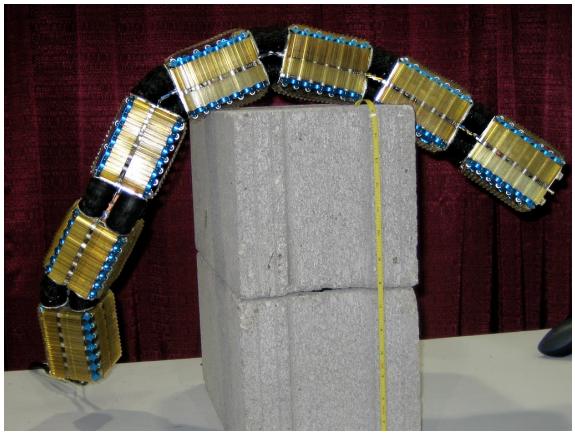


Figure 3.4: Each joint of Omni-Tread 4 (OT-4) is actuated by four independent pneumatic bellows. Locomotion is created by tracks. One of the design principles was to maximize the area of the robot covered by tracks. [44]

The of the AmphiBot-project (Crespi et al. [14]) was to build a biologically inspired amphibious snake-like (or eel/lamprey-like) robot. It is shown on figure 3.5. Goals were:

Figure 3.5: AmphiBot is based on hyper-redundant chain of identical modules but has some special parts to increase its capabilities. There are four legs and a tail. Inspired by salamanders this amphibious robot is designed to swim and crawl in a robust way. (École polytechnique fédérale de Lausanne [57])

[1] 4 inch ≈ 10 cm

1. Building an amphibious robot, inspired by snakes and elongate fishes such as lampreys, for outdoor robotics tasks.
2. Usage of the robot for testing novel types of adaptive controllers based on the concept of central pattern generators[2].
3. Usage of the robot to investigate hypotheses of how locomotion-controlling neural networks are implemented in real animals.

M-TRAN is a homogeneous modular robotic system that consists of several modules of the same structure that allow to be easily recombined, depending on the current task. In 2002 the second version of M-TRAN[3] (M-TRAN II), described by Kamimura et al. [34], was finished by Satoshi Murata and his team. The third version of MTRAN is shown on figure 3.6. A benefit of this famous modular robot is the automatic reconfigurability

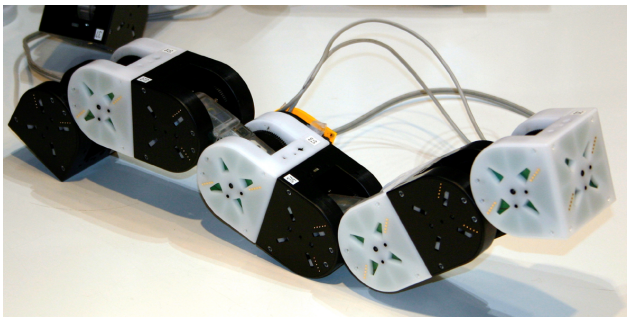


Figure 3.6: M-TRAN III is the third version of **M**odular **Tran**sformer. The limbless body generates locomotion with the help of waves travelling along the robot's body. These waves are generated by CPGs. The property of autonomous reconfigurability allows the robot to change its topology according to the needs to pass current obstacles. (Kamimura et al. [34])

that is supported by its active connection system, which allows to connect or disconnect several independent modules autonomously. By using cooperation between distributed autonomous sub-systems independent modules it is possible to change its shape to fulfil certain tasks like passing obstacles. The key concept was not only to create independent modules of similar structure that are all capable of moving independently but also to let them communicate and cooperate. Supported topological structures are chain and lattice. Locomotion control is implemented using CPGs (described in section 3.3.2). Depending on the current topology multi-legged walking as well as locomotion using body-waves can be used.

The latest contribution of our group TAMS at the University of Hamburg is the climbing caterpillar-like robot. Since van Griethuijsen and Trimmer [51] discovered that vertical locomotion patterns do not significantly differ from locomotion in the horizontal plane, similar methods for locomotion generation can be used for climbing. As part of the BICCA project (described in chapter B.2) a robot has been developed that uses an attachment

[2] As described in section 3.3.2
[3] Modular-Transformer

system based on the combination of passive suckers, high frequency vibrating motors and release valves. As described by Chen et al. [11] vertical locomotion on smooth surfaces like glass can be performed. This work extends previous research by Wang et al. [53].

Figure 3.7: Climbing caterpillar-like robot (Chen et al. [11])

3.1.4 Simulated robot models

Three different robot models, shown on figure 3.8 have been simulated. Depending on purposes of current experiments an adequate robot was chosen.
The left robot in the figure is a commercial product for educational purpose. The company *robotechn intelligent technology* created the *Cubo Robot* [47]. From the topological point of view it is very flexible. It can easily be reassembled and reprogrammed to move in different ways. By using small feet with rubber coating it is capable to produce enough static friction to avoid slippage on the ground when moving on plain surfaces.
The robot model in the middle of figure 3.8 shows a simplification of modular robots that are based on cubes. This very simple model has advantages in computing time. Because of its geometric simplicity the collision engine needs only few simple calculations when locomotion of the robot occurs. In this way it is ideal for testing new ideas.
The robot model at the right side of figure 3.8 is based on the robot Y1, described by Zhang et al. [56]. The three-dimensional model of the modules is taken from the OpenMR plug-in [22] for OpenRAVE. In addition this plug-in provides the system with a controller to simulate real servos that where used by all introduced robot models. By using the Open Dynamics Engine (ODE) physics plug-in in the OpenRAVE core, the simulation results gathered by the proposed system are effected by the physical properties of the robot parts like the shape of the chassis, mass and the controller's capabilities itself. Properties of controllers represent properties of real servos, like maximum speed, torque and angular limitations. Simulating sensors enables to implement intelligent control algorithms. For

achieving this, a touch-sensor plug-in has been developed as part of the proposed work. In this way it is possible to perform simulations of modular robots being comparable to real situations.

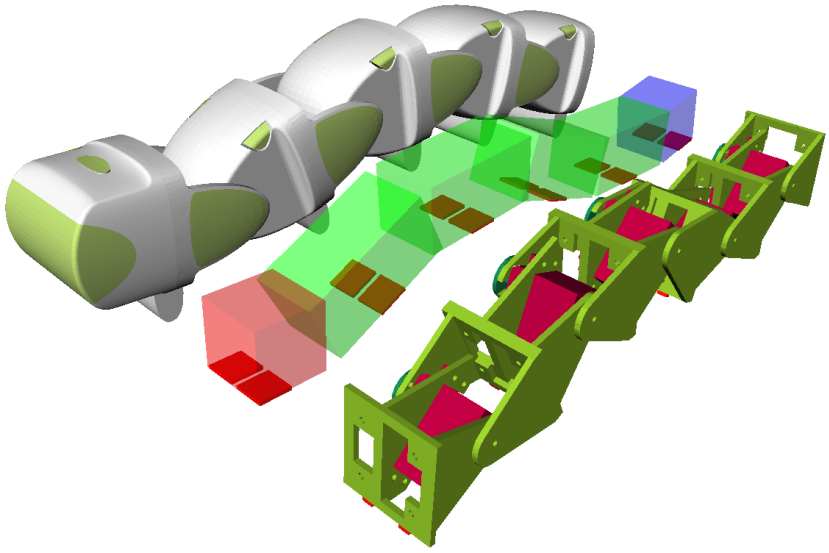


Figure 3.8: Three different robot models with similar structure were used in simulations for this work. Left: A detailed model of CUBO robot, a commercial educational robot platform (robotechn intelligent technology [47]) that can be reconfigured manually in very short time. It shows very good locomotional capabilities with very few slippage. Center: A very simple concept model of a modular robot is used for first evaluations of new ideas. Because of its simple geometrical structure it keeps the load generated by the physics engine on a very low level and reduces the time needed for simulations. Right: Three-dimensional model of a prototype with tactile sensors based on GZ-I (Zhang et al. [55], González-Gómez [22]).

These tactile sensors can be attached to any of the virtual robots introduced at the beginning of the current subsection. In figure 3.8 the robot model in the middle and the right one are prepared with touch sensors. Sensors are represented by red boxes on the lower sides of the robots. By using these simple touch sensors robots have the advantage to imitate the physical structure of the ground efficiently. By using the detailed robot model on the right, the computational effort to calculate collisions with environmental objects is very high and results in larger amount of time needed to finish simulations. Simulations using simplified models like the model in the middle are much faster but they capture only basic properties and do not take special designs of robot chassis into account.

3.2 Kinematics of modular robots

The key to let limbless modular robots travel in desired directions is to establish larger frictional force in the direction opposite to the direction of travel. Then, these robots are able to push themselves forward. Modular robots in chain-like configuration are able to

perform different gaits even without legs or wheels just by using techniques of limbless animals. Despite differences in body properties and sizes similar locomotion pattern as observed in nature can be used if the scale is considered (Quillin [46]). Main principles of locomotion of these robots are based on descriptions of animal locomotion, described in chapter 2. A summary of kinematic analysis of modular robots in chain-like configuration is given in the following.

3.2.1 Principles of modular robot locomotion

In limbless locomotion at least two points of contact with the ground are needed to move relative to the ground. These locations are called *supporting points*. From analysis of the kinematics by González-Gómez et al. [23], Zhang et al. [55] follows that the locomotion is stable when the projection of the COG lies on a point within the parallel line of the connection of the supporting points of the robot. This is the *stability condition* for chain-like modular robots without strong gripping mechanisms.
To save energy, rules of inertia and friction can be considered to create energy efficient locomotion patterns. Therefore the generated movement patterns should be as smooth as possible.As described in chapter 2, slow movements do not allow momentum to be created, but inertial forces can be neglected when locomotion takes place. The smoother the body of the robot moves the more its inertia can be disregarded. In this case governing forces in locomotion creation results from static friction with the ground. For light-weighted robots this can be advantageously.

3.3 Locomotion generation

There are several strategies to generate locomotion. In general, smooth waves or fixed patterns are used to generate motion of robotic chains. Another method to create efficient locomotion that uses combination of fixed patterns is presented by Yamashina et al. [54]. A database with possible transitions between different robot states is used there to generate locomotion. The choice which transition should be applied is optimized by a reinforcement learning method (*q-learning*). Creating locomotion by this is totally different to former techniques. This work focusses on smooth waves created by generators that produce periodic output. To create travelling waves, resulting from body waves of modular robots, sinusoidal generators as well as CPGs can be used. Meanwhile sinusoidal generators exhibit only similarities in the output signal in comparison to animal locomotion, CPGs also share the principle how the output is generated. Both categories are described in the following sections.

3.3.1 Sinusoidal generators

The usage of sinusoidal generators with the intention to generate locomotion of modular robots is presented by González-Gómez et al. [23], Zhang et al. [55]. Equation 3.1 and

table 3.1 are both taken from [23].

$$\varphi_i(t) = A_i \sin\left(\frac{2\pi}{T_i}t + \phi_i\right) + O_i, \qquad \in \{1 \dots M\} \tag{3.1}$$

The use of sinusoidal curves is recommended for low computational power applications. The output is cost efficient to calculate. Higher level parameters, as shown in table 3.1 allow to modulate the locomotion very easily. Recordings from real caterpillars in figure 2.7 on page 18 prove the usage of sine-shaped curves in animal locomotion. The distance

Symbols	Descriptions	Range
$\varphi_i(t)$	Bending angle of module i	$[-90, 90]$ degrees
A_i	Amplitude of generator i	$[0, 90]$
$T_i(t)$	Period of generator i	Time units
$\phi_i(t)$	Phase of generator i	$(-180, 180]$
O_i	Offset of generator i	$[-90, 90]$
M	Number of modules of the robot	$M \geq 2$

Table 3.1: Parameters of sinusoidal generators. They can be used to modulate the output of the sine function. Changing these parameters influence speed of travel, energy efficiency, position of the COG and smoothness of the locomotion, step size and stride frequency.

Δx travelled in one period along the x axis can easily be calculated with the help of equation 3.2 described by González-Gómez et al. [23]. L_T is the total *length of the robot*. The *number of complete waves* is denoted as k and the *wavelength* as λ.

$$\Delta x = \frac{L_T}{k} - \lambda \tag{3.2}$$

This means that increasing the amplitude while maintaining the frequency results in larger distance travelled in the same amount of time. This forces faster rotational speeds in the robot joints and increases the height of the center of gravity of the robot. The frequency correlates with the travelled distance per time in a direct way. It is limited by the maximum rotational speed of engaged motors. The phase difference between neighboured modules is the most important dynamic parameter. It determines the number of waves that will be used by the robot. According to the stability condition, presented in section 3.2.1, the phase is important to be well chosen.

3.3.2 Central pattern generators

The creation of rhythmic locomotion patterns as a result of neuroscience and robotics can be carried out with CPGs. From the biologists point-of-view CPGs are organized as coupled bursting elements. There is at least one unit per articulated body element.

Each unit can be divided further into oscillatory centres for flexor and extensor muscles. As explained by Ijspeert [32] in nature CPGs find application in activities like chewing, breathing and digesting as well as building blocks in locomotor neural circuits.
Inspired by neural circuits networks producing smooth regular curves, locomotion can be created. Similar to sinusoidal generators (described in section 3.3.1) travelling waves are generated by rhythmically actuating the joints of a robot. The amazing aspects are that CPGs are able to produce rhythmic output signals without receiving rhythmic inputs and have self-stabilizing abilities. Input from sensors is not needed for the process of creating rhythmic patterns, but it is used to modulate the output. Locomotion generation can benefit from simple sensory input by causing gait transitions in dependency from the magnitude of the signal. Especially for maintaining movements coordinated sensory input is used.

In robotics several CPG models were used to control swimming, walking and creeping robots. Models integrating sensor feedback, turned out to be the most robust ones. Ijspeert [32] present an overview about CPGs in locomotion control of animals and robots. Depending on researchers' focus there are different categories of CPG designs. It determines which components will be modelled.

- biophysical system
- connectionists' model
- system of coupled oscillators
- neuromechanical simulation

CPG-nets can be created by connecting several oscillators. Important oscillators are the half-center model, where two populations of neurons are mutually coupled with inhibitory connections that use fatigue mechanisms. These fatigue mechanisms like leaky-integration are used to create the dynamics in the system. Another famous building block is the

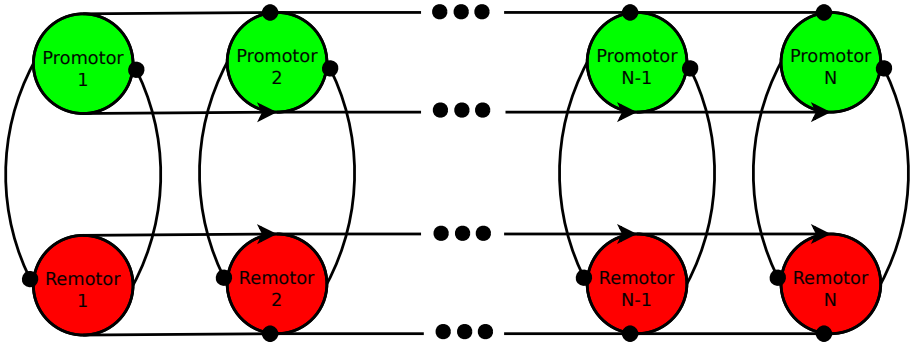


Figure 3.9: CPG network according to push-pull model introduced by Herrero-Carrón et al. [25]. Oscillators are composed of promoter and remotor neurons which are interconnected by inhibitory synapses bidirectionally. Unidirectional inhibitory and excitatory connections to next neighbours of promoters and remotors allow the whole net to synchronize. In this way a strong self-stabilizing network of oscillators is build that produces efficient locomotion patterns.

matsuoka oscillator, described by Matsuoka [41]. To calculate the state of CPGs-networks dynamically numerical integration is used to solve coupled differential equations.
Figure 3.9 shows a CPGs-net with a topology according to the push-pull model presented by Herrero-Carrón et al. [25]. It consists of a chain of oscillators connected by excitatory and inhibitory synapses. Each oscillator is composed by two different kinds of neurons, promotor and remotor neurons. These are connected by inhibitory synapses. Inhibitory synapses prohibit neurons, connected to them, to fire at the same time. Thus a very stable and rhythmic behaviour is achieved. Synaptic connections between neighbouring oscillators determine dynamic characteristics of the whole network.
Two other CPG models were successfully used by Li et al. [37, 36] to generate robust locomotion patterns.

Advantages:

1. limit cycle behaviour → robustness against perturbations
2. well suited for distributed implementation
3. smooth transitions when modulation occurs → differential equations act as filters ⇒ avoids damage in motors
4. integration of sensor feedback as coupling terms in differential equations possible
5. well suited for learning and optimization algorithms

Disadvantages:

1. higher computational effort for calculation of neural networks
2. missing methodology for designing CPGs
3. no theoretical foundation
4. sometimes many control parameters with complex correlations
5. proves of stability are difficult

3.3.3 Adaptive actuation

In addition to algorithms producing driving, adaptive mechanisms, it can be used to change the basic shape of the robot. Sensor feedback is needed to overcome obstacles and evade objects blocking the direct way to a destination. Adaptive methods using sensor feedback can be integrated into locomotion algorithms directly. Having a look into nature we can examine that caterpillars react to stimuli on their head with a change in the current gait. Brackenbury [7] analyses fast backward movement and concludes that it is a response to stimuli. This kind of behaviour is produced by a CPG-based control system that integrates sensor feedback directly like Kamimura et al. [34] present. But it is sometimes more easy and flexible to extend existing systems with adaptive offset controllers. A sketch of this approach is shown in figure 4.6.

Based on the topological control principle described in section 4.3.1, adaptive control modules can be added to locomotion modules to improve the locomotion capabilities of the current robot.

3.4 Optimization of locomotion

As described the proposed system can be used to implement arbitrary forms of locomotion. The control architecture allows to combine several actuation modules into behaviours e.g. fast crawling and adaptive crawling. After a behaviour has been created, the problem of finding optimal parameters arises. In this chapter methods that are integrated into the simulation system are described to find good solutions for certain sets of parameters. A short introduction into some theory to analyse the complexity of the parameter optimization problem is given. Then, after an introduction into the optimization methods itself, set-ups are described as case studies that were optimized with the help of these algorithms. In each experiment predefined sets of parameter values are going to be improved to achieve efficient locomotion with different focuses. To make the results comparable the same set-up of the environment is used to be optimized under different methods.

3.4.1 Analysis of parameter optimization problem

The concrete problem that has to be solved is composed of a chain of parameters determining the locomotional behaviour of a modular robot. Each parameter can be applied to one value of its domain. To find adequate solutions and to compare results from different optimization techniques an analysis of the solution space is necessary. Because of the high order dimensionality (see equation 3.3) it is very difficult to analyse the problem. Defining approximations of the distribution of the solutions of the problem is challenging because the distribution of weighting functions, rating single solutions, are mostly non-linear.

Complexity

Depending on the formulated problem different degrees of complexity arise. The number of parameters that are involved and the size of their domains determine the complexity of formulated problems.

$$O(n) = \prod_{n=0}^{\text{numOfParameters}-1} |\text{valueSet}_n| \triangleq a^n \tag{3.3}$$

Runtime

The runtime of optimization processes depends on the complexity of the current problem in combination with the number of evaluations needed by the optimization and in the end, on the runtime of a single simulation round either. Finding an optimal or at least good solution usually means repeating experiments until the convergence of the resulting solution's quality is given.

$$\text{maxTime} = \text{complexity} \times \frac{\text{time}}{\text{performance}} \tag{3.4}$$

In equation 3.4 `time` means the amount of real time that needs to be simulated. This can be the time given to overcome a certain obstacle. How much computing time it takes to simulate this period depends on the performance of the system. Thus the resulting time needed for finishing one simulation run can be calculated by taking into account the performance factor of the current machine running the simulation. In this case the *performance* is defined as *fraction of realtime*.

3.4.2 Description of optimization techniques

In selection of optimization methods it has to be considered that appropriate weighting functions are normally non-linear. By choosing appropriate optimization techniques it should be avoided to get stuck in local maxima of solution spaces. Heuristic methods seem to perform very well in this case, although global search for good solutions should be applied by selected optimization methods. For application it is not necessary to find the best solution. One reason for this is caused by rounding error of physics engines. Especially calculating frictional forces is highly complex and can not be done as accurate as needed to perfectly match reality.

Genetic algorithms

Genetic algorithms (GAs) can be regarded as stochastic optimization methods. They evolve one generation after the other meanwhile each generation is based on the previous except the first one. Each generation holds a population of individuals where each individual represents one complete solution of the formulated problem. After one individual is evaluated it gets rated. The idea is to generate following populations based on the fittest individuals of the previous one by using only simple combination and manipulation methods shown in figure 3.10 and 3.11. Termination occurs when scores are converging, a certain number of generations is evaluated or a predefined limit is reached by at least one individual. GAs are recommended for optimization of CPGs by Ijspeert [32]. The advantage is that even cost functions, which are neither continuous nor linear, can be optimized in this way. Unfortunately, the runtime can be very large depending on the characteristics of certain problems that need to be optimized and the configuration used to adjust the GA as explained by Goldberg [21].

$$O(n) = \prod_{n=0}^{\text{genomeSize}-1} |\text{alleleSet}_n| \triangleq a^n \tag{3.5}$$

$$\text{duration}[sec] = \text{numOfGenerations} \times \text{populationSize} \times \frac{\text{time}}{\text{performance}} \tag{3.6}$$

It can be very useful to analyse the problem space before parameters defining characteristics of the current GA instance are going to be adjusted. But in general evolutionary algorithms produce good results if the cost of time is of less importance.

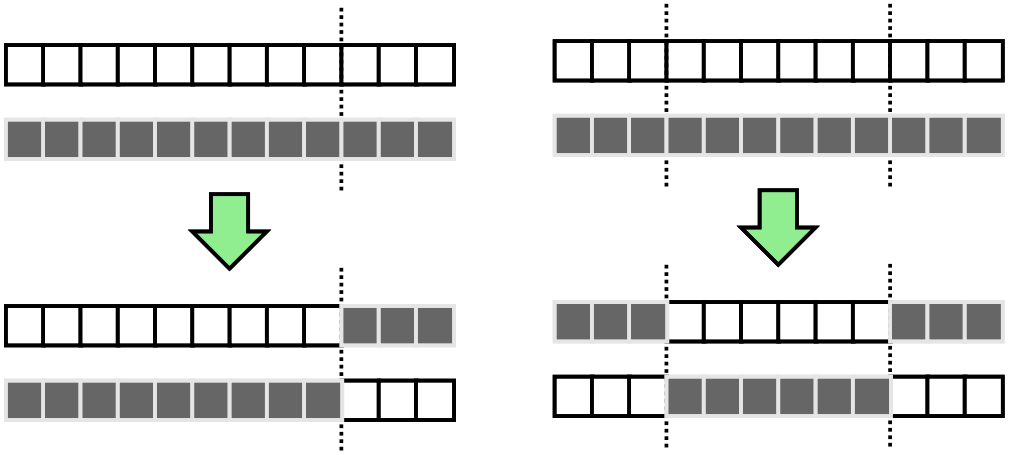


Figure 3.10: The principle of genome operation *crossover* is to swap sequences of genes between two genomes. Two genomes are necessary to perform this operation.

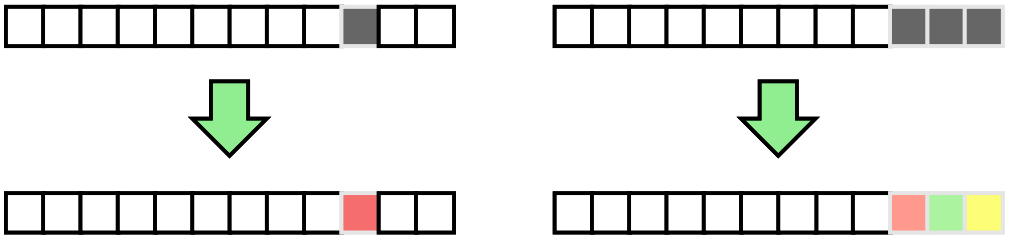


Figure 3.11: The genome operation *mutation* changes the values of a single gene or a sequence of genes. Only one genome is needed for this operation.

Modified great deluge algorithm

As a representative of classical approaches to solve optimization problems the *great deluge algorithm (GD)* has been implemented. The complexity of problems being solved by this implementation can be calculated using the more general formulation using equation 3.3. According to the problem space analysis in section 3.4.1 the algorithm was modified to avoid to get stuck in local maxima. The idea of the classical GD algorithm is shown in figure 3.12. Initially, the algorithm tries to find a solution that satisfies a very low condition, called *water-level*. Every time a solution is found, it increases this condition by a given amount, named *rain*. To find a proper candidate the algorithm searches in the neighbourhood of the last accepted solution. The idea to overcome the problem of local maximum is to use dynamically calculated neighbourhood definition. In this way the radius of search is increased more and more the longer no proper solution is found after a fixed number of retries. If no satisfying solution for current water-level is found, after a fixed number of allowed tries evaluated, the algorithm stops. For further improvement the algorithm offers itself several new chances when failing. This means after a fixed number of fails it resets the search radius to initial state and proceeds as normal. These new tries can be started by consuming *credits*. The algorithm terminates if after a certain number

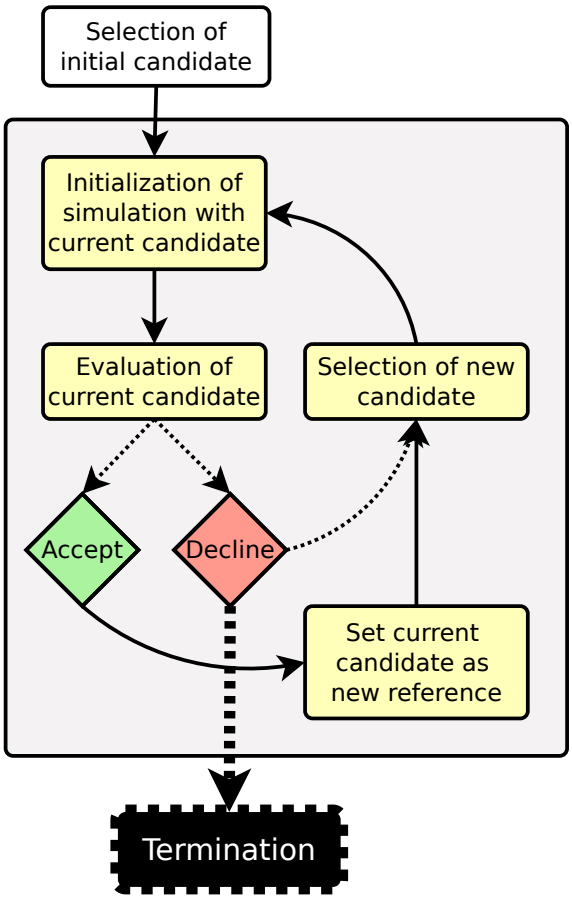


Figure 3.12: The principle of classical great deluge algorithm (GD): In each iteration a candidate in the neighbourhood is selected randomly and evaluated. If the results do not satisfy the current requirement (water-level) then the algorithm continues with selection and evaluation of the next candidate. If a candidate is accepted the water-level is raised before selecting the next. Termination occurs when a desired score is reached or a certain number of unsuccessful iterations is reached.

of evaluations no candidate was accepted and no credit is left to reset search. If the number of allowed retries per chance is reached, one credit is consumed and the number of used retries is set to zero again. The *number of used retries* together with the *confidence* parameter determines valid range of neighbourhood. Confidence of twenty means that every twentieth simulation round when no valid candidate was found, the step-size used to modulate current candidates is increased by itself. But after a valid neighbour that satisfies the current condition of water-level is found, the number of retries is set to zero again and in this way the step-size is set to its initial value again. Adjustments of initial water-level, allowed number of retries, number of credits and confidence parameters define the optimization behaviour and efficiency of this algorithm.

Fitness function

Both optimization methods described above need fitness functions to rate their candidates. To generate results that are applicable for different purposes, several fitness functions are needed. After each evaluation the resulting state gets rated by one of them. To rate results of experiments, that are part of this work, a very simple fitness function has been implemented.

$$\text{currentScore} = \sqrt{(\text{xStart} - \text{xEnd})^2 + (\text{yStart} - \text{yEnd})^2} \tag{3.7}$$

It simply calculates distances between starting point and last position of currently used robots. Fitness function 3.7 was intended to be used for producing results with largest displacement in given time. Because of randomness, that is always part of collision checking in physics engines, there is some inaccuracy. In repeated experiments with exactly same settings, deviations around arithmetic average in results of the described fitness function ± 0.15 metres were observed. Using the fitness function above results are governed by an inaccuracy of $\approx 5 * 10^{-5}$ each step taken by the simulation system. Depending on current goals, any other fitness function can be used to obtain results with different focus from simulations.

3.5 Related work

Many researchers invested much effort in pattern generation for locomotion of modular robots. Often they use optimization methods and simulations to find insights for the application of locomotion patterns to real robots. Marbach and Ijspeert [40] used GAs in simulations of modular robots with different topologies to evolve efficient locomotion patterns. In Habib [24, chap. 7] kinematic analysis of modular robot locomotion is done in detail and the results are formulated to principles. These were tested in simulations with the ODE (Smith [49]) physics engine. With the help of the simulations they were able to record data in order to apply comparative analysis. Another work uses Q-Learning, an unsupervised reinforcement learning technique, to select stepwise locomotion patterns from a database of locomotion patterns (Yamashina et al. [54]). In this way locomotion is not generated by a function, when needed, but predefined by stored sequences of selections from a database of patterns. The robot itself has to generate transitions between applied patterns from the database.
Kamimura et al. [34] created their own simulation software for optimization of locomotion for their famous modular robot M-TRAN II. They integrated the model of the robot into a 3d-library system and optimized the CPG-driven locomotion with the help of GAs.
A similar combination of hard- and software was done by Daidie et al. [16]. They created an educational robot module that can be combined with others to create a modular robot with hyper-redundant structure that is able to move. They included software to simulate and program built robots.
All of them produced good results for their research but they all created systems for special cases: their own research. There are other works on simulation systems that are more general and not special purpose systems. Webots (Cyberbotics [15]) is a commercial

software that uses ODE (Smith [49]). It is part of the Nao (Aldebaran [1]) development kit for research and education. Webots is very general and has great capabilities in creation of complex environments like living rooms or buildings in general. They also have examples regarding modular robots but there is no further support that would make the development of locomotion patterns easier. Optimization methods are not integrated and must be implemented by users, if needed.
OpenRAVE (Diankov [17], Diankov and Kuffner [18]) is an open-source software for planning, simulation and control of arbitrary robots (see chapter 5.1). It is widely used and offers good support in case of questions and feature requests. It is a very general software, but it is designed to be extended very easily. With the help of plug-ins the system can be extended with many desired functions afterwards. González-Gómez [22] created a plug-in for OpenRAVE that contains one modular robot model and a servo controller plug-in (see chapter 5.1). The proposed work combines the flexibility of OpenRAVE by using plug-ins like OpenMR and wraps it into a framework for modular robots. As described by Krupke et al. [35] the proposed work tries to fill the gap between very general open simulation systems and closed implementations of simulation software for specific modular robots.

3.6 Summary

This chapter introduced modular robotic locomotion to the reader. Famous prototypes were presented and after an explanation of their kinematics, methods to generate bio-inspired locomotion patterns like sinusoidal generators and CPGs are explained. In addition this chapter presented methods like GAs and the modified GD algoritrhm to improve the locomotion. The presented methods of pattern generation and optimization are part of the proposed work and are partially used later in the presented experiments in chapter 6. Section 3.5 summarized related work about generation and optimization of locomotion patterns for modular robots and justifies the development of the proposed system. The next chapter presents the proposed system that is used in the experiments (see chapter 6) to optimize locomotion patterns of simulated modular robots in special situations.

Simulation System Description 4

This chapter presents the proposed system for modular robot locomotion generation and optimization. As a fully integrated system it offers also necessary components to create configuration files for the robot, the sensors, the locomotion, the environment and the optimization of control algorithms.
Section 4.1 and section 4.2 present the concept and structure of the system. The core mechanisms are explained in section 4.3. In this section important concepts that make the system that flexible as it is are clarified. Benefits in the usability result from two different GUIs. Section 4.4 presents the configuration interface and section 4.5 presents the control interface. They contain descriptions of their main functions and some hints how to use them. In the end of this chapter a summary is given.

4.1 Introduction of the simulation system

The proposed modular robotic environment is intended to be easy to use, even for people with only few knowledge about modular robots' control. In this way its usage as an educational tool is possible. Furthermore, it is designed to work as a framework in research to investigate the robust generation of locomotion patterns for modular robots. New locomotion methods can easily be implemented, evaluated and optimized. The user can benefit from passed simulations by re-using already configured parts in next simulations. This is very comfortable e.g. when comparing different robots in the same environment.

It allows fast creation of several robot configurations including sensor placement and definition of the control behaviour. The behaviour is defined through the combination of several algorithms which are applied to groups of joints. New control algorithms for locomotion, adaptive behaviour using sensor feedback or additional special tasks can be implemented with the help of the configuration user interface, any C++ development environment or text editor. The integrated environment editor allows to create the workspace of the robot which is needed to perform some experiments. Building environments can be done by drag-and-drop from object library. Configurations are stored in XML format to allow re-usage. In this way the configuration is not only human readable but also editable by hand retroactively. The structure of the configuration files can be seen on figure 4.1.

The structure of simulations from the simulation core's point of view is illustrated in figure 4.2. In principle robots moving in specific environments need to be simulated. The environment consists of several objects and its physics underlie some well defined laws.

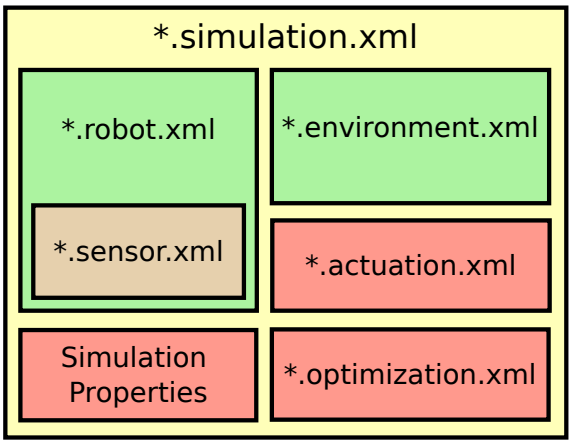


Figure 4.1: Structure of configuration files. Simulation configuration files include independent configured parts, like the robot or the environment. In this way components can easily be exchanged or reused.

Robots interact with the environment following the present physical laws. Every robot consist of rigid links, rotational joints and optional sensors. For generating locomotion robots' joints show off autonomous behaviour. All autonomously actuated joints together can be regarded as locomotion system.

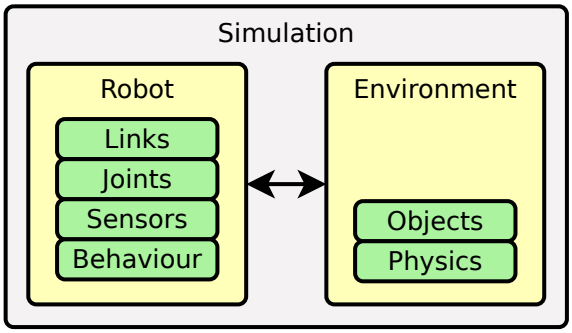


Figure 4.2: Overview of simulations. In simulations arbitrary configured robots interact with objects inside environments according to rules defined by the physics engine. Part of robots are the visible parts of their body as well as their behaviour and their sensing capabilities.

The Information given by the configuration files is enough to run a proper simulation for the purpose of evaluation, demonstration or optimization. Once the simulation environment is initialized it can be supervised or manipulated using the components of the graphical control interface or by code. When running in command line mode only manipulation by code is possible of course.

4.2 Overview

Figure 4.3 shows the main components of the system with high granularity. There are two different kinds of graphical user interfaces. Not only a configuration but also a control GUI allow easy use of the system. A container object holds the simulation environment, including the OpenRAVE (described in chapter 5.1) core itself, as well as the control kernel, several data handlers and implementations of real robot control modules. Input and output of configuration and data files is performed by special objects.
Using configuration GUI files all settings for one specific simulation can be written to disk. Extracting information from these files provides the system with everything needed to properly initialize the simulation container with all its objects.
The control GUI allows to supervise and manipulate the objects held by the simulation container. A 3d viewer can be used to watch the simulated environment. It is also possible to change the pose of single objects or to change the point of view.

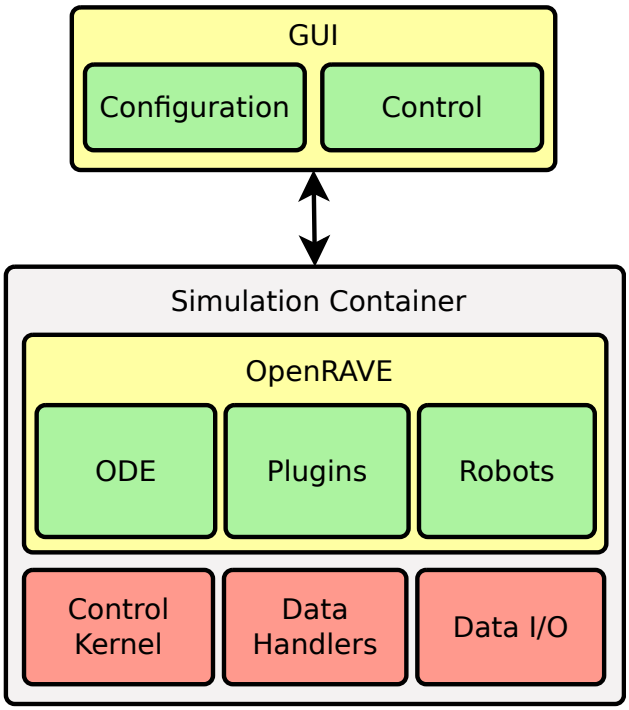


Figure 4.3: Main components of the system are the GUI and a simulation container. The GUI is divided into two different parts, one for configuration and one for control. In the container of the simulation the OpenRAVE-core is running as well as the control-kernel that controls the robot and all data handling.

4.3 Core-system

The core is based on the simulation container object shown in figure 4.3. As an independent unit it runs without the need of graphical components. But it is able to communicate

with the control GUI optionally. It is bidirectional communication that allows not only to represent the current status of the simulation in different ways using the GUI but also the possibility to manipulate the simulation by actions of the user. The control kernel holds all algorithms assigned to the current robot that actuates its joints. Data handlers for each kind of data are able to catch every produced value and pass it to file writers or to the GUI for visualization purposes. Information from the physics engine, robots, environments and actuation modules can be handled properly by using these objects for data handling. When needed, the data handlers can pass the data to the I/O component for writing result files.

$$t_{simulation} = t_{old} + \delta * \text{duration}_{step} \tag{4.1}$$

In addition there is a simulation clock that allows to make quantitative statements about the simulation. The system implements continuous simulation and will be done step by step. The simulation time of the current step is calculated by equation 4.1 where δ is the sampling factor of the continuous simulation. The sampling factor is of high importance for the accuracy of the physics engine and related parts of the system like OpenRAVE-core, sensors and locomotion algorithms. To calculate the current simulation time, the time needed to calculate the current step needs to be taken into account. This depends on the simulation set up, can vary from step to step and is highly correlated with computer system's performance.
To allow slow-motion simulations *delay* μ has been added to equation 4.1.

$$t_{simulation} = t_{old} + \delta * \text{duration}_{step} + \mu \tag{4.2}$$

Using equation 4.2 simulation and control of real robots can be slowed down and system load can be reduced by increasing the value of μ.

4.3.1 Control kernel

To configure the control of arbitrary robots in a very flexible way a control object has been developed. Figure 4.4 shows a scheme of the control kernel that keeps all actuation modules assigned to the current robot. The main purpose of the control kernel is to generate the control signals for the joints of the robot. This works by summarizing the output of every single actuation algorithm's output joint-wise to one list of output signals. It can also be used to modify any parameter of the algorithms by code or by GUI at the runtime of the program. Depending on the chosen interface for implementation of the actuation module it has access to sensors connected to the robot in addition. The

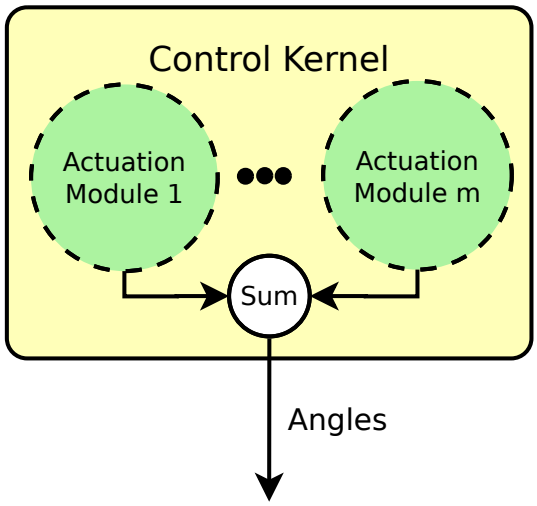


Figure 4.4: The control kernel contains all algorithmic modules calculating new desired positions for the joints of the robot. Several modules can easily be combined by summarizing their output. Using this technique simple locomotion can be extended with adaptive algorithms using sensor feedback.

integration of sensor feedback for adaptive algorithms is possible using this feature.

while *running* **do**
  `ReadSensors` ();
  for $i \leftarrow 0$ **to** $|ActuationModules|$ **do**
    `CalculateAngles` ();
    angles $\leftarrow$ `AddAngles`();
  end
  `SetServos` (angles);
end

Algorithm 1: Calculation of angular output

4.3.2 Data logging

All kind of data is managed by a group of data handlers as shown in figure 4.5. Information from simulated environments[1], simulated sensors, control kernel and real robots is gathered and can be acquired by data file writer or control GUI. The file writer takes all unwritten data and flushes it to an XML file. The control GUI can ask for specific data lines to display them in the GUI directly.

For every different kind of data there is a specialized handler class. OpenRAVE environments and robots are passing information like position of the robot, movement speed, energy consumption and many more to their handler. For every actuation module in the control kernel, one data handler for catching calculated values is assigned. In this way

[1] robot and environment in OpenRAVE

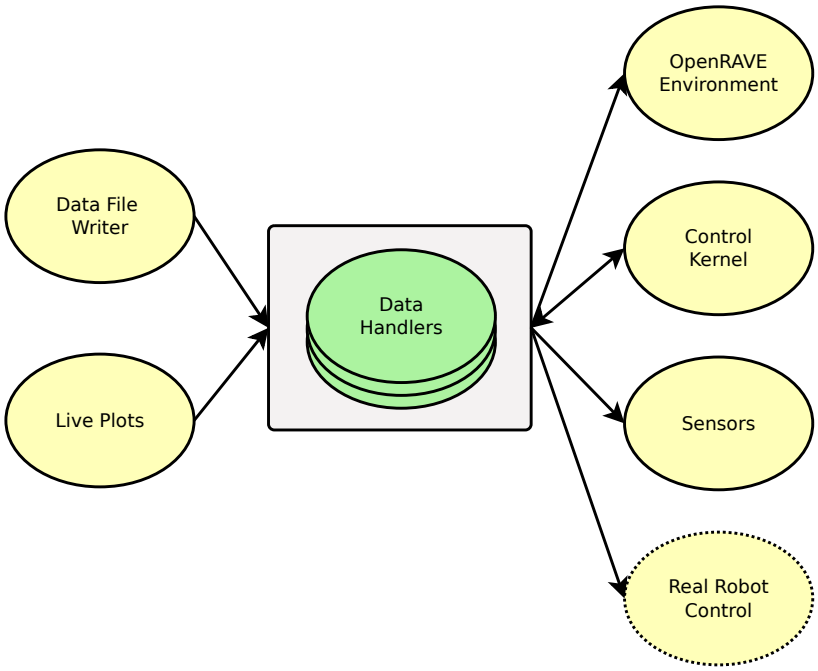


Figure 4.5: Data management is performed by several handler objects. Calculated data is appended to their buffers for live data observation and writing to files in a separate thread.

every value taking part in changing the robots' joint angles is buffered. For efficient realization of this feature, the implementation of a circular buffer from the boost library is used. One specialized handling object is used for real robots.

4.3.3 Simulation modes

By defining the simulation mode the behaviour of the simulation core is determined. For example there are some experimental settings stored in a mode that has only one purpose: to run a GA with a special configuration of robot and its actuation. But there are also some configurable modes for locomotion optimization purposes. It is possible to start a real robot control loop by selecting the right mode or for running the simulation loop in the most simple way.

Single run simulation

The easiest way to perform a simulation is in *single run mode*. After starting, the simulation will run until the user stops the program. It is useful for several purposes:

- evaluation of results
- interactive investigation of parameters
- testing new configurations

- demonstrations

This mode simulates the robot within his environment, if defined, including the occurring physical forces.

Real robot control

For the purpose of controlling real modular robots this mode has been created. If a library for controlling a desired robot is available it can be wrapped to fit the system. In this case the output of the control kernel, shown in figure 4.4 is used to actuate the joints of real robots and virtual robots either. In addition information from the robot can be gathered and send to the control kernel. Figure 4.6 explains how sensors are virtually connected to the control kernel.

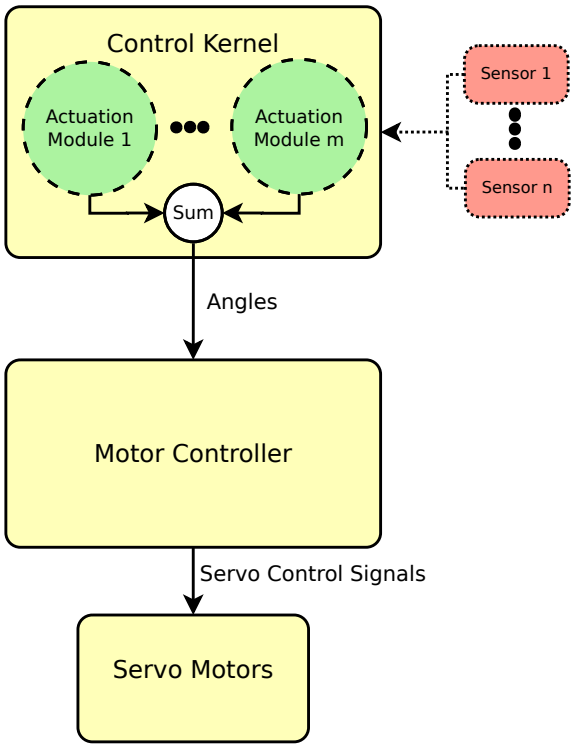


Figure 4.6: The control kernel with sensors in the big picture of the control system. Actuation modules have access to sensors if needed. In this way sensor information has influence on the angular output of the sum of control algorithms

If the GUI is used, a corresponding simulated robot is shown in the viewer. In this case no environment is shown. The virtual robot can be used to display the desired movement of the robot or to visualize the current state of the robot.

Optimization modes

As one of the most important features offered by the proposed system optimization modes are integrated. Part of this work is the implementation of unsupervised reinforcement learning (modified GD algorithm) and the integration of a GA library (Wall [52]). Various locomotion modes can be improved by finding good sets of parameters with the help of these optimization methods. Given that every algorithm generating movement patterns implies an optimization problem these methods can help to solve it. Implemented optimization methods can easily be configured with the help of the configuration interface as described on page 56. Both modes are using an objective function to evaluate single simulation runs. By repeating a limited number of simulation steps until a satisfying set of values is found the locomotional behaviour of the current robot is refined according to the expectations of the user.

During each simulation cycle the simulated environment can be supervised if the GUI is used. But it is also possible to run the optimization as background process to achieve a higher efficiency and to enable remote access from other machines. After each cycle results are written to a file.

Optimization using genetic algorithms Here heuristic optimization is applied. It tries to find good solutions for the formulated problem in a global way by using a fitness function. After increasing the amount of time the probability to get stuck in a local minimum is reduced. In GAs the problem consisting of a set of parameters will be formulated as genome. The initial genome will be randomly manipulated using two main operations.

- mutation
- crossover

The algorithm will run for many generations each consisting of a population with a bunch of individuals. Each individual represents one solution in terms of a genome. Termination occurs when convergence of resulting scores, calculated by the fitness function, is reached. This works according to the principle of *survival of the fittest*.

Optimization using modified great deluge algorithm After starting randomly with one possible solution for the problem this solution gets refined slowly by searching for a good neighbour. Every time a proper solution has been found the requirements for accepting a solution increase. To avoid getting caught in local minima the search space increases with the number of failed attempts to find a proper neighbour.

4.3.4 Online modulation of locomotion

The control kernel is designed to allow modulation of the locomotion procedures during run-time. Between two simulation steps, values of the parameters defining the characteristics of locomotion patterns can be changed. This is possible using the control GUI, by code or by using a remote control. As described by Noeske et al. [42] e.g Nintendo's WiiMote

can be used as human robot interface (HRI). It works well as intuitive and easy-to-use device for robot interaction. Using a remote controller device and the GUI high-level remotes can be prototyped very easily.

4.4 Configuration GUI

According to the different requirements of users at different levels of knowledge there are two graphical user interfaces for the task of configuration. One focuses on easy usability, while the other gives more configuration possibilities to experienced users. In [35] the user interface is explained more detailed.
The main function of the configuration interface is to assist users in generating configurations to run the system. In the beginning a robot can be constructed or selected. Then actuation modules can be implemented optionally that fulfil a specific task using the actuated joints of robots. After that, robots can be supported with a behaviour consisting of one or more actuation modules. Environments with the function of workspaces for the robots can be created in the next step. In the end global settings regarding the core system can be made as well as the simulation mode. In case of optimization modes there is an additional page for configuring how the optimization will be performed.

4.4.1 Beginner's configuration wizard

The wizard guides the user step-by-step through the process of configuration. Some pages are optional and can be ignored while others are obligatory and form a precondition for valid configuration files. Using the wizard interface the user is guided through the configuration task. File-names are generated automatically using timestamps and users do not need to take care about this issue. Each page has some fields that needs to be filled out to reach the next page. Many explanations and hints help users to finish the wizard.

4.4.2 Expert's configuration dialog

The main advantage of this configuration interface is that it can be used to configure single components of a simulation and to compose already configured parts to a complete configuration. In this way e.g. existing environments or robots can be reused and do not need to be defined again for another set up.

4.4.3 Robot construction

Figure 4.7 shows the page of the graphical user interface that can be used to construct a robot. The *upper left* group allows to give a name to the robot, which will be used for the robot configuration file name as well. The field to enter the name is mandatory to proceed with the next step of the configuration: the orientation of the joints and the number of joints[2]. With the help of the *lower left group* sensors can be configured and attached at

[2]Both together defines the *topology* of the robot.

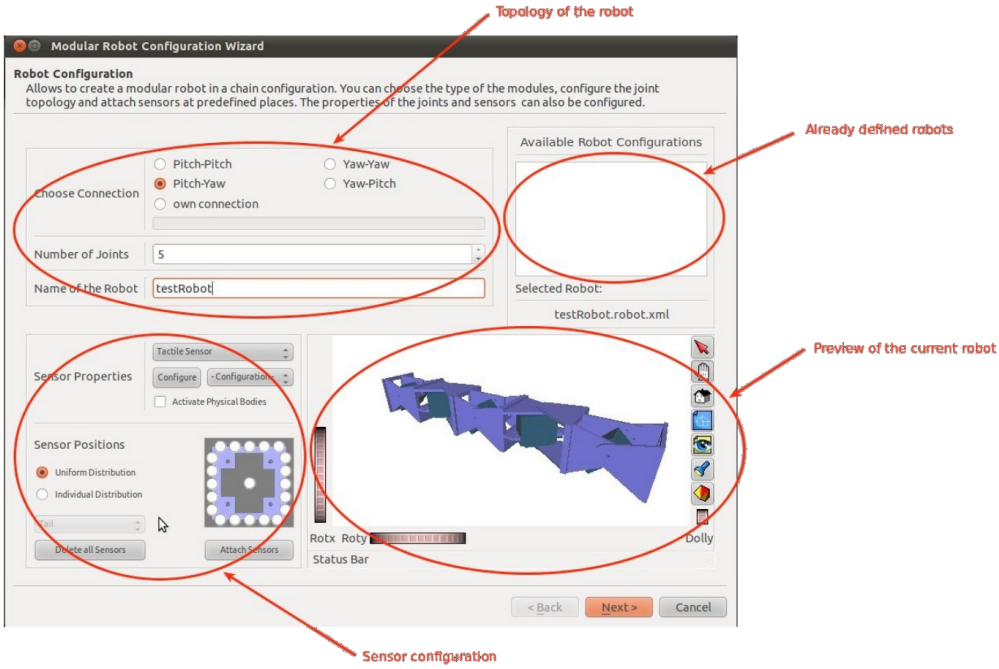


Figure 4.7: The wizard page for robot configuration is suggested to be used for easy creation of a robot. Joint topology can be adjusted and sensors can be placed at predefined places of the robot.

predefined places. Using the list widget in the *upper right* corner of this page it is also possible to select and reuse an already existing robot configuration. In the *lower right* part of the screen a preview of the current selection can be seen.

4.4.4 Creating new control algorithms

The first step to create a new control algorithm can be done within the wizard page shown in figure 4.8. The *upper part* allows to give a name to the algorithm and to add parameters, function and resulting values with desired names. The *lower part* shows a summary and allows to edit the settings after double-clicking on an arbitrary entry.
After everything is successfully declared the new control algorithm module has to be implemented using the wizard page shown in figure 4.9. In dependency to the previous wizard page a C++-style header was generated and a implementation file was prepared. The *left view* shows the header file that can be left as it is. The right view shows the implementation of the header file. The user only needs to implement at least the function that calculates the output angles for the joints of the robot. If the user is satisfied with his settings he can click on the compile button to add the new control algorithm module to the user-library of control algorithms. Both viewers support basic C++ syntax highlighting capabilities to make it easier to work with.

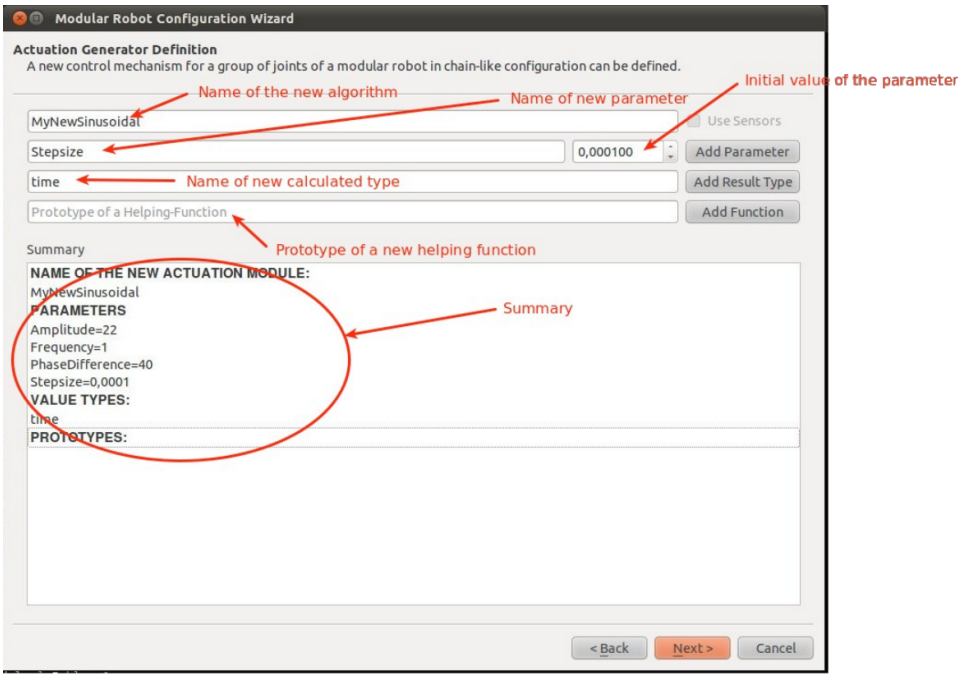


Figure 4.8: Actuation algorithm declaration allows to declare a new control algorithm, to add some parameters and to apply initial values to them. In addition helping functions can be added.

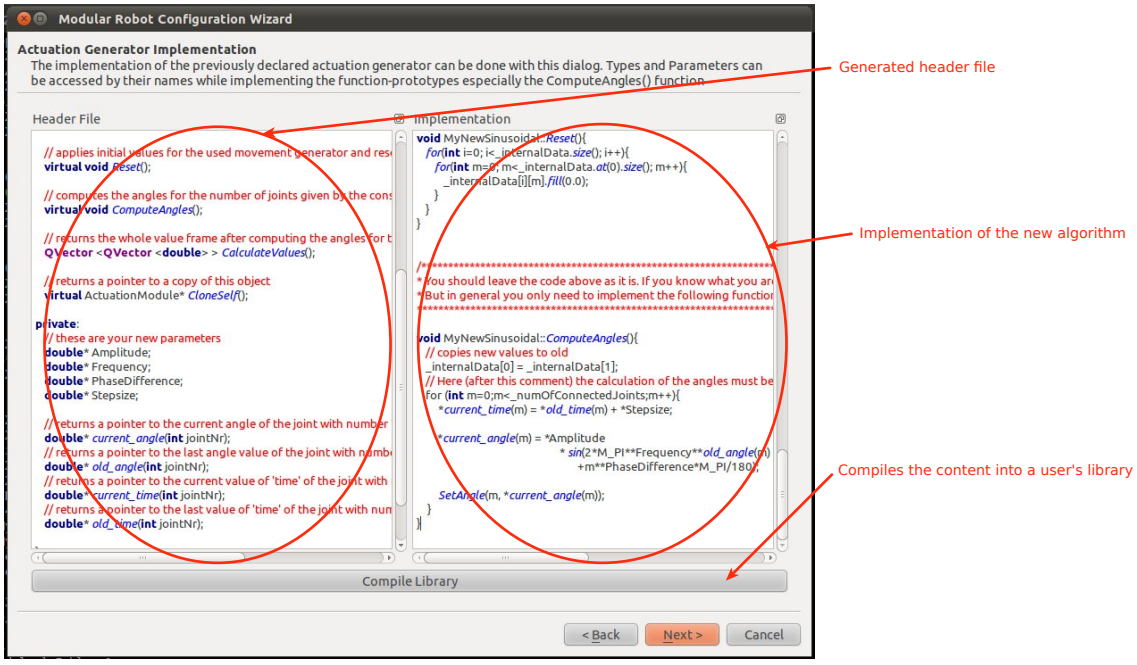


Figure 4.9: Actuation algorithm implementation is performed as second step. The main function that calculates the angular output has to be implemented as well as all helping functions. Then it gets compiled into the user library as a shared object file.

4.4.5 Defining robot's behaviour

Within this wizard page the user can assign a behaviour to the robot. There are two possibilities. *Simple actuation* can be used which means that all of the robot's joints will be actuated from the same algorithmic module to produce locomotion. The alternative is to use *grouped actuation* which allows more complicated combinations of different algorithms within one robot according to the principle shown in figure 4.4.

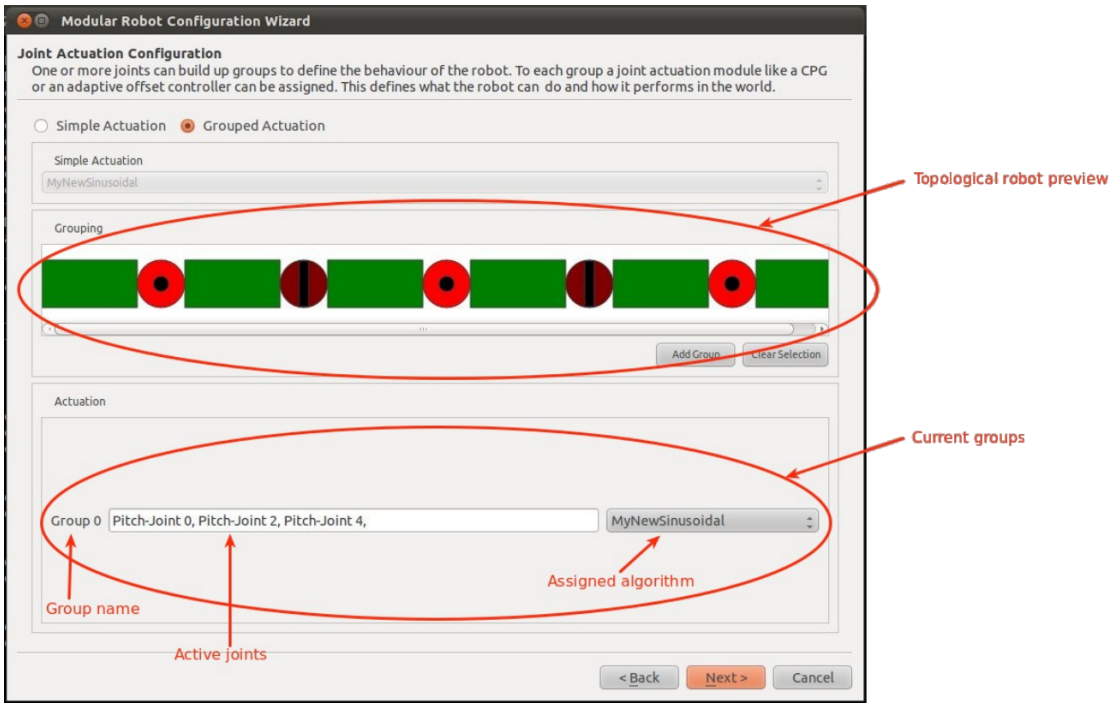


Figure 4.10: Assigning actuation modules to groups of joints creates the behaviour of a robot. Firstly, joints can be selected and grouped. Then one or more actuation algorithms can be assigned to a group.

This make this work two concepts are used:

- grouping of joints
- assignments of algorithms to groups

Several groups can be built by selecting joints in the topological robot preview. At the lower part of the current wizard page all groups that were built up are summarized. To each of these groups one of the available control algorithm modules can be assigned using the combo box. Each entry shows off a field with the active joints that are in this group.

4.4.6 Building environments

This wizard page enables the user to create an environment where the robot can operate. This can be done without any knowledge of OpenRAVE-XML.

At the left of figure 4.11 there a two graphic views. They are two-dimensional orthographic projections of the scene. The upper frame shows the scene in a side-view while in the lower frame the scene can be examined from the bird's eye view. The two views are synchronized and support zooming.

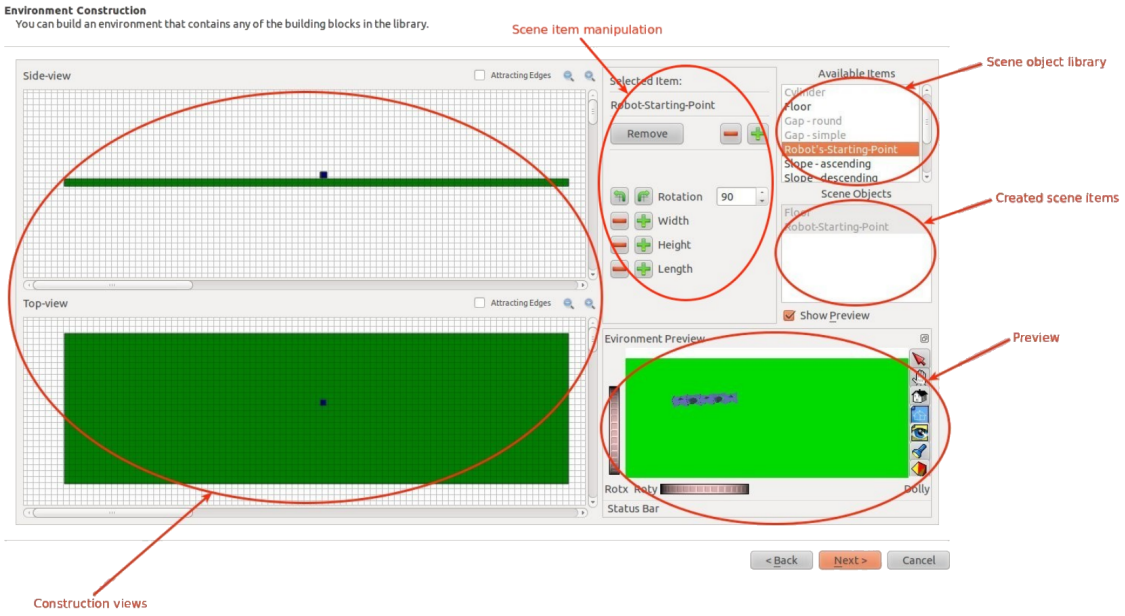


Figure 4.11: The environment editor can be used to create an environment by drag-and-drop of objects from the scene object library and to place the current robot in the environment. Objects can be manipulated and previewing is supported.

There is a library of objects that can be placed in the scene by drag-and-drop. The representative object for the robot's position is obligatory to place the robot at a reasonable pose. Below the list of available objects there is a list of already created scene items. Each object can be right-clicked to set its frictional coefficient[3].
The editor allows manipulation of scene items in a limited way. Selected objects[4] can be moved by mouse dragging. It is also possible to rotate scene items in the xy-plane using the manipulation frame. To rotate a selected object the 'rotate' buttons can be used to apply rotation with a fixed step size. Alternatively desired angles in degree can directly be entered into the combo box. In addition scene items except the robot itself can be scaled using the 'plus/minus' buttons.
To inspect a preview of the current scene in a 3d-viewer the *'Show Preview'* check box must be selected[5].

[3]The frictional coefficient of the coulomb friction model is valid in the range of [0 1]
[4]Selection is performed by double-click.
[5]Due to its implementation this feature is very expensive in computing time. It is recommended to use it only for short time to check the result of the creation and deactivate it again before additional changes will be applied to the scene.

4.4.7 Global simulation properties

The upper circle of figure 4.12 marks the simulation properties. Initial values regarding the simulator's core can be defined there. Most attention should be given to the simulation mode. It will be explained in the next section

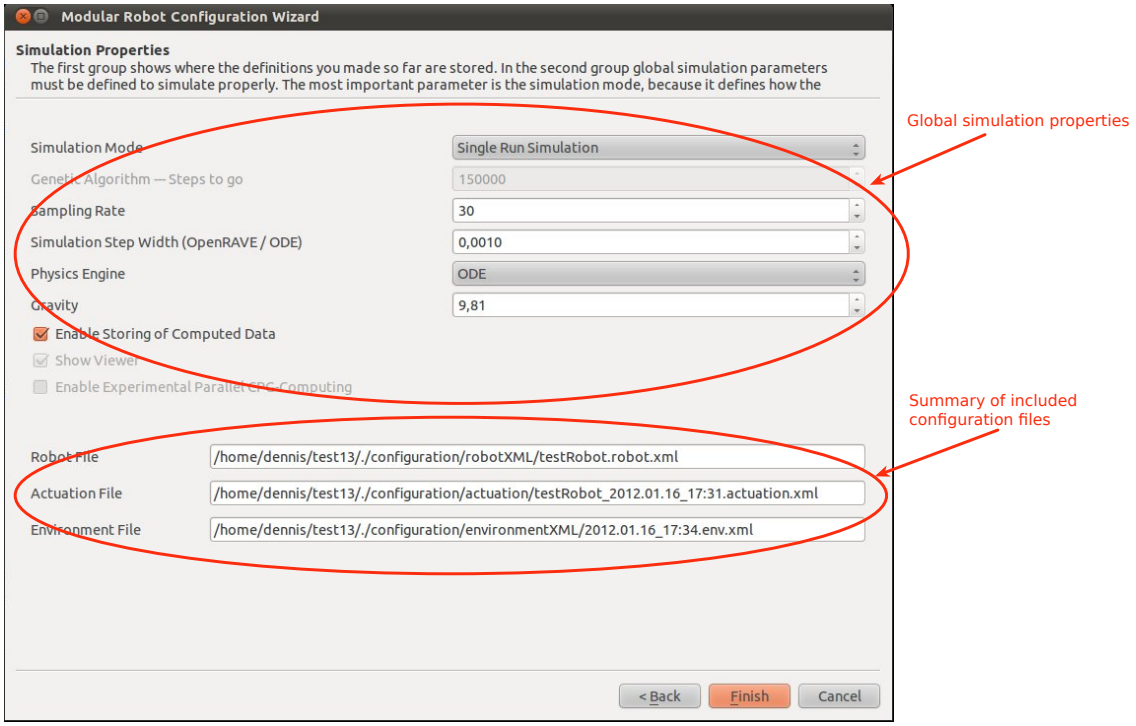


Figure 4.12: Global simulation properties needs to be adjusted using this wizard page. The simulation mode as well as some accuracy options and properties of the physics engine can be changed. At the bottom of the page there is a summary of other included configuration files.

4.4.8 Simulation mode configuration

Each supported simulation mode described on page 48 can be configured. Especially the learning methods offer many settings.

GAs – defining genomes

In figure 4.13 and 4.14 the necessary dialogues to configure a genetic algorithm with the help of the GUI is shown. The upper part of the wizard page, shown in 4.13, is needed to change the behaviour of the genetic algorithm that is going to be created.

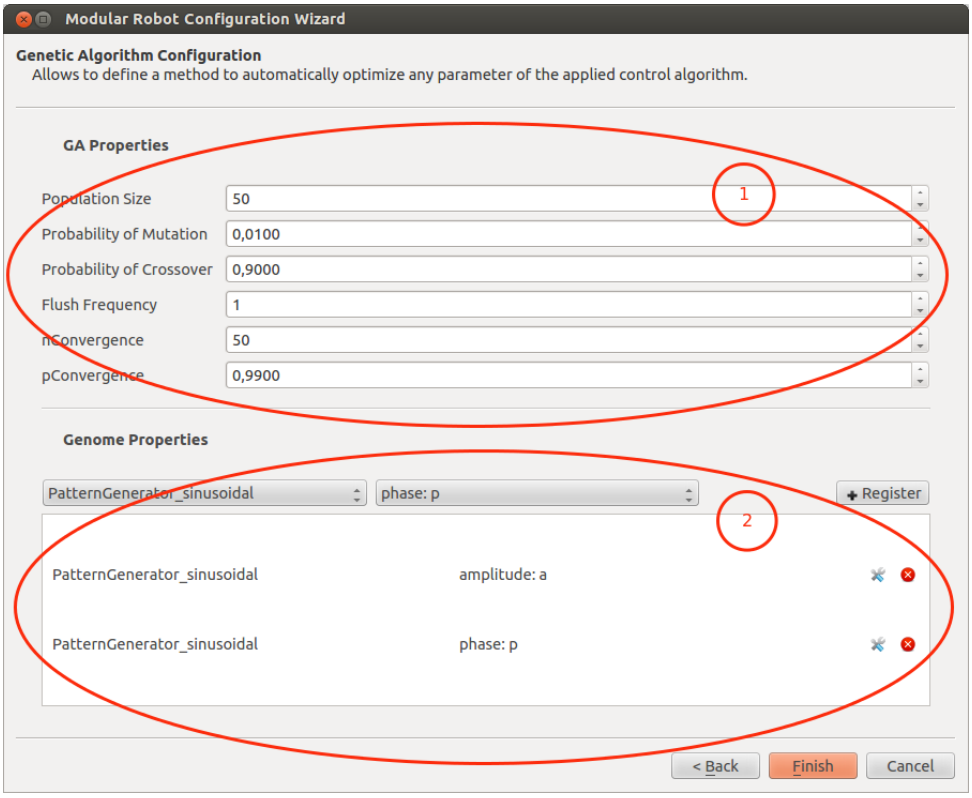


Figure 4.13: The first page of the GA configuration is needed to register parameters that should be part of the genome used in the genetic algorithm. All parameters from the current set of actuation algorithms assigned to the robot can be used.

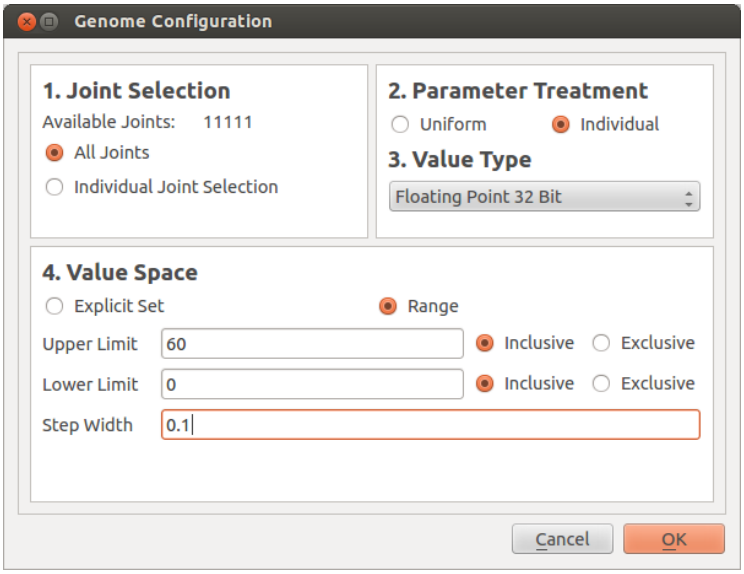


Figure 4.14: After registering a parameter its properties have to be configured. Especially the set of allowed values must be specified for efficient optimization.

To register arbitrary control parameters of one of the actuation algorithms from the control kernel, the second part of the parameter registration wizard page can be used. After one of the currently assigned actuation modules is selected, one or more parameters of the selected module can be registered to the configuration of the planned optimization run of the system. For further adjustments, the dialogue shown in figure 4.14 needs to be used. Each parameter added to the set that needs to be optimized can be changed in configuration. At first it has to be defined if its value should be improved for all or only selected joints and secondly if it should have the same value for each assigned joint. The alternative is to set them individually. For each parameter it is possible to define valid value spaces. Ranges can be defined or single values can be added to the set of valid values. How optimization of actuation algorithms works in detail is explained in section 3.4.2.

4.5 Control GUI

The simulation and control system has to be initialized with the information stored in a `*.simulation.xml` file. The simulation core as well as the GUI and the OpenRAVE components are initialized in this way. The control GUI allows the user to interact with the control/simulation core of the system. There is a tab widget containing several control and observation pages. In figure 4.15 the *tab bar* can be seen. It is useful to switch

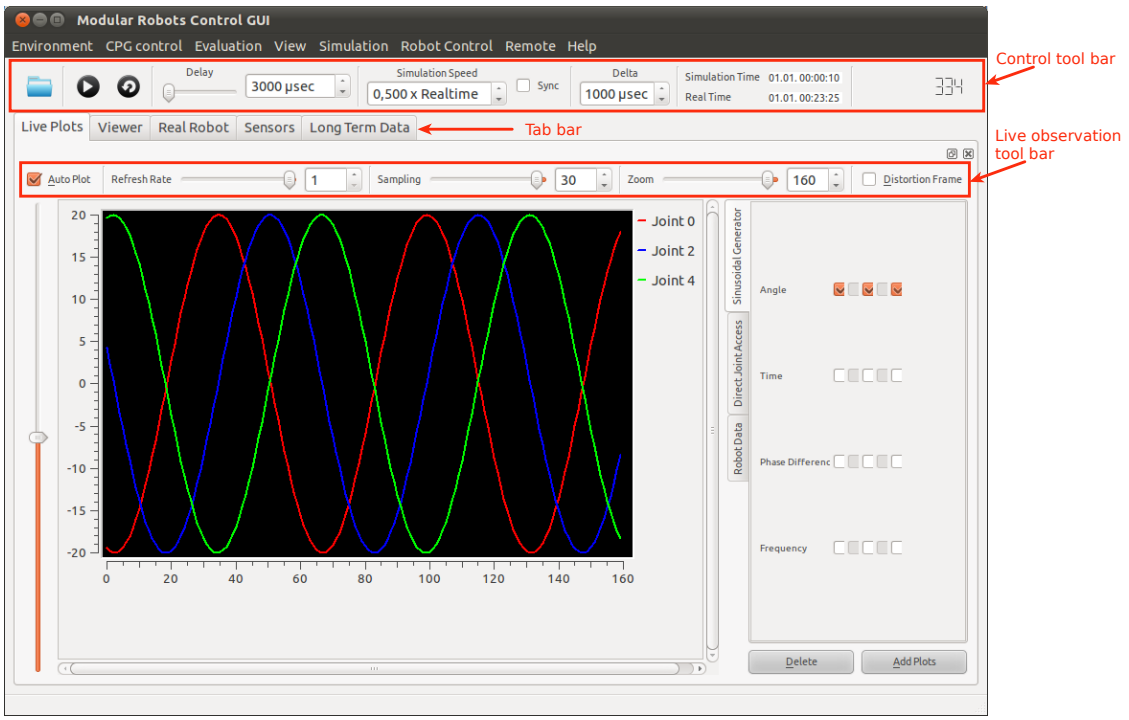


Figure 4.15: The main window of the control GUI enables users e.g. to supervise control algorithms with the help of live plots and the simulated environment by using the integrated coin3d viewer that is part of OpenRAVE.

between the existing tabs of the main window. The live data observation tab has its own toolbar and can be detached if needed. In addition there are several dialogue windows for configuration purposes, like the actuation module properties dialogue and a remote configuration dialogue.

4.5.1 Main window toolbar

The *main toolbar* shown in figure 4.16 offers some important basic information and direct access to some useful functions of the system. Its components are numbered as the following:

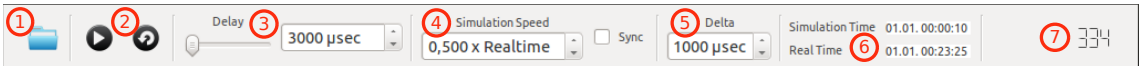


Figure 4.16: The main window toolbar provides users with knowledge about the elapsed simulation and realtime. In addition simulations can be started, paused and restarted. Systemload and accuracy can also be modified.

1. loads a configuration file
2. start-/pause- and restart-button
3. additional delay between two simulation steps
4. adjusts the simulation time in parts of real time or synchronizes the simulation time to real time
5. changes the step size δ of the physics engine (unit is $[\mu sec]$)
6. shows the simulation time and real time of the current simulation step
7. simple simulation step counter

The menu allows to hide this toolbar when necessary.

4.5.2 Live data observation

The active tab of the main window, shown in figure 4.15 allows to observe current data of the robot, its sensors or the control algorithm(s) used by the robot. As known by system monitors, this tab allows to display a kind of live plots on the screen. Using the widget shown in figure 4.17 the user can combine any desired data lines for observation into one group. With the help of the check-boxes of the different tabs within the plot selector, which represent joints or modules, single data lines can be selected. Clicking the 'Add Plots' button will create an element to display the selected graphs. For each control algorithm, sensor type and the robot itself there is a tab that contains check-boxes. A push on the 'Delete' button will remove the latest plot widget.
According to the desired re-plotting rate the graphs will be refreshed. This tab offers his own toolbar to apply some important settings. In this way when simulations are running in GUI mode it is possible to display current values of interest. For this purpose a sliding

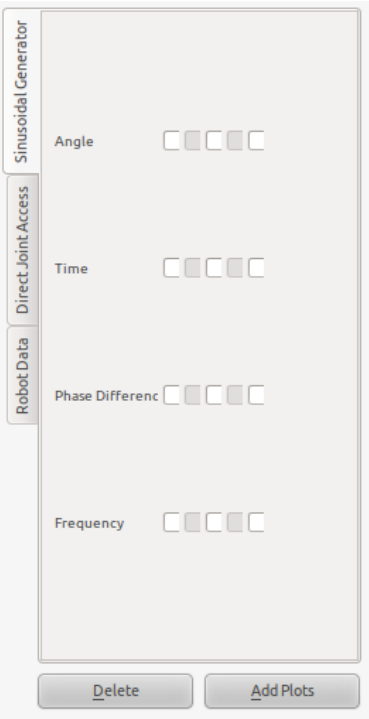


Figure 4.17: Live plots are selected by using this widget of the main window. For each available category graphs can be selected and combined to live-views as shown in figure 4.15.

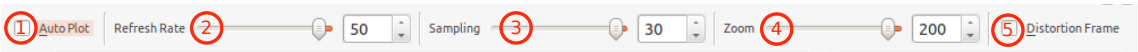


Figure 4.18: By using the live plot toolbar the characteristics of the visualization of data are changed. Performance is influenced by changing the refresh rate and sampling rate.

window of variable size is defined. To display live views the sliding window is moved to the end of each data line of interest every time the painting device gets refreshed. The live plot toolbar, shown in figure 4.18 has the following components:

1. If 'Auto Plot' is enabled, the plots will be repainted according to the refreshing rate.
2. Refreshing rate determines after how many new values the plots will be refreshed.
3. Sampling determines how many calculated values will be left out for plotting purpose.
4. Zoom adjusts how many values will be displayed at the same time.
5. The 'Distortion Frame' check-box shows or hides a widget to apply a constant distortion to any of the control algorithms.

4.5.3 Environment viewer

The environment viewer, shown in figure 4.19, is based on Coin3d. It is part of OpenRAVE and has been fully integrated into the GUI of the proposed system. It allows to manipulate the poses of all objects. In addition there are three wheels at the lower border of the viewer for changing the camera properties. Zooming and rotating the camera is possible. At the right edge of the viewer a small control panel is placed to change between camera perspective adjustments and object's pose manipulation mode. The other icon buttons are of lesser importance[6].

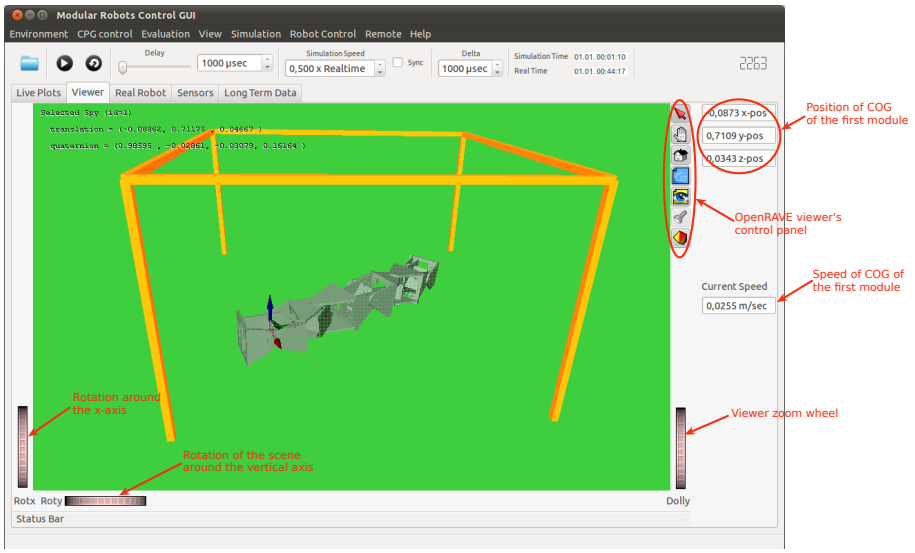


Figure 4.19: The environment viewer is based on the OpenRAVE viewer that has been integrated into the control GUI. OpenRAVE uses the Coin3d viewer for visualization of environments. To provide some additional useful information several GUI elements have been added that show the current position of the selected robot in the environment and its smoothed speed based on displacement.

4.5.4 Real robot control

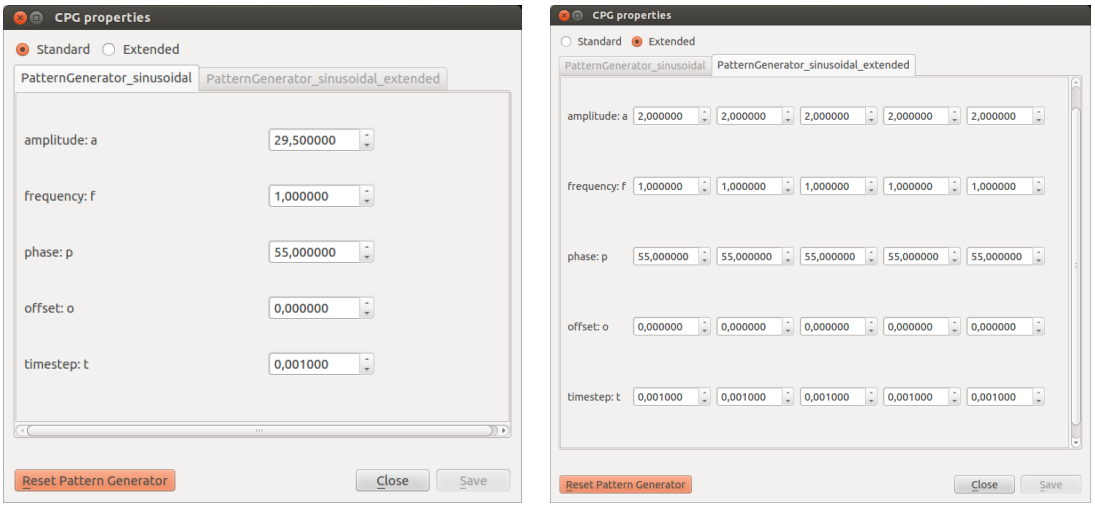
This tab needs to be reimplemented for each new real robot because of the different features of arbitrary modular robots.It allows to monitor the real sensors and other information read from the robot device. It is also possible to send commands to the robot using components of this wizard page.

4.5.5 Control algorithm properties

The dialogue shown in figure 4.20 allows to manipulate the values of all control parameters used by the robot's control algorithms on line, while the system is running. It allows to

[6]Detailed information can be taken from the documentation of Coin3d.

explore the effects of control parameter modulation in an interactive way. For each algorithm it offers an own tab containing all registered control parameters. Each algorithms' parameters can be adjusted in two different ways. They can be changed *uniformly* where every assigned joint will be calculated using the same parameter value or in an *individual* way using different values for each joint. The individual method allows to apply non uniform amplitudes. As explained by Chang and Chen [10] modular robots' actuation algorithms can benefit from variable amplitudes during locomotion. The usage of variable bending rates in animals' locomotion has been observed with real animals as well.
If the value of a certain parameter changes, e.g. by code, the current value in the dialogue will change as well. Core and GUI are synchronized by timers. When necessary the control algorithms can be reset to initial state by pressing the 'Reset' button.



(a) Normal property dialogue

(b) Extended property dialogue

Figure 4.20: To change parameter values of arbitrary control algorithms online these properties dialogues are used. The simple dialogue changes the values instantly for each regarding joint in a uniform way. In contrast the advanced dialogue allows to change the values individually for each joint. When optimizing actuation algorithms' parameters with enabled GUI these dialogues can also be used to watch the current values.

4.5.6 Data file processing

At all times data files can be stored to a drive containing all calculated values of the current simulation. To inspect and process these files, the data file inspection tab from figure 4.21 can be used. First of all a data file needs to be loaded. After that the tab will be initialized with meta information about the contained data. Then, a data line needs to be assigned to the x-axis using the combo box that is intended for this purpose. For the x-axis the value count or one of the offered time stamp systems should be used by default. Then, any of the desired data series for the y-axis can be selected. It is possible

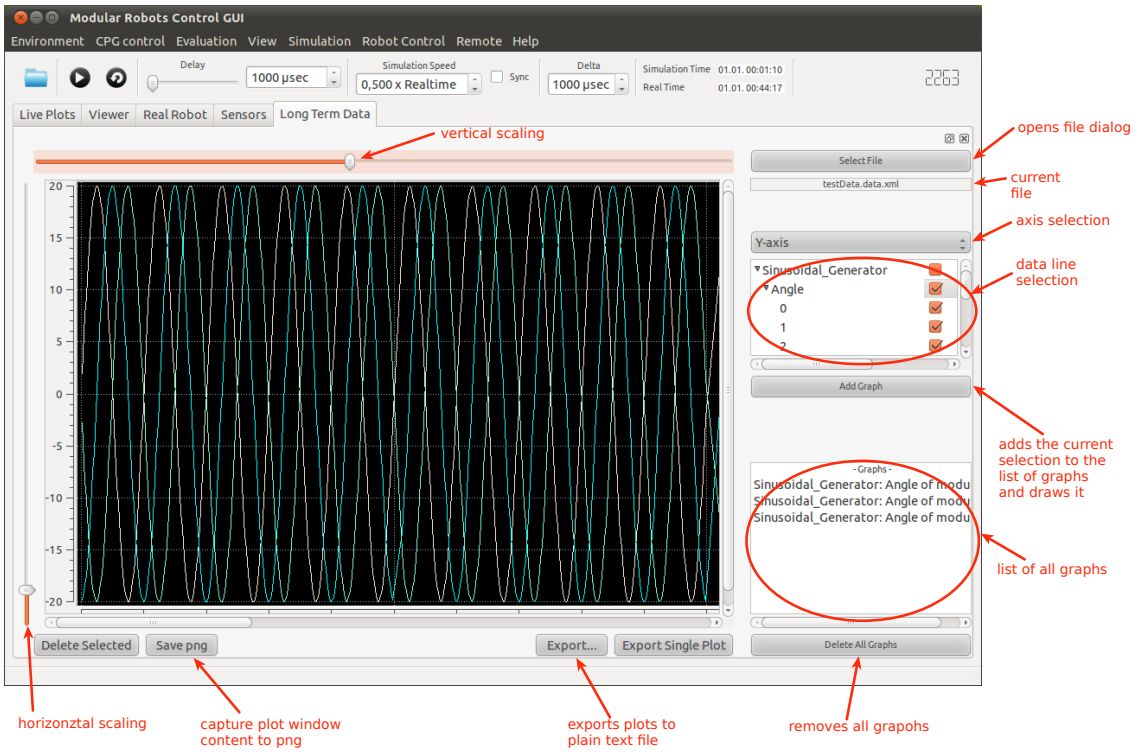


Figure 4.21: Data inspection tab helps to analyse data from files of passed simulation steps. After selecting a data file meta information is extracted about the content of this file and can further be selected for displaying or exporting. A nice feature is to capture directly screenshots of currently plotted data or to select desired plots for export to tab separated files.

to select only specific joints or to use them all by checking the parent node of the desired type. Clicking 'Add Graph' will cause the displaying widget to draw the composed graphs on the screen. An entry into the list of graphs will also be created for each curve. The visualization can be scaled in x- and y-direction with the help of two sliders on top and on the left side of the plot widget. It is also possible to zoom into a region of the widget, by clicking at the desired place and then, after releasing the mouse button again, moving it to another place and clicking again. With this kind of rubber band selection any rectangle can be used to zoom into. A right click causes to go back to the normal view. To capture the current content of the plot window the button named 'Save png' can be used. A useful feature is the capability to export the current selection of data into a plain text file that can be further processed with another tool like *Matlab* or *gnuplot*. This can be done simply by clicking the button labelled with 'Export'.

4.5.7 Definition of a high level remote control

With the GUI a connection to a remote control can be established. Currently, there is implementation of the remote interface (see section 5.4) for Nintendo's Wiimote. The

Wiimote is widely used in human robot and human computer interaction [13, 48, 39, 3, 45, 12]. In this case bluetooth is used for communication. After the connection has been successfully established, the remote control can be configured. The GUI allows to assign arbitrary control parameters to the elements of the Wiimote that can be used for the purpose of control. Reasonable values for the stepsize that determines the ratio of the change of parameters with respect to the values read from the remote hardware can be applied. In this way several parameters can be bound to each element of the Wiimote using different scales for modulation. Noeske et al. [42] describe this in more detail.

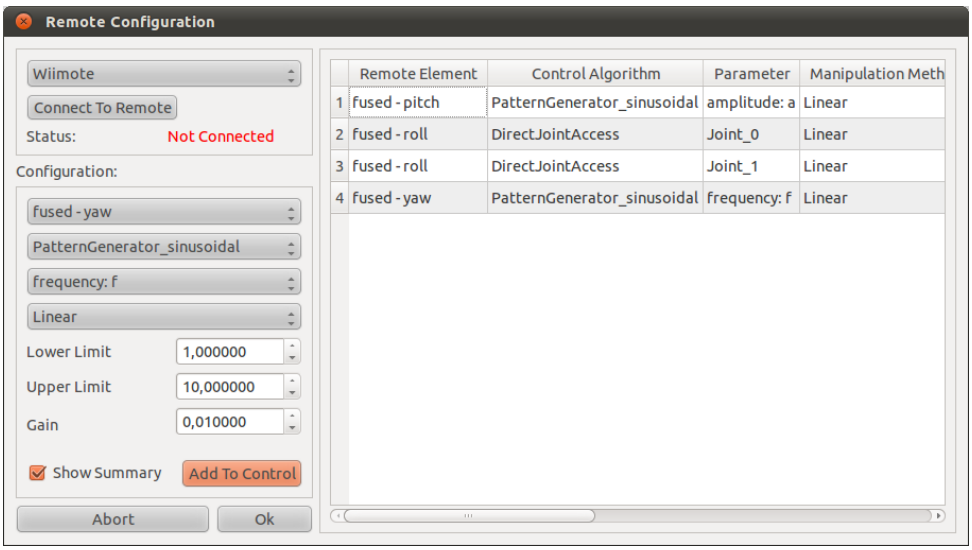


Figure 4.22: Using the remote configuration dialogue it is possible to select and configure a remote according to the selected set of actuation algorithms. Remotes are available in a library that can be extended. The concept is to bind manipulation of one or more parameter values to available hardware elements of a certain remote. This is useful for testing and evaluation of results and new ideas.

Figure 4.22 shows an example of a configuration. According to the capabilities of the current remote[7] and its implementation, hardware elements of the remote can be used to change parameters' values in order to manipulate the movement of the robot. In the example, the fusion of gyroscopes and accelerometers is selected. Pitching of the Wiimote will result in increasing or decreasing the amplitude of the applied travelling waves. Meanwhile rolling will change the offsets of the yawing joints in order to steer the robot to the left and to the right. Yawing can be used to manipulate the frequency of the travelling waves for accelerating or decelerating the robot. This scheme can be applied to arbitrary algorithms actuating joints.

[7] Sensors and buttons can be used.

4.6 Summary

This chapter gave extensive descriptions of the functions of the proposed system. Benefits from the GUIs make it easy-to-use and the concepts of the core system allow efficient re-use of parts of previous work. It was explained that it is general enough to fit the needs of future demands. But as described the system is also suitable for people who are not very familiar with modular robotic locomotion. As explained no further tools are needed because everything necessary is already integrated. The next chapter gives deep insights into the implementation of the system and focuses on the library-based approach that allows to extend the system at runtime without recompiling the whole software. It is explained how these mechanisms are implemented.

Implementational Details

5

The proposed simulation system has been implemented using C++, Qt and some libraries enumerated in section 5.1. This chapter presents details of the implementation of important components.
Descriptions about integrated software and libraries is given at first. Second, the most important concepts allowing a high degree of flexibility and extendibility are explained. The idea and abstract interface of actuation modules (see 5.2) is explained. It is followed by the introduction of a robot hardware control wrapper interface (see 5.3) and a remote control base interface (see 5.4). Finally some file formats (see 5.5) used for configurations and results are introduced.

5.1 Integrated software, libraries and plug-ins

To build a powerful system many components have been integrated into the modular robotic simulation system. By doing this, the focus of this work was placed on modular robotic locomotion design, evaluation and optimization.

OpenRAVE

The Open Robotics Automation Virtual Environment (Diankov [17]) is a robotic simulation and control environment with focus on kinematic and geometric information. The structure of this open-source system is very flexible and allows users to write plug-ins and to use arbitrary robots. It features 3D-viewer, physics engine and a small runtime-core that can easily be integrated into other systems. Robots and their environments can be defined using XML files. Using one of many base interfaces of OpenRAVE, developers extend the system as needed with new sensors, robot parts, XML readers, inverse kinematics modules, planners and many more.

ODE

The Open Dynamics Engine (Smith [49]) is used by OpenRAVE to simulate rigid body dynamics when robots are interacting with environmental objects. It is widely used in robotic simulation software[1] and many video games. It allows collision detection with

[1] E.g. Cyberbotics' *Webots* [15] uses ODE in simulation environments.

friction. Based on friction, shifting of modular robots is produced. Therefore, ODE is the most important part of the software because it holds basic principles of creeping modular robots' locomotion.
In addition ODE defines several joint types and allows to map real joints to simulated ones very closely.

OpenMR

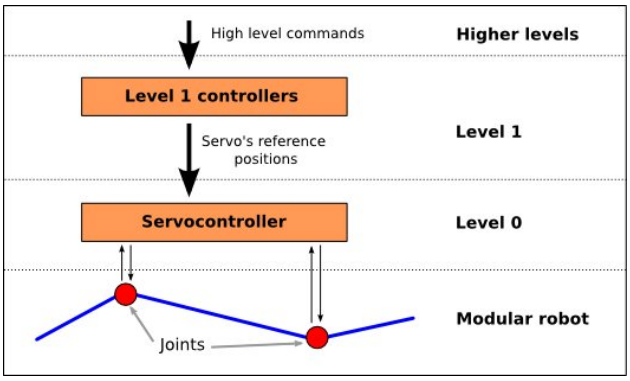


Figure 5.1: Concept of simulated servo controllers of the OpenRAVE-plugin OpenMR. As level 1 controllers several algorithms as actuation modules described on page 70 have been implemented. (González-Gómez [22])

With the OpenRAVE plug-in *OpenMR* González-Gómez [22] implemented a servo controller, shown in figure 5.1, that internally has a proportional controller. Angles in degree can be sent as a command to instances of this controller, that will change the state of the assigned joint according to the dynamical properties of the controller. Within a robot definition file that OpenRAVE can read, this controller can be assigned to the joints of the robot. Together with its joints this controller imitates servo motors. In addition González-Gómez [22] included a 3d-model of modular robot prototype, used in the research at the University of Madrid and the University of Hamburg. Both have been used in the experiments the 3d-model and the controller. Using the OpenRAVE XML format or collada XML format arbitrary robots can be used in the same way.

Tactile sensor plug-in

The used sensors do all implement the Sensor-Base-Interface of OpenRAVE and can be regarded as OpenRAVE plug-ins. This allows to use any sensor provided by OpenRAVE or developed by any institution for the use with OpenRAVE. To access the sensor data, special sensor data handlers were implemented for the simulator which can access the data and are able to reprocess the gathered data. That means the number of available sensor types depends on the existence of a matching data handler. These handlers can provide algorithms that actuate the joints with sensor information. They are also necessary to get the data for storing and plotting purpose. Using an abstract base class for these data

handlers, new handlers can be implemented easily.
The implementation of tactile sensors, used in this work, produces tactile data, according to OpenRAVE's sensor architecture. Collisions within the physics engine are used to determine contact of tactile sensors with any obstacle. Tactile-data objects can contain information about direction and amount of occurring forces. In case of the proposed work the implementation of tactile sensors uses debounced collision information and neglects amounts of occurring forces. Each sensor has only two different states. Like a simple button it can be pressed or not.

Qt

Qt is a cross-platform application and GUI framework for C++-developers. Graphical user interface elements allow to create powerful, flexible and nice looking applications. Each GUI consists of combinations of widgets that communicate with the signal-and-slot-principle. These signals and slots can be used for asynchronous communication with other non-GUI components. Using this principle and many other non-GUI classes provided by Qt C++ is extended widely. The proposed simulation system uses many features, data types and classes offered by Qt.

Qwt

With the help of *Qt Widgets for Technical Applications* series of data can be plotted on the screen nicely. In the GUI this library is used to display static plots of data as well as live-data from the currently running simulation or control of robots. As known by window-based CPU-load tools data can be plotted directly after it has been created in an efficient way.

Boost

Boost is a collection of C++ libraries containing many implementations of container-classes, data types and concepts. These libraries can be characterised as very stable and efficient. Several implementations of e.g. circular buffer and thread are used in the simulation system.

GAlib

The C++ Library of Genetic Algorithm Components (Wall [52]) contains a set of C++ genetic algorithm objects that can be used to produce solutions to custom optimization problems. The library allows to change all properties that are of importance to create efficient genetic algorithms in order to produce good results in an efficient and reasonable way. In the proposed system it is used to optimize parameters of actuation modules based on results from simulation.

5.2 Actuation module library

All available actuation algorithms are compiled into shared libraries to make the system extendible and modular. There is a standard library with some basic algorithms that can be used to define the behaviour of robots and one user library that can be further extended. Because the system was created using Linux these are share object files[2]. The idea is to enable users to create their own algorithms for actuating joints of several modular robot configurations. Without recompiling the whole system it is possible to add new actuation modules. This makes the system suitable for projects and workshops with hard time restrictions.
Figure 5.2 shows actuation modules from users point of view. It is a system receiving commands to generate desired angles for the joints of the robot. These angles will be converted by controllers into motor control signals. How angles are computed is chosen by users who implement the module. To fulfil this task users can define parameters changing the behaviour of the module. Parameters allow subsequent modulation of the output. The function computing the angular output can use helping functions, defined by the user himself. In this way the function calculating angles can be structured easily. The third component is a storage for intermediate results. Every calculated value of interest can be stored in a matrix. Every value gets stored in one XML data file if needed.

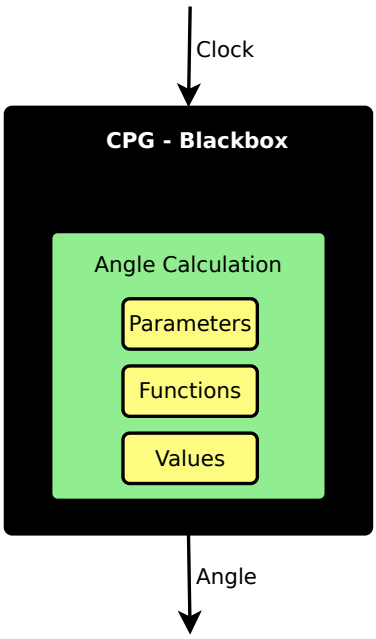


Figure 5.2: Each control algorithm that calculates angular positions of joints follows the concept of the actuation module blackbox. After receiving the command to calculate the next step angular output is generated. The function that produces the output uses previously in the implementation defined parameters, function and intermediate data.

[2]The filename extension is `so`. Using windows these libraries are called *dynamic link library* (`*.dll`)

5.2.1 Abstract base interfaces

To implement new algorithmic modules that generate input signals for the servo controller there are three different interfaces as shown in 5.3. Each interface provides modules implementing it with different properties. `ActuationModule` is the most basic interface. The code is part of the appendix A.1. It allows to implement modules for manipulation of joints in general. It offers on-line modulation of parameters, values can be logged and displayed in plotting windows at runtime. Classes implementing optimization methods are able to change the values of current parameters. For future purposes of parallel evaluation of different set-ups they offer a cloning function. With the implementation of `CloneSelf`, a cloning function, this interface follows the design of OpenRAVE[3].

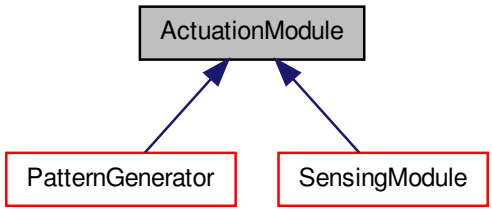


Figure 5.3: The abstract base class of actuation modules is inherited by two different interfaces. In this way users have three different possibilities to extend the control algorithm library. As pattern generators algorithms can be regarded that calculate periodic output. Sensing modules allow to access sensor data and to use them for calculation of the output. Implementing actuation modules give most freedom to users' decisions but also excludes some useful functions like calculation of frequency of periodic signals.

As `PatternGenerator` can be regarded everything that produces periodic signals comparable to a sinusoidal curve. Using this interface provides the user automatically with information like the calculated phase difference between neighbouring modules as well as the frequency. At the beginning or on reset the module's storage gets initialized with Gaussian white noise, which is often an important starting condition for CPGs. Another feature provided by this interface is the possibility to add distortion to one or more entries in the storage. Robustness and behaviour in case of drop out can be investigated. For deeper insights the reader is suggested to have a look at the appendix A.2.
The `SensingModule` interface allows use of sensor data from sensors that are attached to the virtual or real robots' body. Each algorithm, inheriting from this base interface, is provided with an instance of a sensor data handling object for each attached sensor type to access the latest and preprocessed sensor data. The code of the interface is in the appendix A.3.

[3]Every component of an OpenRAVE environment can be cloned either.

5.2.2 Creating new algorithms

Arbitrary control algorithms can be integrated into the system if they were implemented using one of the base interfaces. This can be done by using any text-based editor or with the help of the control GUI of the proposed system. When using any editor the user must assure to compile it correctly as a shared library that can be integrated into the system. The control GUI offers mechanisms to automatically generate the largest part of the code and handles the compilation into the user-library of actuation algorithms. In principle every control algorithm follows the structure shown in figure 5.2. The task of a control module is to produce angular positions of at least one joint. Every time it gets the command to calculate the output, it uses a top-level function to produce it. Then the output can be used by controllers to actuate joints. Within each of these control algorithms this function must be implemented. This function can use other assisting functions, their intermediate results and some parameters to calculate a solution as an output of the module, described in section 5.2.

5.2.3 Implementation using software design patterns

Dynamic class loading of classes with *self-registering types* was the key to provide computational efficiency and flexibility. Instances of dynamically loaded classes can be created using a *factory*. Users are allowed to create their own algorithmic modules having access to the joints of the robot at runtime and without recompiling the whole system.

In the standard UNIX/POSIX architecture there is C-header *dlfcn.h* that contains C-functions to use shared libraries. We use the functions from this header file to load shared object files (*.so) containing modules that can actuate the joints of the robot. When opening a shared library, symbols can be exported and used to extend our program. The classes defined in a shared library must export symbols that can be used to create instances themselves. For this purpose each class gets a static function *maker* that creates a new instance of the class and returns a pointer to the type of its abstract base class. Adding this function and its name as a key to a globally defined *map*, builds the factory. The map will be exported when the library gets opened by the *dlopen*-function of the *dlfcn.h*. In [19] the principle of a factory is described. After adding a *proxy-class* that registers the *maker*-function of the current class with the name of the class in the factory, instances of dynamically loaded classes can be created.
After opening a shared object file of this kind, each class defined in the library will have an entry in the factory containing the name of the type and a pointer to the maker function. That's all we need to work with these classes. How to create an object with the help of the factory is shown in the following example:

```
1   ActuationModule* newCPG = cpgFactory["MTRAN"](5);
```

This creates an object that represents a CPG of a type called *MTRAN* for five connected joints. *'MTRAN'* is the class name and *'5'* the parameter of the constructor.

The system is equipped with a *standard library* that contains some algorithms for joint actuation of the robot. For example these can be locomotion algorithms to move the

robot. Users can extend the number of available algorithms by creating a *user library*. The configuration interface allows to extend the user library of joint control algorithms in a very easy way. A new module has to be defined according to figure 5.2. After that the function(s) must be implemented and compiled into the extended user library. If the software is properly installed the user does not need any knowledge about compiling libraries. A closed solution is offered without the need of any additional text-editor. There is a simple editor integrated in the configuration interface. It features very simple syntax highlighting for C++ and this should be enough to implement one or more functions. After a new algorithm class is added it remains in the user library for future usage.

5.3 Real robot control interface

To control real modular robots' hardware a robot control interface has been created. It enables users to implement control libraries of available robots that needs to be controlled by the proposed system. The connection to the proposed system is created by using a wrapper for this interface. The wrapper extends the base control library with Qt-concepts which allows to use signals and slots. In this way the wrapper can be connected and disconnected to system at runtime. Listing A.4 shows the idea how of realizing the base control interface in software. All listed abstract functions need to be implemented to create control objects for new robots.
The benefits of controlling real robots are that users can directly test their control algorithms on hardware after the phase of testing in the simulator is accomplished. In this way the same implementation of robots' behaviour can be used for both controlling real and simulated robots. For this purpose on the one hand the interface needed to be general enough to support a wide range of arbitrary robotic prototypes and on the other hand in a certain way restrictive to maintain efficient control of hardware. To be as general as possible commands can be send to robots by using a single function `sendCommand`. As parameter it expects a string that contains the real command sent to the robot. Command strings need to be parsed by the user's implementation of robot control. Functions with side effects are used to get information about robots' state. In addition, there are some common but useful functions like `connect, disconnect, reset` and functions returning meta information about currently controlled robots.

5.4 Remote control interface

For the purpose of extending interactive capabilities of the simulation and control system, a remote control interface has been designed that allows to implement the usage of arbitrary remote control hardware like gaming device controls or other mobile devices. As an example, an implementation of this interface for Nintendo's Wiimote has been created. The motivation is to allow evaluation of robots' actuation mode using hardware remote controls. To implement the interface, supported control units must be defined. These can be assigned to parameters of current control methods actuating robot's joints. In case of joysticks this means pushing to the left could lower the value of a certain parameter and

pushing to the right would result in increasing the value. There is a graphical configuration interface, explained on page 63 for detailed configuration of the remote. Users can create a kind of *high-level control* for robots in this way.
The base interface is shown in section A.5 on page VI of the appendix. To create implementations of remote devices this interface must be implemented. There are functions called `connect` and `disconnect` that manage availability of remotes. Especially for wireless devices these functions are of importance. A `sendCommand` function has been implemented to provide the interface with high flexibility for future implementations of remotes. It allows access to any special function of new devices by parsing the command string. To acquire snapshots of the current state of the remote and its sensors and control parts `readAllData` can be used. To get the properties of the current remote there are two functions `getAvailableControlDevices` and `getAvailableCommands`. These are used by the remote control configuration dialogue of the GUI to allow the user to configure the robot remote control according to his needs.

5.5 File formats

The configuration of a simulation or control loop is stored in XML formatted files. They show off a modular and hierarchical structure. It is possible to change the configuration within these files by hand if the keywords and structure used in the proposed system are known by the user.

5.5.1 Configuration files

There is a top-level simulation file (`*.simulation.xml`) that includes other configured components. These are:

- global properties
- robot configuration (topology and sensors)
- actuation of the robot
- environment
- (optional: genetic algorithm configuration)

In listing B.1 an example of a top-level configuration file is given.

Global properties

The settings of the global properties of a simulation can be seen in the example of listing B.1. Within `simulationProperties` all global options can be found. Currently the following options are available.

- showViewer
- parallelComputing

- storingData
- mode
- accuracy
- stepWidth

With `showViewer` it can be determined if the coin3d viewer will be shown. Valid values are 1 and 0, these values correspond to boolean values. `ParallelComputing` can enable or disable a special feature for parallel computing[4] of the angular output of the actuation modules using OpenMP[5]. To enable or disable the data storing mechanisms of the system `storingData` can be used. This also affects the creation of a temporary datafile. There are several operating modes provided by the system. They can be distinguished by their name within the `mode` tag. The value in `accuracy` changes the sampling rate of the system regarding the visualization with live-plots and the updating rate of sensor data. It does not effect the 3d visualization of the environment. The `stepWidth` changes directly the step-width of the physics engine δ with unit [μs] and other related things. Thus, we perform continuous simulation it corresponds to a kind of sampling of the world's laws.

Robot configuration

To describe the topology of a robot OpenRAVE-XML[6] is used. Robots can be build from arbitrary modules simply by connecting links and joints. The properties of these can also be changed according to the needs. In addition it is also possible to attach several sensors to the robot. In listing B.2 the description of the robot shown in figure 5.4 is noted down.

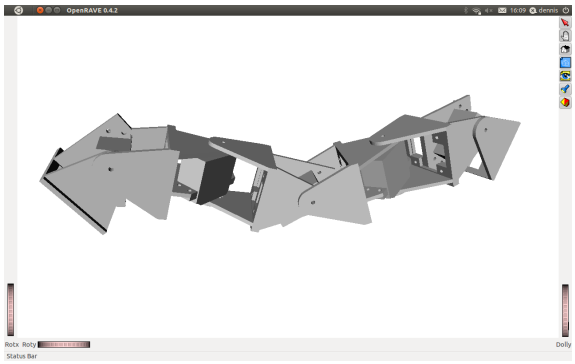


Figure 5.4: OpenRAVE-XML description files are used to compose robots (see appendix B.2) out of rigid body parts and joints. Restrictions of maximum torque and maximum speed can be defined within these files.

[4] This feature is currently not in use. But it works well when implemented inside an actuation module. There will be a benefit when using locomotion algorithms with a very high computational complexity running on multi-core machines. Then every involved module or joint can get his own thread for calculation of the output to speed-up the calculation time.

[5] http://openmp.org/wp/

[6] http://openrave.programmingvision.com/wiki/index.php/Format:XML

Sensor configuration

The utilized sensors are based on the abstract OpenRAVE sensor base interfaces. Thus they have their own configuration files, depending on their implementation. Sensors can be added to the system as a plug-in and dynamically loaded if they are compiled as a shared object file. In listing B.3 and B.4 there are examples of how these sensor configuration files can look like. In general only properties of a certain sensor can be adjusted if the corresponding functions are implemented that allow to change them.

Actuation configuration

Listing B.5 shows an example how a configuration file that determines the behaviour of the robot could look like. The attribute `robotName` of the tag `validRobot` stores the name of the robot file for which the actuation configuration was created. Each of the `actuationModule` tags stores the information needed to assign an actuation module to a group of joints. The attribute `name` holds the name of the current actuation module. The name can be used later to create an object of this kind that calculates the angular output for a group of joints. The tag `jointConfiguration` stores the information with which the joints should be actuated by this module. Under the attribute `joints` a string of zeros and ones with as many elements as the number of joints of the current robot is stored. '1' means the joint is actuated by the current module and '0' means that this joint is disabled. The position in the string corresponds to the indices of the robots joints. In this way groups of joints can be built and several locomotion methods can be assigned to these groups. Assignment of groups is allowed to be redundant and overlapping.

Environment configuration

As well as the robots' configuration files the environments' configuration files follow the format OpenRAVE uses. So the OpenRAVE-XML documentation can be used to understand how to build environments by hand. There is only one exception. Robot configuration files are not included in the environment files. Indeed the pose of the robot is stored as a comment in the last line of the file. An example is shown in listing B.6. These configuration files allow to adjust the physics engine's properties as well as placing several objects in the world. There are geometric primitives available but also importing of triangle-meshes is possible. Each object can be placed in the desired pose as *static* or *dynamic* object. The properties of each object can be adjusted. Some of these properties are:

- frictional coefficient
- mass
- mass distribution
- color

For a more detailed explanation of the available options please refer to be OpenRAVE documentation.

Genetic algorithm configuration

To configure integrated genetic algorithm mechanisms configuration files similar to listing B.7 on page IX are expected to be used. In general two categories have to be adjusted, the properties of the genetic algorithm and its genome.
In *properties* some parameters are defined to determine algorithm's behaviour. These have to be chosen wisely because they mainly determine the runtime and efficiency of the GA. In addition parameters are registered by their name within the properties-tag.
The second part of the configuration file consists of definitions of genome fragments for each parameter that has to be part of the genome. The whole genome is composed of fragments, one for each parameter that has to be part of it. Each fragment needs definitions of valid values. This can be for instance a sampled range containing of minimal value, maximum value and a step-size. The proposed file format allows even individual parameter handling. This means that for every joint that is concerned a different configuration of valid values can be chosen. If this feature is not needed it is also possible to use the same value for each involved joint. By the fact that the control kernel shown on page 47 can consist of several actuation modules in the genome fragments it has to be noted down which joints of the robot are effected. For this purpose *genomeConfig* tags own the attribute *activeJoints*. It is a binary string of the size of the current robot, where the position in the string equals the enumeration of joints of the robot. '1' means that this joint is effected by this parameter and '0' means not affected.

5.5.2 Data files

All information calculated in one simulation can be stored to a single XML data file. In Listing B.8 a shortened example of a data file is shown. It does not contain information from sensors, because no touch sensor was attached to the robot model.
The first subnode in the file format contains metadata about information stored in the current file. This `header`-node has one entry for each kind of actuation module, sensor type and one for offered information about the robots position and speed. In this way it is possible to reconstruct very fast how many different types of values are stored and how many individual values are generated in each simulation step. In addition each entry contains a list with names of all data types stored.
All following nodes contain data values. Since writing of data files occurs after a fixed number of samples there is a node `round` that contains all data since last storing event. Data is organized in `frame` nodes. For each computed step of a single actuation module all data is stored frame by frame. These frames are easy to distinguish by using their enumeration number, real-time stamp or simulation-time stamp. Within one frame of a single data producing module all of its available data types are stored in subnodes.
In the control GUI data analysis is included that allows to read metadata stored in the header node in order to allow users to select any plot from available data for displaying or exporting purpose. Displaying allows to inspect several data lines at the same time. Once selected these files can be exported.

5.6 Summary

After a summary of integrated software and libraries, programming techniques were explained that allow to extend the proposed system further. In this way more algorithms for modular robot locomotion can be implemented and loaded at runtime of the program. Then, users have access to these with the help of the configuration interface. In this way the proposed system is extendible regarding algorithms producing locomotion patterns. For evaluation purposes arbitrary real robots can be controlled using the same mechanism. Since users can implement new wrappers to control desired real robots, the system is also extendible for evaluation and verification purposes on real hardware. This and the recording of data from simulations make the system useful as framework for research. The next chapter presents two different experiments with simulated robots in order to show the usefulness of the system for application. In these experiments two different optimization methods (see section 3.4.2) are used.

Experimental Results

6

Several set-ups have been created to compare different methods for achieving good solutions for formulated problems of locomotion optimization. Since simulation set-ups are highly polymorphic, only very few combinations can be described in this work. To make different methods of optimization comparable, the same simulation set-ups are chosen to be optimized. Even the sets of valid values and their limitations are exactly the same. For comparison the same fitness function is used and to make assumptions about efficiency the same number of repeated evaluations is used. Despite many different aspects of optimality exist, only improvements in speed are analysed. Depending on application arbitrary fitness functions can be implemented to improve locomotion at different aspects.
Section 6.1 presents an experiment where the shape of travelling waves is optimized in order to provide fastest locomotion possible by varying amplitudes individually. Section 6.2 presents an experiment where several parameters of an adaptive locomotion algorithm are optimized in order to pass an irregular shaped obstacle with the help of sensor feedback. These parameters determine the resulting locomotion patterns in dependency to sensor feedback.

6.1 Optimization of travelling wave shape

Modular robots, that use travelling waves to generate propulsion, can vary the level of efficiency of their locomotion by altering high level parameters like amplitude, frequency, phase-difference and offset. In the following experiment sinusoidal generators are used to move a simulated robot in a very simple environment, consisting of only a flat ground. Phase difference is set to 55 degrees and maximum amplitude is 40 degrees. Figure 6.1 shows a whole cycle of the specific travelling wave in a schematic robot. Since there are five modules, the number of travelling waves, generated by robot's body is always around 0.75[1]. As described in section 3.3.1 these parameters determine dynamics of locomotion. In parameter optimization using the proposed system, two different possibilities are offered to handle these values. One is the *uniform* approach where every module's wave generator will have the same value each parameter applied and the other one is that every module can have an *individual* value. Chang and Chen [10] describe different models of variable amplitude to improve locomotion speed of chain-like modular robots. Inspired by their work this subsection describes experiments where parameters determining the shape and dynamics of travelling waves are treated independently. The following optimization

[1] $\frac{55*5}{360} = 0.763888889$

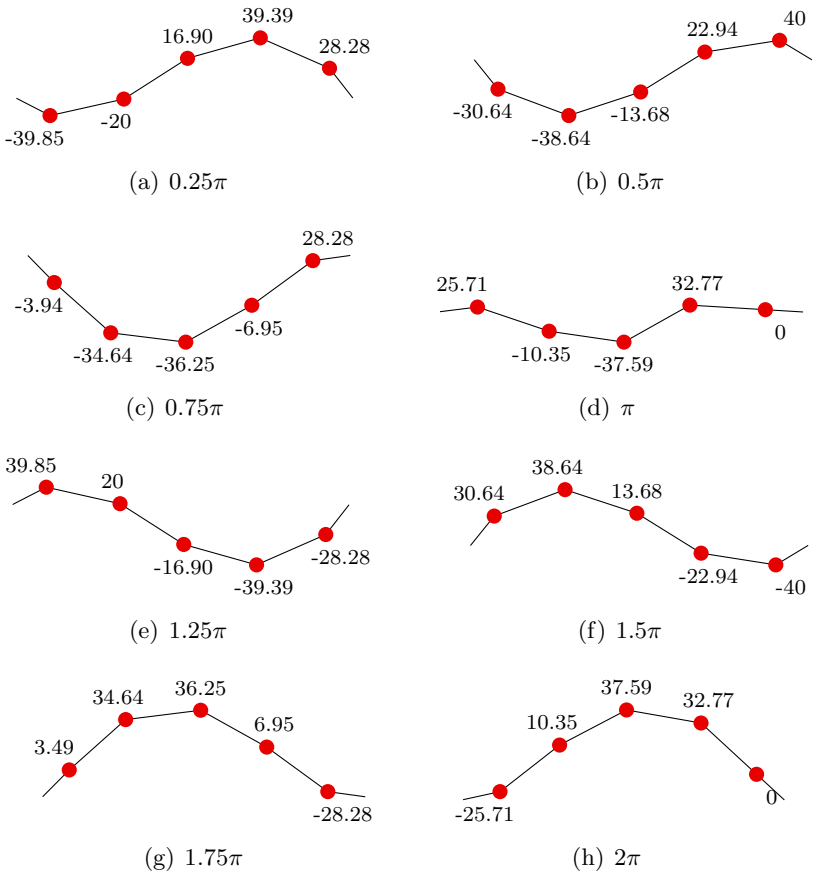
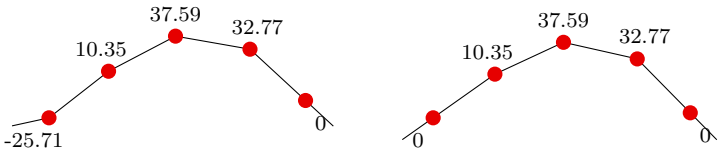


Figure 6.1: Different stages of a sine-pattern with uniform amplitudes applied to a chain of rigid links and joints. From (a) to (h) all stages of a whole locomotion cycle of 2*π is shown. The step-size between each stage shown is 0.25*π. The movements of the robot results in displacement from left to right.

experiment modulates values of amplitude of each of the five robot modules individually to achieve fastest possible locomotion. Each amplitude parameter is limited with an upper limit of 40 degree and a lower limit of 0 degree. The range in between is sampled with step-size of 0.25 degree. The idea is to look for a better solution than simply setting every amplitude to maximum value to achieve largest displacement possible. The search space of the formulated problem has more than 100 billion possible solutions[2]. For evaluation 30000 steps are simulated (30 seconds) and resulting scores in unit meters are compared. Average reference distance for uniform amplitudes of 40 degree is around 1.65 metres[3].
In general larger values of amplitude seem to result in larger displacement of the robot, since amplitude correlates with step-width. Increasing step-width, while stride-frequency is fixed, results in faster movement in legged walking. But in limbless locomotion static friction plays an important role in generating propulsion.

First tries of this experiment contained a mistake which led to an important discovery.

[2] $(\frac{40}{0.25}+1)^5 = 108,175,616,801$ solutions
[3] Average value is taken from 50 repetitions.



(a) Uniform amplitudes (b) Amplitude of last joint is zero

Figure 6.2: Uniform and optimized sine-patterns in comparison: In 6.2(a) a sine-pattern at $2^*\pi$ is shown. Amplitudes are all uniform. In 6.2(b) an optimized version of the same stage with individual amplitudes can be found. In simulations robots using amplitudes of version 6.2(b) had much greater displacement in the same amount of time.

The initial idea was to restrict amplitudes to range of 20 to 40 degrees. By mistake lower limit of amplitude of the last joint was set to 2 degrees. 15 of 20 optimization trials found the best solution by setting all joints to 40 except the last one which had the value of 2-3. This was the reason to expand the interval of valid values to something between 0 and 40 to allow joints to be fixed or move only very little. These settings produced best results when the amplitude of the tail-joint was set to zero or very small values. Figure 6.2 shows the difference at the most important part of the body movement. The tail has to anchor when the whole body tries to propulse itself in direction of movement. Slippage against direction of travel reduces efficiency of locomotion. In 6.2(b) static friction is increased by placing the tail part of robot's body in better orientation than in 6.2(a).

6.1.1 Results from great deluge algorithm

Trials of modified great deluge algorithm have been adjusted parameter values according to table 6.1.

Parameter	Value
Credits	10
Retries	200
Initial water-level	0.15
Rain	0.01
Confidence	20

Table 6.1: Parameter values of modified GD optimization in experiments where the shape of travelling-waves is improved. Credits, retries and confidence determine the behaviour of the algorithm to avoid to get stuck in local extrema.

Table 6.2 contains parameter values of ten of the best results from modified great deluge algorithm. Joints are enumerated from head to tail. It can easily be seen that the current robot performs best if the last joint is not moving. In contrast to the last joint others are moving with very high amplitude. With respect to largest displacement in given time, these combinations of amplitudes seem to produce the best results possible.

Amplitude	1	2	3	4	5	6	7	8	9	10
Joint 0	40	40	40	40	40	40	40	40	40	40
Joint 1	40	39.5	39.75	40	40	40	40	40	40	40
Joint 2	39.75	40	40	40	40	40	39.75	40	40	39.75
Joint 3	40	39.75	40	40	40	40	40	40	40	40
Joint 4	0	0	0	0.75	0	0	0	0	1.5	0
Displacement [m]	1.958	1.956	1.954	1.953	1.950	1.949	1.949	1.947	1.946	1.945

Table 6.2: Best 10 amplitude sets from GD-optimizations out of 30 trials.

In figure 6.3 all graphs of maximum scores versus number of iteration are plotted. Starting scores and learning rates are spread widely but in the end they nearly converge. Figure 6.4 shows worst case and best case maximum scores as well as the average of all 30 trials.

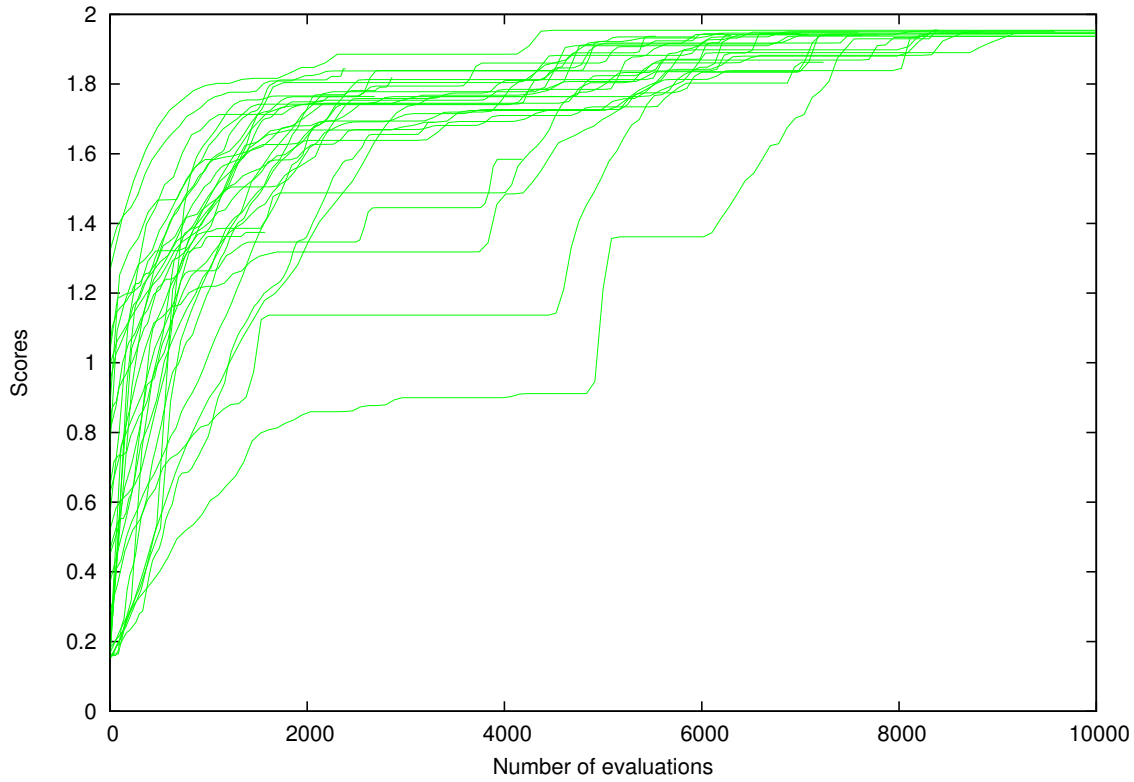


Figure 6.3: Maximum scores of 30 repetitions of amplitude optimization experiment with modified GD algorithm.

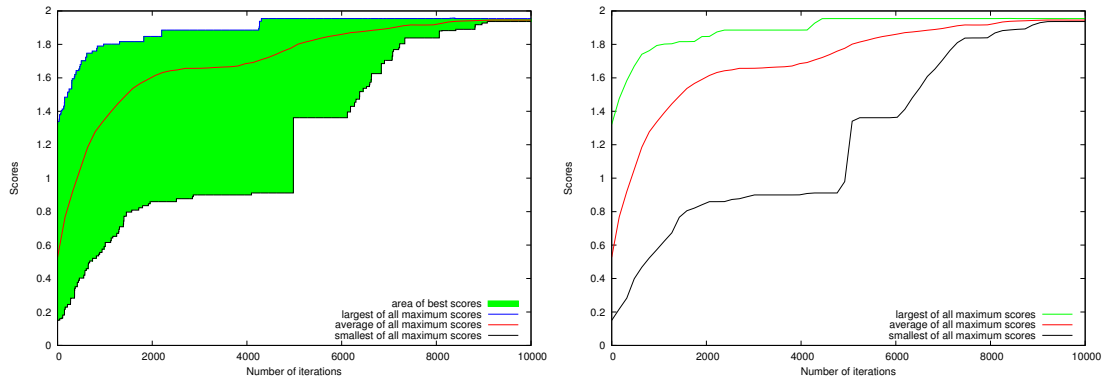


Figure 6.4: Minimum, maximum and average of all amplitude optimization experiments with modified GD algorithm.

6.1.2 Results from evolutionary programming

In table 6.3 values used to instantiate genetic algorithm for optimization of the current robots' travelling wave shape are summarized. Explanations of how to adjust genetic algorithms are given by Goldberg [21]. Results from table 6.4 are quite similar to those

Parameter	Value
Population-size	30
Probability of mutation	0.8
Probability of crossover	0.2
Convergence percentage	99%
Generations to convergence	50

Table 6.3: GA parameters used in travelling-wave optimization experiments. Each generation contains a population of individuals according to the population-size. The algorithm terminates if the mean score of last 50 generations converges to 99%.

of the modified great deluge algorithm but with slightly lower parameter values and lower scores.

In figure 6.5 it can be seen that all plots of max scores are similar and converge but with different results. A maximum possible score is not found in every trial. Single trials seem to get stuck in local maxima. As figure 6.6 shows the deviation from mean scores is quite low. The maximum scores converges in average at around 1.8. In the end all three graphs end up in the same point because there was only one experiment that had so many repetitions until the algorithm stopped.

Amplitude	1	2	3	4	5	6	7	8	9	10
Joint 0	37.5	39.5	38.75	39.75	39.25	37	38.25	39.75	37.75	38.75
Joint 1	39.25	40	40	38.75	39.5	39.5	38.75	39	38.5	39.75
Joint 2	39.25	39	38	40	39.75	39.25	40	40	40	40
Joint 3	39.5	39.25	38.25	37.25	39.25	38.75	39.75	36.75	39.5	30.25
Joint 4	1	2.25	0.5	0.5	0.75	2.25	0	0.75	3.25	0.75
Displacement [m]	1.878	1.874	1.865	1.846	1.839	1.828	1.822	1.820	1.816	1.806

Table 6.4: Best 10 amplitude sets taken from GA-optimizations out of 29 iterations.

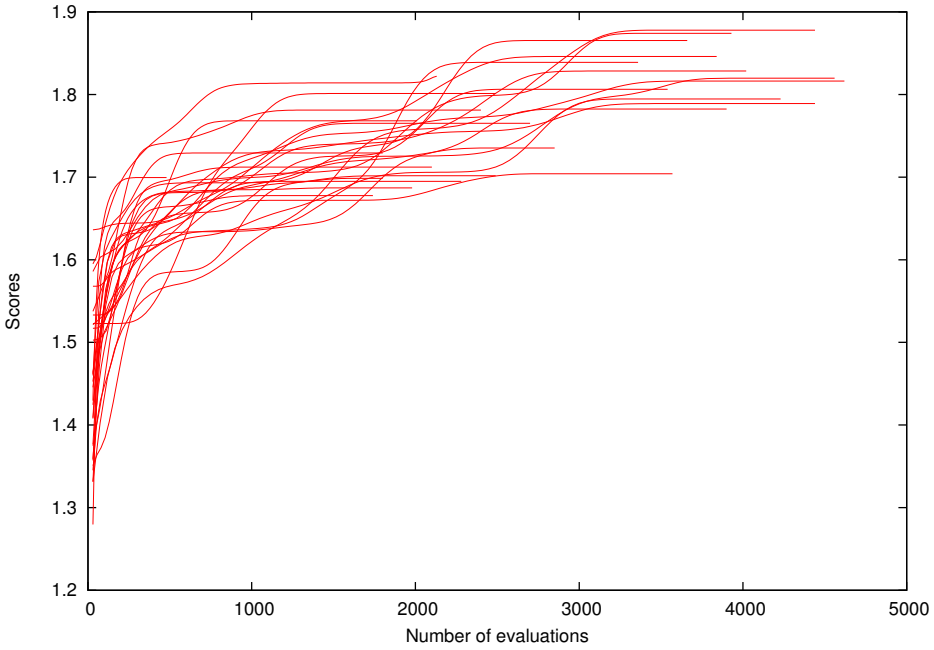


Figure 6.5: Maximum scores of 29 repetitions of GA-based amplitude optimization experiment.

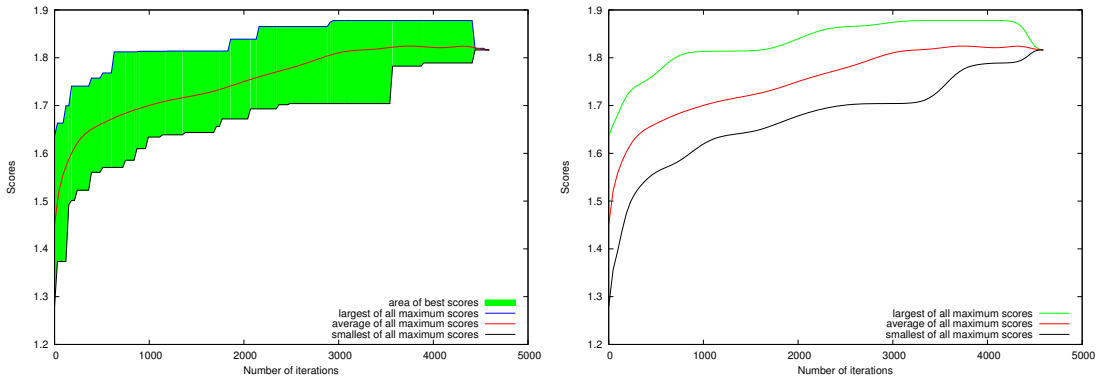


Figure 6.6: Minimum, maximum and average of GA-based amplitude optimization experiments.

6.2 Optimization of adaptive locomotion

In this category the focus lies on optimization of parameters regarding adaptive capabilities of robots' locomotional behaviour. As representative of environments with partially uneven terrain a special object were created. It is shown on figure 6.7.

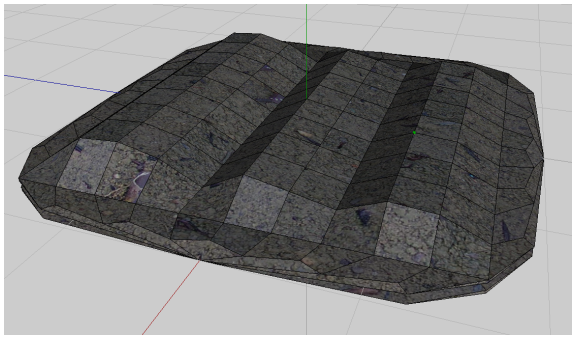


Figure 6.7: This artificial irregular shaped obstacle has been created for experiments with sensor-feedback-driven adaptive control algorithms. It can be used to simulate outdoor environments that are difficult to traverse.

For easier comparison the amount of simulated time was limited to one and a half minute[4]. A detailed robot model with five pitching joints and twelve touch sensors[5] at the lower side was used. Locomotion is generated by two different actuation modules. One is a simple sinusoidal generator[6] and the other one is an adaptive algorithm that generates offsets for each joint using global sensor feedback of all touch sensors[7]. Both actuation modules are combined using the control kernel of the proposed simulation and control system. Figure 4.6 explains the simple integration method. The sinusoidal generator is adjusted to have an amplitude of 30 degree and phase difference between neighbouring segments of 55 degree. This configuration implies a quite low center of gravity to assure stable movements, even if it does not offer smoothest locomotion possible, because there is only about three-quarter of a whole wave generated by body movements[8]. Improvements can be achieved by using more modules.

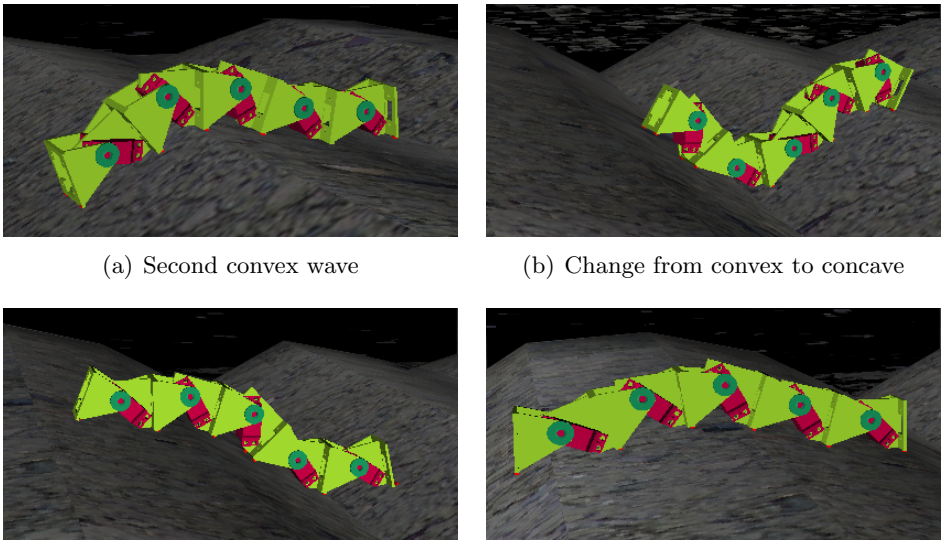
For modular robots in chainlike configuration without adaptive capabilities it is not possible to pass convex and concave obstacles. Using this object locomotional behaviour of modular robots can be improved in four different aspects at the same time.

1. starting locomotion with regular body-shape
2. passing convex shaped terrain
3. passing concave shaped terrain

[4] In this setup it equals 90000 simulation steps of physics engine
[5] Touch sensors are described on page 68.
[6] Described in section 3.3.1.
[7] The adaptive algorithm is designed by Guoyuan Li at the University of Hamburg and is part of his Ph.D-Thesis
[8] $55 * 5/360 = 0.763888889$

4. recovering shape to be efficient on flat terrain

In figure 6.8 four different stages of one successful climbing trial using the proposed algorithm optimized parameters are shown.



(a) Second convex wave

(b) Change from convex to concave

(c) Climbing next wave with high slope

(d) Top of last convex wave reached

Figure 6.8: Here different stages of a simulation where a modular robot with tactile sensors passes an obstacle. The chronological order is ascending from (a) to (d). Figure (a) shows the robot passing the second convex element. In (b) the robot has already mastered the following concave part of the ground. This is the most extreme situation, because applied offset values are very high in positive and negative direction. In (c) the concave part is nearly passed and the robot has already recovered from adaptivity of the concave section. (d) shows the robot on top of the last hill of the obstacle.

Appropriate parameter settings should allow to move efficiently in all four categories. Using fitness functions which reward fast travelling candidates stronger, results can be produced when trained on objects similar to figure 6.7. In table 6.5 information about the possible values of each parameter that has to be optimized is given.

Figure 6.9 shows the environmental set-up used for this optimization task. To visualize resulting scores from optimization, calculated by current fitness function, a yellow-red scale is shown. Each coloured segment represents one meter which equals one point of score. Good resulting sets of parameters are not only capable of passing the irregular shaped obstacle but also to move as far as possible along the scale in given time. Parameters that will be optimized in this set-up are all part of the adaptive algorithm. There is a threshold that defines when the robot got stuck. It determines when manipulation of the robot's base shape has to be applied. Then, there is the recovering speed parameter that defines how fast the robot is allowed to recover its base-shape. There are another eight parameters that specify the amount of change of offsets. Four are supposed to increase and another four to lower current values of offsets. Depending on the current state of the sensors, these are used in a control loop that calculates resulting relative joint positions.

Parameter	Minimum	Maximum	Step-Size	Number of possible values
Get-stuck	100	1000	25	37
Recovering	0.01	2	0.01	200
Constant 1	0.1	1	0.1	10
Constant 2	0.1	1	0.1	10
Constant 3	0.1	1	0.1	10
Constant 4	0.1	1	0.1	10
Constant 5	0.1	1	0.1	10
Constant 6	0.1	1	0.1	10
Constant 7	0.1	1	0.1	10
Constant 8	0.1	1	0.1	10

Table 6.5: Parameter configuration settings for optimization

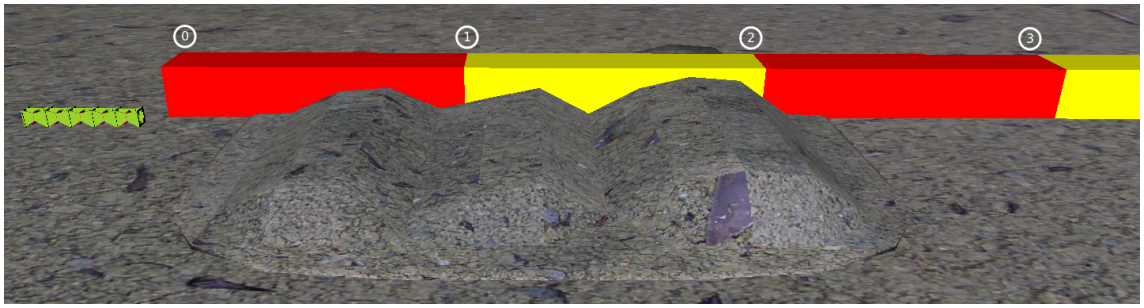


Figure 6.9: Simulation set-up with scale

Both optimization methods use iterations of the same simulation set-up to generate improved sets for these control parameters. Single runs consist of 90000 simulation steps. According to adjustments of stepsize in ODE physics engine of 0.001 the amount of time, simulated in each iteration, is 90 seconds. Having a look at the number of iterations used to reach stopping criterion of proposed optimization methods there are six to seven days that have to be simulated in total for each run to finish[9]. Brute-force methods would have to evaluate more than 700 billions[10] of possible solutions.

6.2.1 Results from great deluge algorithm

To obtain satisfying results, three parameters of the proposed modified great deluge algorithm have to be set to reasonable values. As shown in table 6.6 minimal score of the first satisfying solution, the so-called water-level, is 0.6 meter. This means that the first accepted solution allows the robot to reach at least the top of the first convex-shaped wave of the ground. In figure 6.9 scale relative to obstacle is shown. After a solution was accepted the water-level is increased by rain of strength 0.02. In this way the next score

[9] $6000 * 90$ seconds = 540000 seconds = 9000 minutes = 150 hours = 6.25 days
[10] $37 * 200 * 10^8 = 740,000,000,000$

Parameter	Value
Credits	10
Retries	200
Initial water-level	0.6
Rain	0.02
Confidence	20

Table 6.6: Modified GD-parameters in optimization of the adaptive algorithm.

that has to be reached is 0.62 meters. The confidence parameter that adjusts adaptive search radius is set to twenty. Ten credits are given for the case of bad luck. One credit is consumed when 200 failures in a row are passed.

In table 6.7 resulting parameters from the best ten optimization trials of proposed modified great deluge algorithm are shown.

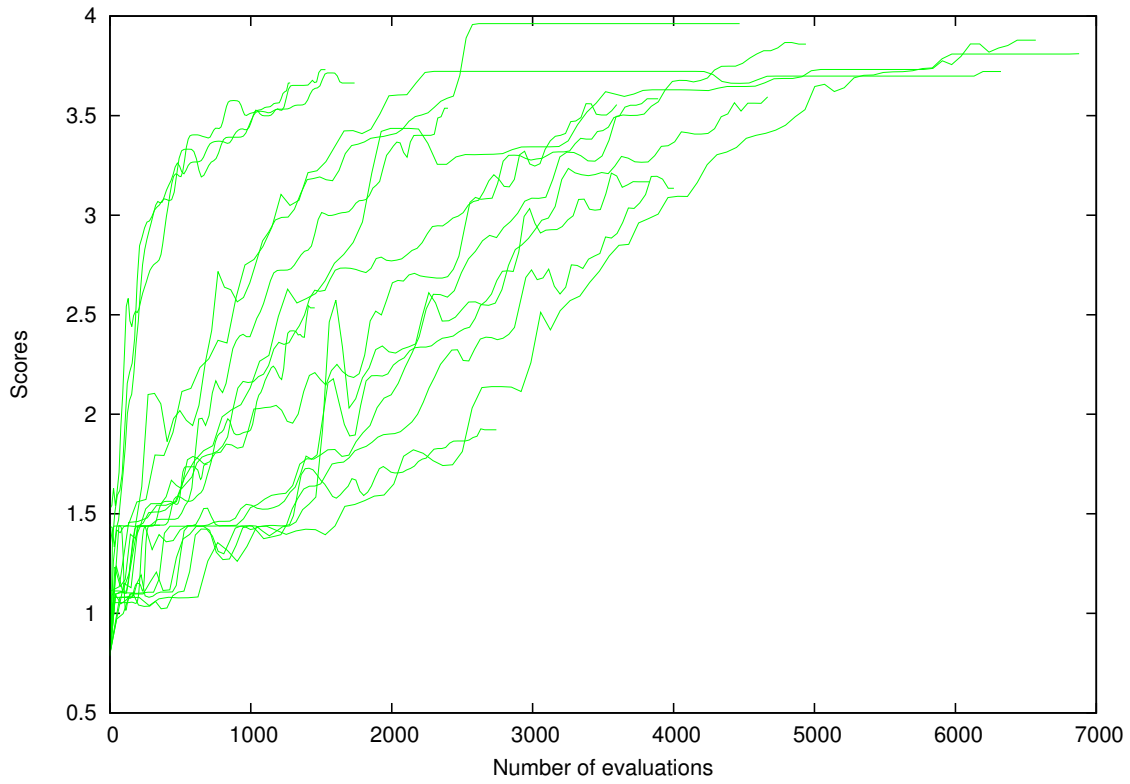


Figure 6.10: Maximum scores of 20 repetitions of irregular terrain experiment with modified GD algorithm.

Parameter	1	2	3	4	5	6	7	8	9	10
Get-stuck	110	85	185	160	160	185	60	160	135	360
Recovering	0.33	0.74	0.56	0.55	0.56	0.13	0.37	0.18	0.25	1.23
Constant 1	11.8	3.3	2.7	6.6	2.6	1.4	0.6	2.4	1.4	1.9
Constant 2	21.6	1.9	0.7	1.6	2.3	0.8	0.7	1.3	1.9	4.6
Constant 3	11.8	9.7	5.1	6.5	2.7	2.2	2.5	1.9	1.7	2.8
Constant 4	0.6	1.3	2.4	5.34	2	1	16.2	1.4	0.9	0.3
Constant 5	20.2	1.1	1.2	0.9	0.4	0.9	1.5	0.1	0.8	1.5
Constant 6	1.5	5.9	1.7	6.8	0.5	0.6	0.3	0.9	1.6	0.09
Constant 7	2.36	5.1	5.76	6.5	4.1	2.9	4.3	2.7	2.8	6
Constant 8	11	1.5	1.2	1.7	2.3	0.9	0.9	0.6	1	3.1
Displacement [m]	3.96	3.88	3.86	3.81	3.77	3.73	3.72	3.66	3.66	3.56

Table 6.7: Parameter values of candidates from best 10 GD-optimizations.

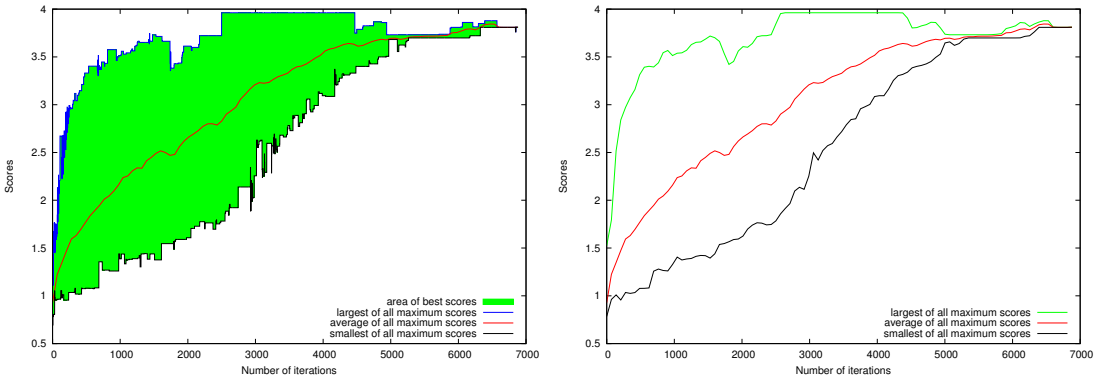


Figure 6.11: Minimum, maximum and average of all irregular terrain experiments using modified GD algorithm.

6.2.2 Results from evolutionary programming

Table 6.8 contains settings used for genetic algorithms of this experiment. In this case the crossover operator is disabled to achieve faster convergence. Optimization terminates when convergence occurs. It is defined as the ratio of the n-th previous best-of-generation score to the current best-of-generation score. In this case it is 99%. *Generations to convergence* determine how many generations are taken into account for convergence test. Each generation is populated with 40 individuals that have to be evaluated. In table 6.9 best results from genetic algorithms are shown. As expected in this kind of experiment resulting values and the correlations among them are not understandable for humans. But they can be used for adaptive locomotion to pass a certain obstacle. All plots maximum scores in Figure 6.12 show off a similar shape and are very close together. In the first 1000-2000 iterations learning rates are very high but then converge to zero. Then further improvements happen very seldom.
Figure 6.13 shows the area between worst case and case learning behaviour. Using the described genetic algorithm the difference between best and worst result is quite small.

Parameter	Value
Population-size	40
Probability of mutation	1
Probability of crossover	0
Convergence percentage	99%
Generations to convergence	50

Table 6.8: GA parameters in parameter optimization of adaptive algorithm.

Parameter	1	2	3	4	5	6	7	8	9	10
Get-stuck	175	125	100	125	175	175	125	150	125	250
Recovering	0.09	0.13	0.2	0.14	0.11	0.06	0.21	0.05	0.02	0.03
Constant 1	1	0.7	0.9	0.9	1	0.8	1	0.9	0.4	0.9
Constant 2	0.9	0.9	0.7	1	0.7	0.8	0.6	1	1	0.9
Constant 3	0.7	1	1	1	0.7	1	1	0.1	0.4	0.4
Constant 4	1	0.5	0.2	1	0.1	0.9	0.3	1	0.2	0.3
Constant 5	0.1	0.5	0.4	0.7	0.3	0.8	0.4	0.7	0.3	0.3
Constant 6	0.5	0.5	0.6	0.8	0.3	0.2	0.4	0.5	0.7	0.3
Constant 7	1	0.9	1	1	0.9	0.8	1	0.9	1	1
Constant 8	0.7	0.7	0.7	1	0.8	0.6	0.9	0.6	0.5	0.6
Displacement [m]	3.19	3.11	3.08	3.07	3.05	3.02	3.02	3.02	3.02	3.01

Table 6.9: Parameter values of candidates from best 10 GA-optimizations out of twenty.

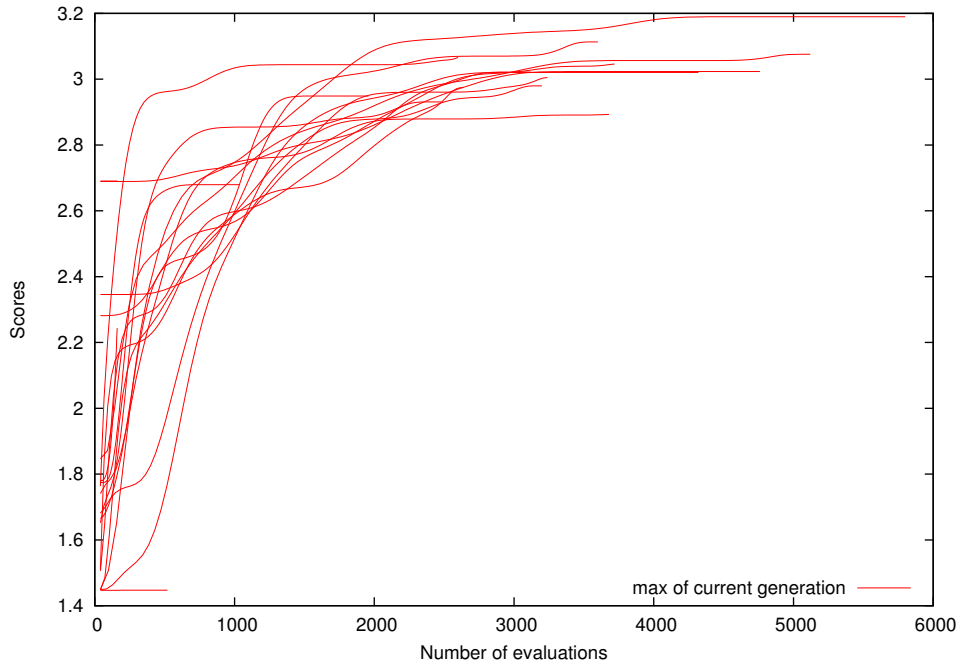


Figure 6.12: Maximum scores of 20 repetitions of irregular terrain experiment with GA.

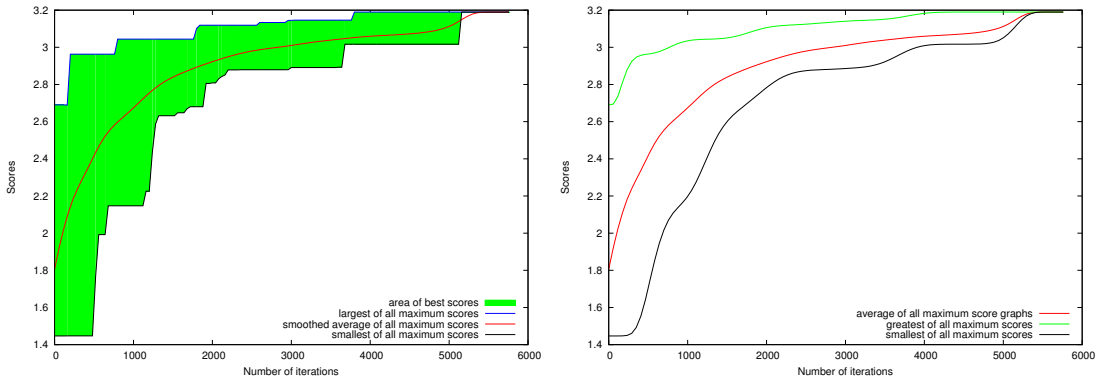


Figure 6.13: Minimum, maximum and average of all irregular terrain experiments using GA.

6.3 Summary

This chapter presented results from two different experiments where locomotion patterns were optimized with respect to fast locomotion. Since both experiments successfully produced optimized results the proposed system can be regarded as a useful contribution for modular robotic research. Both optimization methods in both experiments found better solutions for the parameters that had to be optimized then generally possible by trial-and-error. Moreover, the meaning of the parameters of the locomotion algorithm in the second experiment are not well understood, in contrast to the first experiment, but they were also successfully optimized. In the following chapter the quality of results from both optimization methods are compared to each other. After that benefits of the proposed work for research are discussed.

Discussion

7

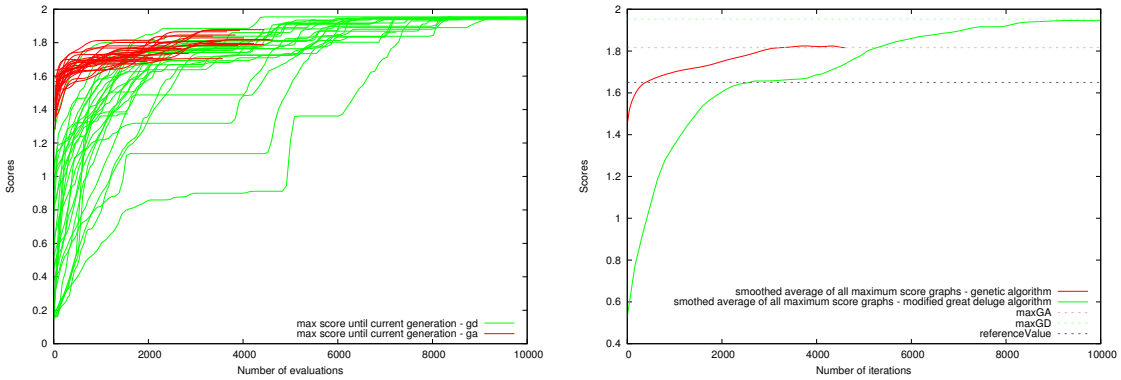
In the following, the purpose and usability of the proposed modular robotic simulation and control system is discussed. Not only its usefulness but also its limits will be explained. Section 7.1 compares the results of the two different optimization methods used for the experiments in chapter 6. Then, in chapter 7.2 these results are placed in the big picture of this thesis and they are used to justify the contribution of this work to modular robot science. The chapter ends with ideas for future work to improve the usability of the system and to increase its applicability for modular robot science in research and education by extending its number of functions.

7.1 Comparison of results from both methods

In the following, results from both experiments using two different optimization methods are compared with respect to quality and runtime. Aim of these experiments is to show the plausibility and the usefulness of the proposed modular robotic simulation system. In the first experiment, meaning of parameters that have to be optimized are easy to understand. In this way readers are able to discuss the results more easily. The second experiment shows how parameters can be optimized even if users do not know their meanings.

7.1.1 Travelling wave shape

Figure 7.1 shows learning behaviour of both optimization methods in variable amplitude experiments. In 7.1(a) for example maximum scores versus number of iteration within optimization process are shown. Each red graph represents one complete optimization trial using GAs. Each green graph shows the maximum scores of one modified GD optimization run. While each GA trial was terminated until 5000 evaluations were performed, great deluge algorithm is capable of further improvements of the results. The averages of all optimization trials, shown in 7.1(b), shows the differences more clearly. Learning-rate with GAs is much faster in the first 5000 repetitions than in modified GD algorithm. But above 5000 evaluations convergence of resulting scores appears. Learning performed by the proposed modified GD algorithm is able to produce better results if more time is given. In mean modified GD algorithm outperforms GAs at around 5000 iterations.



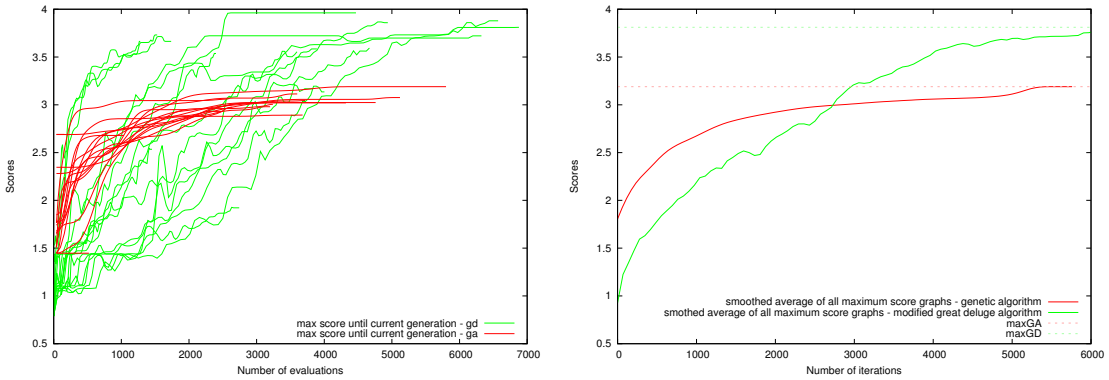
(a) All maximum scores

(b) Averages of both methods

Figure 7.1: Comparison of learning rates in wave shape optimization. In (a) all maximum scores of each iteration of both experiments are shown. Red graphs represent scores of GA-runs and green graphs scores of GD-runs. Figure (b) shows the smoothed averages of both optimization methods. The black dotted line marks the distance travelled by a robot using the same locomotion method but without individually optimized amplitudes. Red and green dotted lines mark largest distances reached by optimized parameters.

7.1.2 Adaptive locomotion

The second experiment can be characterized as much more complicated and challenging for optimization algorithms as well as for human understanding. It is hard to describe what exactly the parameters that have to be optimized do in particular. The solution space is much bigger, we do not know anything about correlations between these parameters and tasks, that have to be fulfilled are much more complex. This explains the bigger difference between averages of simulation results. Figure 7.2 shows differences of learning behaviour using two different optimization methods. In figure 7.2(a) it can be seen that red lines lay all very close together. These belong to results from trials with genetic algorithms. In contrast to this, the green graphs representing resulting maximum scores of proposed modified GD algorithm, are spread more wide. These also show off in much better results. In figure 7.2(b) it can easily be seen that great deluge algorithm outperforms GA optimization after around 3000 iterations. Although results from GAs in earlier iterations were better in experiments described above.
To compare best results of both optimization methods, a modified screenshot shown in figure 7.3 can be used. There are marks on the scale that indicate maximum distances travelled by using optimized parameter sets of both techniques. In addition there is a mark that shows a reference distance, travelled by the current robot but using the same locomotion method without any adaptivity and on flat terrain. It can be seen that the best result of modified GD algorithm is quite close to normal locomotion without passing irregular shaped object. This can only be achieved by fast and efficient adaptive behaviour of the shape adjusting algorithm used by the robot.



(a) All maximum scores

(b) Average of both methods

Figure 7.2: Comparison of learning rates in adaptive locomotion optimization. In (a) all maximum scores of each iteration of both experiments are shown. Red graphs represent scores of GA-runs and green graphs scores of GD-runs. Figure (b) shows the smoothed averages of both optimization methods. Red and green dotted lines mark largest distances reached by optimized parameters.

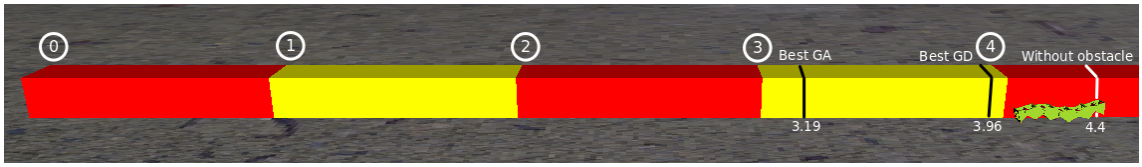


Figure 7.3: The largest distances travelled in 90000 simulation steps of traversing obstacle experiment are marked in this figure. Each coloured bar has the length of one meter. Best travelled distances of both optimization methods can be compared with the size of the robot. In addition the distanceis marked that is travelled by the same robot with the same configuration but without the obstacle.

7.1.3 Conclusion

In both experiments improvements of locomotion were performed. The variable amplitude experiments, using two different optimization techniques found combinations of amplitudes that were better than just setting maximum values for the given environment. Learning with GAs was faster in the beginning than with modified GD algorithm. With the increasing number of evaluations, modified GD algorithm found even better solutions while GAs converged earlier.
Analysing results of the adaptive locomotion algorithm and using sensor feedback on irregular terrain, it can be found that in both cases good results were produced. Using resulting parameter sets from optimization, the current robot was able to overcome the irregular shaped obstacle sufficiently. If the problem is formulated as traversing an obstacle the two methods can be successfully used. As in the first experiment GAs have better performance in the beginning of the process of optimization. Later, modified GD

algorithm runs outperform GAs. In this experiment the gap between best scores of every optimization strategy is much bigger than in the first experiment. Practically this means that by the help of the best set of parameters it is possible to change the locomotional behaviour extremely fast. The experiment starts on regular flat terrain, continues over an obstacle of irregular shape and finally ends of flat terrain again. Especially switching between surfaces with different properties is a challenging task. In addition the irregular obstacle shows concave parts as well as convex parts which increases the difficulty.

7.2 Benefits

This diploma thesis describes a fully integrated modular robotic simulation, optimization and control system with focus on bio-inspired locomotion generation. Benefits, arising from its usage for research and education in locomotion generation of chain-like modular robots, are pointed out. The core system is based on OpenRAVE and ODE physics engine. Both are widely used and accepted simulation tools that can be used to simulate robotic tasks with real world application. As explained by Diankov and Kuffner [18] the precision is high enough to make results from simulations applicable to real world situations. Features of the proposed system are an easy-to-use GUI with two different modes that allows experts and beginners to set up a simulation. This includes robot configuration, environment creation, locomotion behaviour definition and composition and even optimization of newly created locomotion techniques. The system structure is flexible enough to be extended as needed at many points. Its usability and flexibility makes it applicable for research and education. One of the greatest benefits for researchers lies in the possibility to optimize hardly understandable parameters with complex correlations among others. This works by the help of standardization of control algorithms, using a common interface for locomotion generation. It enables the system to treat all control parameters in a similar way. Thus even new locomotion modules that have just been added can be optimized in the same way as algorithms, that are part of the standard set of the system. Since the configuration GUI is responsible for setting up everything necessary, the system could also be used by students that are new to modular robotic locomotion. They can create new locomotion techniques, watch and evaluate the results.
The control GUI can be used to test new set-ups and especially new control algorithms with the help of visual feedback from 3d-viewer and live-plots of control data, before starting an optimization. But it is also possible to check the results after optimization is finished in the same way.

7.3 Future work

Depending on current goals both GUIs (configuration and control GUI) can be extended to implement more functionality from the core system. For usage as an educational platform its robustness should be improved regarding the execution order of user input. Important tasks for the future are summarized:

- work on sensor fusion in order to select autonomously proper locomotion patterns (work on intelligent behaviour)

- parallel optimization methods
  - distributed version of GD algorithm
  - parallel evaluation of single populations in the GA
- extension of the core system
  - integration of more kind of sensors
  - implementation of more fitness functions (energy efficiency)
  - integration of on-line optimization to learn from real-world experiments
- extension to the configuration GUI
  - easy import of new robot models
  - increasing robustness of the GUIs
- extension of the control GUI
  - more intuitive visualization of sensors

Benefits for education and research arise from easy exchange of robot models with designs created by users interactively or by importing external robot models.
The implementation of more sensors would help to work on sensor fusion techniques in the field of modular robotic locomotion. In this way robust autonomous behaviour of modular robots in complex terrain can be achieved.
Finally it can be taken into account that on-line optimization can be integrated to learn real-world experiments like Marbach and Ijspeert [40] presents. Combined optimization of simulated and real world experiments could further improve locomotional strategies. Further benefits would arise from usage of other fitness functions in locomotion optimization in order to create economic gaits. With consideration of increased stability and high energy efficiency limbless modular robots in chain-like configuration come closer to real world applications in industry, surveillance and space science.

7.4 Summary of the thesis

From related work (see section 3.5) follows that there is no other open simulation system with focus on modular robot locomotion design and optimization that is flexible enough to be useful for the application with new modular robot systems and locomotion algorithms. Its design concepts allow to extend it as needed for specific purposes. Experiments (see chapter 6) showed that the proposed system is suitable for the optimization of locomotion patterns. Two easy-to-use GUIs make it even suitable for educational purposes. In this way this thesis represents a valuable contribution to modular robotic locomotion research.

Bibliography

[1] Aldebaran. Nao. http://www.aldebaran-robotics.com/en/. Accessed: 01/08/2013. 41

[2] R.M.N. Alexander. *Principles of Animal Locomotion*. Princeton University Press, 2003. ISBN 9780691086781. URL http://books.google.de/books?id=2anZacrFaxoC. 6, 8, 9, 10, 23

[3] D. Balakrishna, P. Sailaja, R.V.V.P. Rao, and B. Indurkhya. A novel human robot interaction using the wiimote. In *Robotics and Biomimetics (ROBIO), 2010 IEEE International Conference on*, pages 645 –650, dec. 2010. doi: 10.1109/ROBIO.2010.5723402. 64

[4] J H Belanger and B A Trimmer. Combined kinematic and electromyographic analyses of proleg function during crawling by the caterpillar manduca sexta. *J Comp Physiol A*, 186(11):1031–9, 2000. ISSN 0340-7594. URL http://www.biomedsearch.com/nih/Combined-kinematic-electromyographic-analyses-proleg/11195279.html. 14, 15

[5] Jim H. Belanger and Barry A. Trimmer. Combined kinematic and electromyographic analyses of proleg function during crawling by the caterpillar manduca sexta. *Journal of Comparative Physiology A*, 186(11):1031–1039, 2000. ISSN 0340-7594. doi: 10.1007/s003590000160. URL http://dx.doi.org/10.1007/s003590000160. 19

[6] David Berrigan and David J. Pepin. How maggots move: Allometry and kinematics of crawling in larval diptera. *Journal of Insect Physiology*, 41(4):329 – 337, 1995. ISSN 0022-1910. doi: 10.1016/0022-1910(94)00113-U. URL http://www.sciencedirect.com/science/article/pii/002219109400113U. 13, 14

[7] J Brackenbury. Fast locomotion in caterpillars. *J Insect Physiol*, 45(6):525–533, 1999. ISSN 1879-1611. URL http://www.biomedsearch.com/nih/Fast-locomotion-in-caterpillars/12770337.html. 14, 35

[8] John Brackenbury. Caterpillar kinematics. *Nature*, 390(6659):453–453, 1997. doi: 10.1038/37253. URL http://dx.doi.org/10.1038/37253. 12, 13, 14

[9] T.M. Casey. Energetics of caterpillar locomotion: biomechanical constraints of a hydraulic skeleton. *Science*, 252(5002):112–4, 1991. 13

[10] Kai-Hsiang Chang and Yung-Yaw Chen. Efficiency on snake robot locomotion with constant and variable bending angles. In *Advanced robotics and Its Social Impacts, 2008. ARSO 2008. IEEE Workshop on*, pages 1 –5, aug. 2008. doi: 10.1109/ARSO.2008.4653626. 62, 79

[11] Jian Chen, Eugen Richter, and Jianwei Zhang. Bio-inspired caterpillar-like climbing robot. In NathanF. Lepora, Anna Mura, HolgerG. Krapp, PaulF.M.J. Verschure, and TonyJ. Prescott, editors, *Biomimetic and Biohybrid Systems*, volume 8064 of *Lecture Notes in Computer Science*, pages 359–361. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-39801-8. doi: 10.1007/978-3-642-39802-5_34. URL `http://dx.doi.org/10.1007/978-3-642-39802-5_34`. 30

[12] Wei-Cheng Chen and Ren-Yuan Lyu. A hmm-based fundamental motion synthesis approach for gesture recognition on a nintendo triaxial accelerometer. In *Signal Processing and Communication Systems (ICSPCS), 2011 5th International Conference on*, pages 1 –5, dec. 2011. doi: 10.1109/ICSPCS.2011.6140869. 64

[13] Pei-Ying Chiang, May chen Kuo, J. Lee, and C.-C.J. Kuo. Wiistick: Enhancing motion recognition capability for wii systems. In *Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on*, pages 1445 –1448, may 2009. doi: 10.1109/ISCAS.2009.5118038. 64

[14] Ro Crespi, André Badertscher, André Guignard, and Auke Jan Ijspeert. Amphibot i: an amphibious snake-like robot. *Robotics and Autonomous Systems*, 50:163–175, 2005. 28

[15] Cyberbotics. Webots 7. `http://www.cyberbotics.com/overview`. Accessed: 10/04/2013. 40, 67

[16] D. Daidie, O. Barbey, A. Guignard, D. Roussy, F. Guenter, A. Ijspeert, and A. Billard. The dof-box project: An educational kit for configurable robots. In *Advanced intelligent mechatronics, 2007 IEEE/ASME international conference on*, pages 1 –6, sept. 2007. doi: 10.1109/AIM.2007.4412571. 40

[17] Rosen Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, September 2010. 41, 67

[18] Rosen Diankov and James Kuffner. Openrave: A planning architecture for autonomous robotics. *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34*, page 79, 2008. 41, 96

[19] Erich Gamma, Richard Helm, Ralph E. Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995. ISBN 978-0-201-63361-0. 72

[20] Carl Gans. Locomotion of limbless vertebrates: Pattern and evolution. *Herpetologica*, 42(1):pp. 33–46, 1986. ISSN 00180831. URL `http://www.jstor.org/stable/3892232`. 14, 20, 21

[21] David E. Goldberg. *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA, 2002. ISBN 1402070985. 37, 83

[22] Juan González-Gómez. Openmr: Modular robots plug-in for openrave. http://www.iearobotics.com/wiki/index.php?title=OpenMR:_Modular_Robots_plug-in_for_Openrave. Accessed: 23/01/2013. 30, 31, 41, 68

[23] Juan González-Gómez, Houxiang Zhang, Eduardo Boemo, and Jianwei Zhang. Locomotion Capabilities of a Modular Robot with Eight Pitch-Yaw-Connecting Modules. In *9th International Conference on Climbing and Walking Robots. CLAWAR06*, September 2006. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.87.9422. 32, 33

[24] M.K. Habib. *Bioinspiration and Robotics: Walking and Climbing Robots*. I-Tech Education and Publishing, 2007. ISBN 9783902613158. URL http://books.google.de/books?id=IsmwPgAACAAJ. 23, 40

[25] Fernando Herrero-Carrón, Francisco B. Rodríguez, and Pablo Varona. Study and application of Central Pattern Generator circuits to the control of a modular robot. Master's thesis, Escuela Politécnica Superior, Universidad Autónoma de Madrid, 2007. 34, 35

[26] HiBot. Acm-r5h. http://www.hibot.co.jp/en/products/robots_1/acm-r5h_33. Accessed: 28/05/2013. 27

[27] S. Hirose and H. Yamada. Snake-like robots [tutorial]. *Robotics Automation Magazine, IEEE*, 16(1):88 –98, march 2009. ISSN 1070-9932. doi: 10.1109/MRA.2009.932130. 25

[28] Shigeo Hirose. Hirose – fukushima robotics lab. http://www-robot.mes.titech.ac.jp/robot/snake/koryu1/koryu1_e.html. Accessed: 24/01/2013. 26

[29] Shigeo Hirose. *Biologically inspired robots : snake-like locomotors and manipulators / Shigeo Hirose ; translated by Peter Cave and Charles Goulden.* Oxford University Press, Oxford ; New York :, 1993. ISBN 0198562616. URL http://www.loc.gov/catdir/enhancements/fy0640/92034986-t.html. 25, 26

[30] Shigeo Hirose and Akio Morishima. Design and Control of a Mobile Robot with an Articulated Body. *The International Journal of Robotics Research*, 9(2):99–114, April 1990. doi: 10.1177/027836499000900208. URL http://dx.doi.org/10.1177/027836499000900208. 25, 26

[31] Rhosgobel: Radagast's home. Manduca proleg attaching to ground. http://ossiriand.net/rhosgobel/images/manduca-5th-hind-end.jpg. Accessed: 05/08/2013. 19

[32] Auke Jan Ijspeert. Central pattern generators for locomotion control in animals and robots: A review. *Neural Networks*, 21(4):642 – 653, 2008. ISSN 0893-6080. doi: 10.1016/j.neunet.2008.03.014. URL http://www.sciencedirect.com/science/article/pii/S0893608008000804. ¡ce:title¿Robotics and Neuroscience¡/ce:title¿. 34, 37

[33] B C Jayne. Muscular mechanisms of snake locomotion: an electromyographic study of the sidewinding and concertina modes of crotalus cerastes, nerodia fasciata and elaphe obsoleta. *J Exp Biol*, 140:1–33, 1988. ISSN 0022-0949. URL `http://www.biomedsearch.com/nih/Muscular-mechanisms-snake-locomotion-electromyographic/3204332.html`. 20

[34] A. Kamimura, H. Kurokawa, E. Yoshida, K. Tomita, S. Kokaji, and S. Murata. Distributed adaptive locomotion by a modular robotic system, m-tran ii. In *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2370 – 2377 vol.3, sept.-2 oct. 2004. doi: 10.1109/IROS.2004.1389763. 29, 35, 40

[35] Dennis Krupke, Guoyuan Li, Jianwei Zhang, Houxiang Zhang, and Hans Petter Hildre. Flexible modular robotic environment for research and education. In *26th European Conference on Modelling and Simulation. ECMS2012*, 2012. 41, 51

[36] Guoyuan Li, Houxiang Zhang, F. Herrero-Carron, H.P. Hildre, and Jianwei Zhang. A novel mechanism for caterpillar-like locomotion using asymmetric oscillation. In *Advanced Intelligent Mechatronics (AIM), 2011 IEEE/ASME International Conference on*, pages 164 –169, july 2011. doi: 10.1109/AIM.2011.6027047. 35

[37] Guoyuan Li, Houxiang Zhang, Fernando Herrero-Carrón, and Jianwei Zhang. Locomotion of limbless robots using novel biomimetic neural circuits. In *International Conference on Intelligent Robots and Systems*. IEEE/RSJ, 2011. 35

[38] H W Lissmann. Rectilinear locomotion in a snake (boa occidentalis). *J Exp Biol*, 26: 368–379, 1950. ISSN 0022-0949. URL `http://jeb.biologists.org/content/26/4/368`. 20

[39] A. Lorente-Leal, J.A. Ferna andndez Rodrigues, and J.M. Montero. Development of a wiimote-based gesture recognizer in a microprocessor laboratory course. In *Education Engineering (EDUCON), 2010 IEEE*, pages 451 –455, april 2010. doi: 10.1109/EDUCON.2010.5492543. 64

[40] Daniel Marbach and Auke Jan Ijspeert. Online optimization of modular robot locomotion. In *In Proceedings of the IEEE International Conference on Mechatronics and Automation (ICMA 2005*, pages 248–253, 2005. 40, 97

[41] Kiyotoshi Matsuoka. Sustained oscillations generated by mutually inhibiting neurons with adaptation. *Biological Cybernetics*, 52(6):367–376, October 1985. ISSN 0340-1200. doi: 10.1007/BF00449593. URL `http://dx.doi.org/10.1007/BF00449593`. 35

[42] M. Noeske, D. Krupke, N. Hendrich, Jianwei Zhang, and Houxiang Zhang. Interactive control parameter investigation of modular robotic simulation environment based on wiimote-hci's multi sensor fusion. In *Multisensor Fusion and Integration for Intelligent Systems (MFI), 2012 IEEE Conference on*, pages 478 –483, sept. 2012. doi: 10.1109/MFI.2012.6343044. 50, 64

[43] Annenberg Learner Journey North. Microscopic picture caterpillars' cremaster. http://www.learner.org/jnorth/images/graphics/monarch/cremaster01.gif. Accessed: 06/09/2013. 19

[44] The University of Michigan Mobile Robotics Lab. Ot-4. http://mrl.engin.umich.edu/00MoRob_6.html. Accessed: 28/05/2013. 28

[45] S.N. Purkayastha, N. Eckenstein, M.D. Byrne, and M.K. O'Malley. Analysis and comparison of low cost gaming controllers for motion analysis. In *Advanced Intelligent Mechatronics (AIM), 2010 IEEE/ASME International Conference on*, pages 353 – 360, july 2010. doi: 10.1109/AIM.2010.5695830. 64

[46] K. J. Quillin. Kinematic scaling of locomotion by hydrostatic animals: ontogeny of peristaltic crawling by the earthworm lumbricus terrestris. *The Journal of experimental biology*, 202 (Pt 6):661–74, March 1999. URL http://www.ncbi.nlm.nih.gov/pubmed/10021320. 32

[47] robotechn intelligent technology. Cubo robot. http://www.robotechn.com/cn/products.php?id=2. Accessed: 05/09/2013. 30, 31

[48] C. Smith and H.I. Christensen. Wiimote robot control using human motion models. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 5509 –5515, oct. 2009. doi: 10.1109/IROS.2009.5354593. 64

[49] Russel Smith. Open dynamics engine. http://www.ode.org. Accessed: 23/01/2013. 40, 41, 67

[50] Barry Trimmer and Jonathan Issberner. Kinematics of soft-bodied, legged locomotion in manduca sexta larvae. *Biol Bull*, 212(2):130–42, 2007. ISSN 0006-3185. URL http://www.biomedsearch.com/nih/Kinematics-soft-bodied-legged-locomotion/17438205.html. 14, 15, 16, 17, 18

[51] L. I. van Griethuijsen and B. A. Trimmer. Kinematics of horizontal and vertical caterpillar crawling. *Journal of Experimental Biology*, 212:1455–1462, 2009. doi: 10.1242/jeb.025783. 29

[52] Matthew Wall. Matthew's genetic algorithm library. http://lancet.mit.edu/ga/. Accessed: 23/01/2013. 50, 69

[53] Wei Wang, Yingying Wang, Kun Wang, Houxiang Zhang, and Jianwei Zhang. Analysis of the kinematics of module climbing caterpillar robots. In *Advanced Intelligent Mechatronics, 2008. AIM 2008. IEEE/ASME International Conference on*, pages 84 –89, july 2008. doi: 10.1109/AIM.2008.4601639. 30

[54] R. Yamashina, M. Kuroda, and T. Yabuta. Caterpillar robot locomotion based on q-learning using objective/subjective reward. In *System Integration (SII), 2011 IEEE/SICE International Symposium on*, pages 1311 –1316, dec. 2011. doi: 10.1109/SII.2011.6147638. 32, 40

[55] Houxiang Zhang, J. González-Gómez, and Jianwei Zhang. A new application of modular robots on analysis of caterpillar-like locomotion. In *Mechatronics, 2009. ICM 2009. IEEE International Conference on*, pages 1 –6, april 2009. doi: 10.1109/ICMECH.2009.4957149. 31, 32

[56] H.X. Zhang, J. González-Gómez, S.Y. Chen, and J.W. Zhang. Embedded intelligent capability of a modular robotic system. In *Robotics and Biomimetics, 2008. ROBIO 2008. IEEE International Conference on*, pages 2061 –2066, feb. 2009. doi: 10.1109/ROBIO.2009.4913319. 26, 30

[57] École polytechnique fédérale de Lausanne. Amphibot. `http://biorob.epfl.ch/amphibot`. Accessed: 28/05/2013. 28

Glossary

body-wave
wave travelling through longitudinal body 29, 104

clasper
kind of gripper, used by caterpillars to anchor its tail to the ground in horizontal and vertical walking 12, 19, 104

duty factor
percent of total cycle which a given foot is on the ground 6, 8, 13, 16, 17, 22, 23, 104

dynamic class loading
technique of extending programs at runtime 72, 104

electrolyte
solution able to conduct electric current 104

excitatory
interconnection type of neurons, used to trigger firing of connected neurons 104

factory
object to create instances from classes loaded at runtime without concrete knowledge of their constructors 72, 104

Froude number
dimensionless number comparing inertia and gravitational forces 9, 10, 104

gait
pattern of movement or locomotion mode like *walking* and *running* 6, 8, 11, 21, 32, 35, 104

gait transition
switching from one to another gait 8, 34, 104

hemocoel
system of cavities between the organs of arthropods and molluscs through which the blood circulates 16, 104

hydrostatics
branch of fluid mechanics that studies fluids at rest 16, 104

inhibitory
interconnection type of neurons, used to create behaviour similar to mutual exclusion 104

invertebrate
animals without backbone 20, 104

lamprey
kind of jawless fish 28, 29, 104

metabolic cost of transport
amount of energy, used for locomotion (often determined by measuring oxygen consumption) 14, 104

mollusc
group of invertebrates that includes squid, octopuses, cuttlefish, snails, slugs, oysters, scallops, and many more 16, 104

motoneuron
neurons located in the central nervous system, used to directly or indirectly control muscles 16, 19, 104

pneumatic bellow
actuators that can be filled with air like a kind of balloons 28, 104

proleg
passive leg of caterpillars, used for attaching to the ground with miniaturized hooks 12, 14–19, 22, 23, 104

self-registering type
optional data types that register automatically to a factory and after that are available to the program 72, 104

septum
cell wall dividing bodies into segments 16, 104

stride
complete cycle of periodic movement 6–15, 22, 23, 104

thoracic
between head and abdomen 104

travelling wave
wave shape e.g. sinusoidal, used to produce locomotion in limbless hyper-redundant body structures 12, 22, 32, 64, 79, 83, 104

true leg
normal leg of a catterpillar that can be actuated actively 12, 104

vertebra
bony structure in the backbone or spine 20, 104

Acronyms

COG
center of gravity 6, 23, 32, 33, 104

CPG
central pattern generator 3, 26, 29, 32–35, 37, 40, 41, 71, 72, 104

GA
genetic algorithm 37, 40, 41, 48, 50, 77, 93–97, 104

GD
great deluge algorithm 38, 41, 50, 93–95, 97, 104

GUI
graphical user interface v, 1, 2, 43, 45, 61, 62, 65, 69, 96, 97, 104

HRI
human robot interface 51, 104

ODE
Open Dynamics Engine 30, 67, 68, 87, 96, 104

Base Interfaces

A

Listing A.1: Abstract base interface of actuation modules

```
#ifndef ACTUATIONMODULE_H
#define ACTUATIONMODULE_H

#include <iostream>
#include <math.h>
#include <cstdlib>
#include <map>
#include <string>

#include <QVector>
#include <QDebug>
#include <QString>


class ActuationModule{
public:
    //! Is intended to initialize the ActuationModule to a valid state, while
        the simulation is running.
    virtual void reset() = 0;

    //! Sets all parameters, used for the calculation of angles, to the latest
        values.
    inline void setProperties(const QVector <double> propertyVector);

    //! Changes the value of a single parameter.
    inline void setParameter(const QString name, double value);

    //! Returns the current value of a parameter with given name.
    inline double getCurrentParameterValue(const QString parameterName);

    //! Delivers angles for each joint.
    inline QVector <double> getAngleData();

    //! Gives the descriptions (e.g. names) of the parameters.
    inline QVector <QString> getPropertiesInfo();

    //! Returns all current parameter values.
    inline QVector <double> getProperties ();

    //! Returns all current values of the calculated data.
    inline QVector <QVector <double> > getInternalData();

    //! Gives the descriptions (e.g. names) of the calculated data.
    inline QVector <QString> getInternalDataInfo();

    //! Sets the angle 'value' for specific joint with index 'numOfJoint'.
    inline void setAngle(int numOfJoint, double value);
```

```
        //! Gets the current angle of a specific joint with index 'numOfJoint'.
        inline double getCurrentAngle(int numOfJoint);
49
        //! Gets the old angle of a specific joint with index 'numOfJoint'.
        inline double getOldAngle(int numOfJoint);

        //! Computes the values for the next step of the simulation.
54      virtual void computeAngles() = 0;

        inline QVector <QVector <double> > calculateValues();

        //! Returns the name of this module.
59      inline QString getName();

        //! Returns the type of this module.
        inline QString getType();

64      //! Copies the object and returns a pointer to this copy.
        virtual ActuationModule* cloneSelf() = 0;

    protected:
        int _numOfConnectedJoints; //!< represents the number of joints for which
            this module is used
69
        QVector <double> _properties; //!< holds the parameters of this module
        QVector <QString> _propertiesInfo; //!< holds the descriptions of the
            parameters of this module
        QVector <QVector <QVector <double> > > _internalData; //!< holds all kind of
             computed values for the last and current step
        QVector <QString> _internalDataInfo; //!< holds descriptions for each kind
            of used value of the computation of the angles
74      QVector <double> _angleData; //!< are the current angles for each joint (
            computed values, not real angles!)

        QString _name; //!< the name of this module
        QString _type; //!< the type of this module

79      //! Creates and initializes the vectors for the internal data with the
            correct number of elements.
        virtual inline void Init(int);
    };


84  // typedef to make it easier to set up the factory
    typedef ActuationModule* maker_pt(int joints);
    // our global factory
    extern "C" std::map<std::string, maker_pt*, std::less<std::string> > cpgFactory;

89  #endif // ACTUATIONMODULE_H
```

Listing A.2: Abstract base interface of pattern generators

```
1
  #ifndef PATTERNGENERATOR_H
  #define PATTERNGENERATOR_H

  #include "actuationModule.h"
6

  class PatternGenerator : public ActuationModule{
  public:
      //! Applies initial values for the used movement generator
11    virtual void reset() = 0;

      //! Computes the angles for the constructor given number of joints
      virtual void computeAngles() = 0;

16    //! Returns a pointer to a copy of this object
      virtual ActuationModule* cloneSelf() = 0;

      //! Applies a specific value to a certain type of data in the _internalData.
      virtual inline void setDistortion(const int jointNr, const int dataType,
          const double value);
21
      //! Calculates the phase difference between neighboring joint angle
          functions and the frequency for each joint.
      virtual inline void computePhaseDifference(int);

      //! Enables concurrent computing of the angles of neighboring modules with
          OpenMP.
26    virtual inline void setParallel(bool parallel);

      //! Clears the _internalData and applies the result of a noise-function to
          it.
      virtual inline void addNoise();

31 protected:
      QVector <int> _timeSteps;       //!< Counted time steps from the simulator,
          stored for each joint

      QVector <double> _oldAngles;    //!< Angles of each joint of the previous
          time step

36    QVector <double> _cycleDuration;//!< Number of calculation steps used for a
          whole oscillation cycle (2*pi)

      QVector <double> _phaseDiff;    //!< The phase difference of each joint to
          its next neighbor (last is connected to the first)

      bool _parallel;                 //!< Sets the state of using OpenMP (true
          means calculating in parallel)
41
      //! Calculates Gaussian White Noise.
      virtual inline double gaussianWhiteNoise();
  };

46 #endif // PATTERNGENERATOR_H
```

Listing A.3: Abstract base interface of sensing modules

```
#ifndef SENSINGMODULE_H
#define SENSINGMODULE_H

#include "actuationModule.h"
#include "dataHandler/sensorWatcher_tactile.h"
#include "dataHandler/sensorWatcher_sensibot.h"
#include "dataHandler/sensorwatcher_laser2d.h"

using namespace OpenRAVE;


class SensingModule : public ActuationModule{
public:
    //! Applies initial values for the used movement generator
    virtual void reset() = 0;

    //! Computes the values for the next step of the simulation.
    virtual void computeAngles() = 0;

    //! returns a pointer to a copy of this object
    virtual ActuationModule* cloneSelf() = 0;

    //! Assigns a sensor data handler to this SensingModule.
    virtual void setSensorWatcher(SensorWatcher* watcher);

    //! Sets a boolean to true, when the sensors are initialized and ready to
        use. Useful for virtual sensors.
    inline void setSensorsReady();

    //! Sets the behaviour of the algorithm for the offsets.
    inline void setBehaviour(QVector<short int> genome);

    //! Gets the behaviour of the algorithm for the offsets.
    inline QVector<short int> getBehaviour();

    //! Sets the real robot mode (actually only sensibot is supported)
    inline void setRealRobotMode();

protected:
    SensorWatcher* _sensorWatcher;  //!< the assigned sensor data handler

    bool _sensorsReady; //!< true,  if the sensors handled by _sensorWatcher are
         ready (used for virtual sensors)

    QVector<short int> _behaviour;  //!< This QVector determines the behavior
        for calculating the angles.

    bool _isRealRobot;  //!< true, if a real robot is used
};

#endif // SENSINGMODULE_H
```

Listing A.4: Abstract base interface of robot controls

```
#ifndef ROBOTCONTROL_H
#define ROBOTCONTROL_H

#include <string>
#include <vector>

class RobotControl{

public:
    virtual int connect()=0;
    virtual void disconnect()=0;

    virtual void enableServos(bool servosOn)=0;
    virtual void sendCommand(std::string command, unsigned char* values=NULL)=0;
    virtual void sendCommand(std::string command, unsigned char values, unsigned
        char destinationID)=0;
    virtual void sendCommandByValue(std::string command, unsigned char values=
        NULL)=0;

    virtual void requestAllInformation(unsigned char* returnValues, int
        numberOfReturnedChars)=0;
    virtual void requestInformation(std::string type, std::vector<bool>*
        returnValue)=0;
    virtual void requestInformation(std::string type, std::vector<int>*
        returnValue)=0;
    virtual void requestInformation(std::string type, std::vector<double>*
        returnValue)=0;

    virtual void test()=0;

    inline int getNumberOfSensormodules();
    inline int getNumberOfSensorsPerModule();
    inline int getNumberOfJoints();
    inline std::string getTopology();


protected:
    int _numOfSensorModules, _numOfJoints, _numOfSensorsPerModule;
    std::string _topology;

    virtual void resetCommandBuffer()=0;
};

#endif // ROBOTCONTROL_H
```

Listing A.5: Abstract base interface of remote controls

```
#ifndef REMOTE_H
#define REMOTE_H

#include <QString>
#include <QList>
#include <map>
#include <string>
#include "remoteStructures.h"

using namespace RemoteControl;

class Remote : public QObject{
Q_OBJECT
public:
    virtual int connect() = 0;
    virtual int disconnect() = 0;

    virtual void sendCommand(QString command) = 0;
    virtual remoteData readAllData() = 0;

    inline bool isConnected();
    inline QList<QString> getAvailableControlDevices();
    inline QList<QString> getAvailableCommands();

protected:
    bool _connectionStatus;
    QList<QString> _availableControlDevices;
    QList<QString> _availableCommands;

public slots:
    virtual void receiveCommand(QString command) = 0;
};


typedef Remote* maker_rc_ptr();
// the global factory
extern "C" std::map<std::string, maker_rc_ptr*, std::less<std::string> >
    remoteFactory;

#endif // REMOTE_H
```

File Formats

B

B.1 Configuration file formats

Listing B.1: Example of top-level configuration files

```
<simulation>
    <robot>./configuration/robotXML/MFIdemo_5py.robot.xml</robot>
    <actuation>
        ./configuration/actuation/MFIdemo_5py_2012.09.06_17:38.actuation.xml
    </actuation>
    <environment>./configuration/environmentXML/MFIdemo.env.xml</environment>
    <simulationProperties>
        <showViewer>1</showViewer>
        <parallelComputing>0</parallelComputing>
        <storingData>1</storingData>
        <mode>Single Run Simulation</mode>
        <accuracy>30</accuracy>
        <stepWidth>0.001</stepWidth>
    </simulationProperties>
</simulation>
```

Listing B.2: Example of robot configuration files

```
<Robot name="MFIdemo_5py">
    <KinBody name="kMFIdemo_5py">
        <Kinbody file="./configuration/robotXML/base/TailP.kinbody.xml"/>
        <Kinbody prefix="1"
            file="./configuration/robotXML/base/PY.kinbody.xml"/>
        <Kinbody prefix="2" file="./configuration/robotXML/base/YP.kinbody.xml">
            <translation>0 0.072 0</translation>
        </Kinbody>
        <Kinbody prefix="3" file="./configuration/robotXML/base/PY.kinbody.xml">
            <translation>0 0.144 0</translation>
        </Kinbody>
        <Kinbody prefix="4" file="./configuration/robotXML/base/YP.kinbody.xml">
            <translation>0 0.216 0</translation>
        </Kinbody>
        <Kinbody file="./configuration/robotXML/base/HeadP.kinbody.xml">
            <translation>0 0.288 0</translation>
        </Kinbody>
        <Joint type="hinge" name="J1">
            <Body>Tail</Body>
            <Body>1Seg</Body>
            <offsetfrom>1Seg</offsetfrom>
            <axis>1 0 0</axis>
            <maxtorque>0.4</maxtorque>
            <maxvel>4.5</maxvel>
            <limitsdeg>-90 90</limitsdeg>
```

```
            </Joint>
            <Joint type="hinge" name="J2">
                <Body>1Seg</Body>
                <Body>2Seg</Body>
                <offsetfrom>2Seg</offsetfrom>
                <axis>0 0 1</axis>
                <maxtorque>0.4</maxtorque>
                <maxvel>4.5</maxvel>
                <limitsdeg>-90 90</limitsdeg>
            </Joint>
            <Joint type="hinge" name="J3">
                <Body>2Seg</Body>
                <Body>3Seg</Body>
                <offsetfrom>3Seg</offsetfrom>
                <axis>1 0 0</axis>
                <maxtorque>0.4</maxtorque>
                <maxvel>4.5</maxvel>
                <limitsdeg>-90 90</limitsdeg>
            </Joint>
            <Joint type="hinge" name="J4">
                <Body>3Seg</Body>
                <Body>4Seg</Body>
                <offsetfrom>4Seg</offsetfrom>
                <axis>0 0 1</axis>
                <maxtorque>0.4</maxtorque>
                <maxvel>4.5</maxvel>
                <limitsdeg>-90 90</limitsdeg>
            </Joint>
            <Joint type="hinge" name="J5">
                <Body>4Seg</Body>
                <Body>Head</Body>
                <offsetfrom>Head</offsetfrom>
                <axis>1 0 0</axis>
                <maxtorque>0.4</maxtorque>
                <maxvel>4.5</maxvel>
                <limitsdeg>-90 90</limitsdeg>
            </Joint>
        </KinBody>
    </Robot>
```

Listing B.3: Example of tactile sensor configuration files

```
    <sensor type="TactileSensor">
        <power>1</power>
    </sensor>
```

Listing B.4: Example of laser sensor configuration files

```
    <sensor type="BaseLaser2D">
        <minangle>-1</minangle>
        <maxangle>1</maxangle>
        <resolution>1</resolution>
        <maxrange>1</maxrange>
        <scantime>1</scantime>
        <render>1</render>
        <power>1</power>
    </sensor>
```

Listing B.5: Example of actuation configuration files

```
<actuation>
    <validRobot robotName="MFIdemo_5py.robot.xml"/>
    <actuationModule name="DirectJointAccess">
        <jointConfiguration joints="01010"/>
    </actuationModule>
    <actuationModule name="PatternGenerator_sinusoidal">
        <jointConfiguration joints="10101"/>
    </actuationModule>
</actuation>
```

Listing B.6: Example of environment configuration files

```
<Environment>
    <physicsengine type="ode">
        <odeproperties>
            <gravity>0 0 -9.81</gravity>
            <selfcollision>1</selfcollision>
        </odeproperties>
    </physicsengine>
    <KinBody name="floor">
        <Body type="static">
            <Translation>0 0 -0.003</Translation>
            <Geom type="box">
                <extents>8 8 0.003</extents>
                <diffuseColor>0.3 1 0.3</diffuseColor>
                <ambientColor>0.3 1 0.3</ambientColor>
                <transparency>0.05</transparency>
                <friction>0.1</friction>
            </Geom>
        </Body>
    </KinBody>
</Environment>
<!--Pose: ,0 0 0 0,0 0 0,-->
```

Listing B.7: Example of genetic algorithm configuration files

```
<geneticAlgorithm>
    <properties>
        <popsize>30</popsize>
        <pmut>0.01</pmut>
        <pcross>0.9</pcross>
        <nConvergence>50</nConvergence>
        <pConvergence>0.99</pConvergence>
        <flushFrequency>1</flushFrequency>
        <gaMode>1</gaMode>
        <numOfParametersToOptimize>2</numOfParametersToOptimize>
        <registeredParameters>
            <parameter number="0" controlAlgorithm="Global Adaptive Touch"
                parameter="Get-stuck-threshold"/>
            <parameter number="1" controlAlgorithm="Global Adaptive Touch"
                parameter="back to zero constant"/>
        </registeredParameters>
    </properties>
    <genomeConfig number="0" allJoints="1" genomeType="1" valueType="1"
        valueSpace="1" activeJoints="11111">
        <upperBounds>800</upperBounds>
        <lowerBounds>50</lowerBounds>
        <boundsIncrements>25</boundsIncrements>
        <lowerBoundTypes>2</lowerBoundTypes>
        <upperBoundTypes>2</upperBoundTypes>
```

```
            <uniform>1</uniform>
        </genomeConfig>
        <genomeConfig number="1" allJoints="1" genomeType="1" valueType="1"
            valueSpace="1" activeJoints="11111">
            <upperBounds>1</upperBounds>
            <lowerBounds>0.01</lowerBounds>
            <boundsIncrements>0.01</boundsIncrements>
            <lowerBoundTypes>2</lowerBoundTypes>
            <upperBoundTypes>2</upperBoundTypes>
            <uniform>1</uniform>
        </genomeConfig>
    </geneticAlgorithm>
```

B.2 Data file format

Listing B.8: Example of data file format

```
    <dataRoot>
        <Header>
            <Global_Adaptive_Touch types="4" parts="5">
                <typeNames>desired_angle realtime simtime stepcount</typeNames>
            </Global_Adaptive_Touch>
            <PatternGenerator_sinusoidal types="7" parts="5">
                <typeNames>Angle Time Phase_Difference Frequency realtime simtime
                    stepcount</typeNames>
            </PatternGenerator_sinusoidal>
            <tactile types="7" parts="12">
                <typeNames>x-normals y-normals z-normals realtime simtime
                    stepcount</typeNames>
            </tactile>
            <Robot_Data types="16" parts="13">
                <typeNames>speed x_pos y_pos z_pos alpha beta gamma
                    potential_energy average_com_speed desired_joint_positions
                    real_joint_positions servo_currents servo_speeds realtime
                    simtime stepcount</typeNames>
            </Robot_Data>
        </Header>
        <round number="0">
            <Global_Adaptive_Touch>
                <Frame index="0" realtime="1361368880613" simtime="30">
                    <desired_angle>0 0 0 0 0</desired_angle>
                </Frame>
                <Frame index="1" realtime="1361368880632" simtime="61">
                    <desired_angle>0 0 0 0 0</desired_angle>
                </Frame>
            ...
            </Global_Adaptive_Touch>
            <PatternGenerator_sinusoidal>
                <Frame index="0" realtime="1361368880613" simtime="30">
                    <Angle>5.80648 -29.297 -7.10298 28.9827 8.38557</Angle>
                    <Time>0.031 0.031 0.031 0.031 0.031</Time>
                    <Phase_Difference>0 0 0 0 0</Phase_Difference>
                    <Frequency>inf inf inf inf inf</Frequency>
                </Frame>
                <Frame index="1" realtime="1361368880632" simtime="61">
                    <Angle>11.3934 -27.4934 -12.6101 26.9354 13.802</Angle>
                    <Time>0.062 0.062 0.062 0.062 0.062</Time>
                    <Phase_Difference>0 0 0 0 0</Phase_Difference>
                    <Frequency>inf inf inf inf inf</Frequency>
                </Frame>
            ...
```

```
        </PatternGenerator_sinusoidal>
        <tactile/>
        <Robot_Data>
43          <Frame index="0" realtime="1361368880613" simtime="30">
                <speed>3.24042e-18 3.24042e-18 3.24042e-18 3.24042e-18 3
                    .24042e-18</speed>
                <x_pos>5.00018 5.00018 5.00018 5.00018 5.00018</x_pos>
                <y_pos>-1.86 -1.86 -1.86 -1.86 -1.86</y_pos>
                <z_pos>-2.69615 -2.69615 -2.69615 -2.69615 -2.69615</z_pos>
48              <alpha>0.706795 0.706795 0.706795 0.706795 0.706795</alpha>
                <beta>0.0209787 0.0209787 0.0209787 0.0209787 0.0209787</beta>
                <gamma>-0.0209787 -0.0209787 -0.0209787 -0.0209787
                    -0.0209787</gamma>
                <potential_energy>-0.0674037 -0.000114443 -0.000114443
                    -0.134807 -0.000114258</potential_energy>
                <average_com_speed>0 0 0 0 0</average_com_speed>
53              <desired_joint_positions>1.58101e-322 1.36856e-321 6.15379e-313
                    6.15379e-313 6.93178e-310</desired_joint_positions>
                <real_joint_positions>32 0 37 0 1
                    .66857e+09</real_joint_positions>
                <servo_currents>32 0 101 0 1.34227e+09</servo_currents>
                <servo_speeds>2.54029e+07 0 37 0 0</servo_speeds>
            </Frame>
58          <Frame index="1" realtime="1361368880632" simtime="61">
                <speed>0.005093 0.005093 0.005093 0.005093 0.005093</speed>
                <x_pos>5.00066 5.00066 5.00066 5.00066 5.00066</x_pos>
                <y_pos>-1.86 -1.86 -1.86 -1.86 -1.86</y_pos>
                <z_pos>-2.71193 -2.71193 -2.71193 -2.71193 -2.71193</z_pos>
63              <alpha>0.706313 0.706313 0.706313 0.706313 0.706313</alpha>
                <beta>0.0334883 0.0334883 0.0334883 0.0334883 0.0334883</beta>
                <gamma>-0.0334884 -0.0334884 -0.0334884 -0.0334884
                    -0.0334884</gamma>
                <potential_energy>-0.0677983 -0.000115152 -0.000115152
                    -0.135597 -0.000114816</potential_energy>
                <average_com_speed>0 0 0 0 0</average_com_speed>
68              <desired_joint_positions>1.58101e-322 1.36856e-321 6.15379e-313
                    6.15379e-313 6.93178e-310</desired_joint_positions>
                <real_joint_positions>32 0 37 0 1
                    .66857e+09</real_joint_positions>
                <servo_currents>32 0 69 0 3</servo_currents>
                <servo_speeds>2.54029e+07 0 37 0 1.34228e+09</servo_speeds>
            </Frame>
73          ...
        </Robot_Data>
    </round>
    ...
</dataRoot>
```

This work is motivated by my work as a student assistant in modular robots' locomotion research. The student position was supported by the DFG BICCA project[1].

BICCA Project

The focus of the BICCA project lies on the design, development and programming of a climbing caterpillar robot that is able to climb on smooth vertical surfaces like glass using an attachment system. The attachment system is based on a semi-passive system that uses motor-driven vibrating suction cups to stick to surfaces. Releasing from the attached surface is possible by using valves that are connected to microcontroller boards. Re-attachment is performed by high frequency vibration that removes the air from the suction cups. Not only dynamic creation of appropriate locomotion patterns, but also distributed intelligent control of the attachment system is a challenging task in this project.

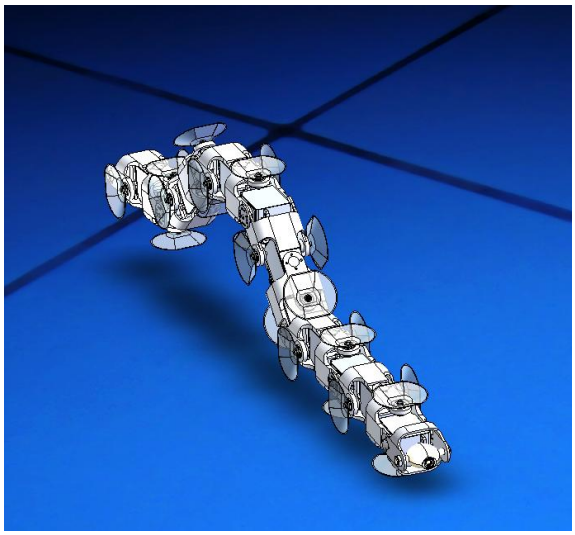


Figure B.1: This concept art shows first ideas of a climbing caterpillar robot.

[1] *B*iologically *I*nspired *C*limbing *Ca*terpillar Project funded by DFG (*D*eutsche *F*orschungs*g*emeinschaft)

Acknowledgments

I want to thank the following people for their support. Their contributions were very helpful to create this time consuming peace of work:

- **Prof. Ph.D. Jianwei Zhang**
  Support of the thesis.
- **Prof. Ph.D. Houxiang Zhang**
  Motivation to start research of modular robot locomotion. Help and guidance to produce a first scientific paper.
- **Dipl. Inf. Eugen Richter**
  Corrections of language and structure.
- **Ph.D. Guoyuan Li**
  Cooperative work on locomotion algorithms. Many suggestions on GUI improvements.
- **Dipl. Inf. Doreen Jirak**
  Very detailed corrections of structure and language.
- **Ph.D. Norman Hendrich**
  Fruitful criticism on optimization strategies.
- **Martin Noeske**
  Structural hints to improve the proposed system and to find bugs.
- **B.Sc. Lasse Einig**
  Contribution of an awesome LaTeX-macro for generating schematics of body waves.
- **Dipl. Inf. Denis Klimentjew**
  Strategical scientific guidance that helped me to make right decisions in questions regarding my career as student.
- **Dipl. Inf. Benjamin Adler**
  Qt hints on programming.
- **Dipl. Ing. Vlad Ciobanu**
  Thanks for finding a bug.
- **My parents, Ina Krupke and Rudi Krupke**
  Many years of financial support and their strong believe that it will end good.
- **My beloved girlfriend Anika Blanck**
  Answering annoying questions and listening again and again. In addition helping with corrections of written English language and of course emotional support.

Ich versichere, dass ich die vorstehende Arbeit selbstständig und ohne fremde Hilfe angefertigt und mich anderer als der im beigefügten Verzeichnis angegebenen Hilfsmittel nicht bedient habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht.

Ich bin mit einer Einstellung in den Bestand der Bibliothek des Departments Informatik einverstanden.

Hamburg, 30. September 2013

Dennis Krupke