



ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE



BIOLOGICALLY INSPIRED
ROBOTICS GROUP (BIRG)

Evolution and Online Optimization of Central Pattern Generators for Modular Robot Locomotion

Daniel Marbach

daniel.marbach@epfl.ch

Master Thesis. January 7th, 2005

School of Computer and Communication Sciences,
Swiss Federal Institute of Technology Lausanne.

Supervisor: Prof. Auke Jan Ijspeert
Biologically Inspired Robotics Group

Contents

CONTENTS	3
INTRODUCTION	1
MODULAR ROBOTICS	3
2.1 INTRODUCTION	3
2.1.1 <i>Classification of modular robots</i>	4
2.1.2 <i>Promises and Applications</i>	5
2.2 MODULE DESIGN	7
2.2.1 <i>CONRO</i>	7
2.2.2 <i>M-TRAN</i>	8
2.2.3 <i>PolyBot</i>	9
2.2.4 <i>YaMoR</i>	10
2.3 MODULAR ROBOT CONTROL	11
2.3.1 <i>Gait control tables and phase automata</i>	13
2.3.2 <i>Role-based control</i>	13
2.3.3 <i>Hormone-based control</i>	14
2.3.4 <i>Constraint-based control</i>	15
2.3.5 <i>Reconfiguration</i>	16
2.4 SENSORS IN MODULAR ROBOTS	18
BIO-INSPIRED LOCOMOTION CONTROL	21
3.1 VERTEBRATE LOCOMOTION	21
3.1.1 <i>Locomotion through rhythmic activities</i>	22
3.1.2 <i>The importance of mechanical dynamics</i>	22
3.1.3 <i>Symmetry – a fundamental characteristic of locomotion</i>	23
3.1.4 <i>Locomotion gaits</i>	23
3.1.5 <i>Neurophysiology of vertebrate locomotion</i>	24
3.1.6 <i>The role of sensory feedback</i>	25
3.2 NONLINEAR OSCILLATORS WITH BALANCED COUPLINGS	26
3.2.1 <i>Introduction</i>	26
3.2.2 <i>Standalone nonlinear oscillator</i>	27
3.2.3 <i>Coupled nonlinear oscillators</i>	29
3.2.4 <i>Predicting the phase difference of two oscillators</i>	31
3.2.5 <i>Convergence of two coupled nonlinear oscillators</i>	36
3.2.6 <i>The amplitude difference</i>	40
3.2.7 <i>Nonlinear oscillators with energy balanced couplings</i>	41
CO-EVOLUTION OF CONFIGURATION AND CONTROL	45
4.1 INTRODUCTION	45
4.1.1 <i>Evolutionary approaches to modular robot control</i>	46
4.1.2 <i>Co-evolution of morphology and control</i>	47
4.2 THE SIMULATION ENVIRONMENT.....	48
4.3 GENETIC ALGORITHM	49
4.3.1 <i>Genetic encoding</i>	50
4.3.2 <i>Genetic operators</i>	56

4.3.3	<i>Fitness function</i>	56
4.3.4	<i>Initialization and detection of convergence</i>	57
4.3.5	<i>Selection and replacement</i>	57
4.4	ASYNCHRONOUS DISTRIBUTED CONTROL	59
ONLINE OPTIMIZATION OF LOCOMOTION		61
5.1	INTRODUCTION TO OPTIMIZATION	62
RESULTS.....		65
6.1	FITNESS FUNCTION	65
6.2	GENETIC ENCODING	66
6.3	NONLINEAR VS. HARMONIC OSCILLATORS	67
6.4	GENETIC ALGORITHMS.....	68
6.5	ENERGY BALANCED COUPLINGS.....	69
6.6	EXAMPLES OF EVOLVED ROBOTS.....	70
CONCLUSIONS AND FUTURE WORK		73
REFERENCES.....		76

Chapter 1

Introduction

This Thesis presents a bio-inspired approach to modular robot locomotion. The goal of the project is self-organization of locomotion for autonomous modular robots. In contrast to previous research in modular robotics, I include the morphology in the self-organizing process. For my experiments I use a realistic simulation of YaMoR, the modular robot that is currently being developed at the Biologically Inspired Robotics Group (BIRG). In the near future, the results could be tested on the hardware itself.

The motivation of my research is two-fold. On one side, I believe that modular robotics is a perfect framework to build novel self-organizing and adaptive machines by applying bio-inspired methodologies. As artificial systems become increasingly complex, more and more resources are required to design, produce and maintain them. In addition, machines are expected to operate in more complex and unpredictable environments. This makes it increasingly difficult or even impossible for the engineer to identify the requirements and functionalities *a priori*. Therefore, autonomous machines must be self-organizing and adaptive. However, adaptation should not be limited to the controller of the machine. In order to adapt to a new task or a new environment, a machine must be able to autonomously change its morphology. Modular self-reconfigurable robots fit perfectly in this picture because they are versatile, adaptive and reliable.

On the other side, modular robots are an ideal test bed for research in complex, distributed and synergetic systems. Thus, I also see my research as being part of a more general effort that aims at a better understanding of complex, self-organizing systems in nature. By taking inspiration from nature to design autonomous, self-organizing machines we can also verify models from biologists and pose new hypothesis that must in turn be verified by biologists. In other words, the bio-inspired approach is mutually beneficial for both computer sciences and biology.

I believe that autonomous modular robots can be designed by implementing the three biological models of self-organization that explain the emergence of complex, autonomous organisms in nature: Phylogeny (P), Ontogeny (O) and Epigenesis (E). This classification of bio-inspired methodologies is known as the POE model [Sanchez et al. 1997, Tempesti et al. 2002]. An example of a system that incorporates all three POE axes would be an evolving (phylogeny) and self-assembling (ontogeny) modular robot with an online optimization algorithm (epigenesis). The ontogenetic axis is not yet included in this project.

The next chapter is a quite comprehensive review of modular robotics. Not all discussed issues are directly relevant for this project. However, the chapter as a whole should give the reader a sense of the particularities of modular robotics, especially with respect to modular robot control.

In the third chapter I discuss bio-inspired locomotion control both from a phenomenological and a neurophysiological point of view. Then I present the nonlinear oscillators that I choose as canonical subsystems to model the central pattern generator. Furthermore I introduce a new notation for the coupling parameters of the nonlinear oscillators. Based on this notation, one can easily predict the phase difference between two coupled oscillators, which has been considered impossible before. Moreover I propose a new coupling term with interesting properties.

Chapter 4 discusses co-evolution of morphology and control. I explain the genetic encoding that I use to co-evolve configuration and control of modular robots. Furthermore I present a powerful genetic encoding that restricts the phenotype space to symmetric configurations.

The online optimization algorithm is outlined in Chapter 5. Finally I discuss the results of the co-evolutionary algorithm and the performance of the different coupling schemes.

Chapter 2

Modular Robotics

2.1 Introduction

A *modular robot*¹ (Figure 1) is built from modules that can be connected together in various ways, much like a child's Lego bricks. The main idea is to use only one or very few module types, but many modules. Current research is done with robots built up from dozens of modules. In the future one can imagine robots with hundreds, potentially thousands of modules.

To live up to their promises (see below), modular robots should have the ability to dynamically change their configuration (also sometimes referred to as structure or topology) by reconnecting the modules in different ways. For example, a modular robot could climb over an obstacle in a quadruped configuration and then reconfigure to a snake in order to slide through a small hole. *Self-reconfiguration* (autonomous or automatic reconfiguration) is one of the main challenges in modular robotics.

Two concepts that are closely related to self-reconfiguration are *self-assembly* and *self-replication*. A cluster of modules may be capable of self-assembling into various types of robots [Kurokawa et al. 2004]. Separate individual modules with some locomotion capabilities could also gather, connect with each other and self-assemble (i.e. self-reconfigure) to a given configuration. Self-assembly opens the door for self-replication. For example one can imagine that a Modular Robot (MR) self-replicates in an environment with enough free modules by releasing a module or by transferring its genome to a free module. This seed could then self-assemble with other free modules to form a new robot. Alternatively the parent can actively assemble a copy of itself [Mytilinaios et al. 2004]. Furthermore, *self-repair* mechanisms can be implemented by ejecting damaged modules and replacing them with spare ones or by self-reconfiguring the robot around them.

¹ Sometimes also called cellular robot or metamorphic robot.

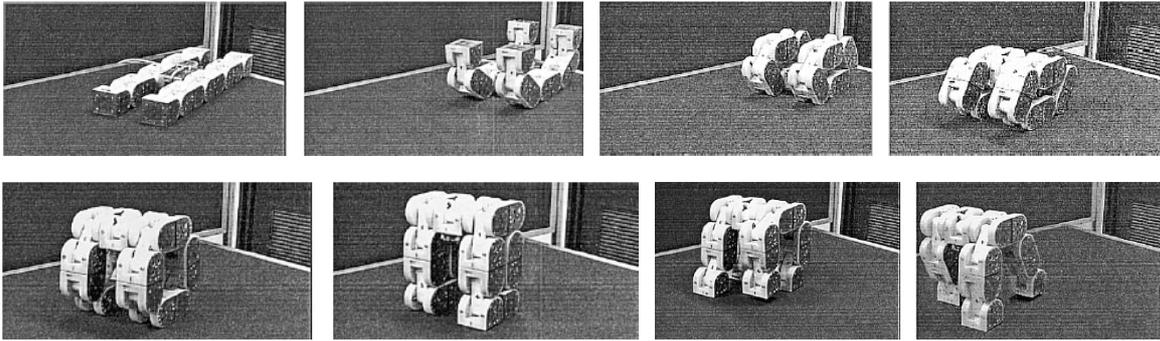


Figure 1: M-TRAN self-reconfiguring first from a crawler to a rolling tank and then to a quadruped configuration.

Collective robotics (also sometimes called *swarm robotics*) is another field that has received a lot of interest lately [e.g. Mondada et al. 2002], swarms of relatively simple robots that can move autonomously and dock to form larger structures cooperate to achieve complex tasks. Intelligence should emerge through interaction, like in swarms of social insects. The similarities with modular robotics, where simple modules form together a complex robot, are apparent. However, there are also some important differences: In swarms, cooperation is generally rather loose. For example, no exact alignment and synchronization are needed to push a heavy object in a team of robots. On the other hand, modular robots demand a very tight cooperation among modules. Consider a walking modular robot where exact synchronization of all modules is required for efficient locomotion.

2.1.1 Classification of modular robots

Generally modular robots are classified along three distinct axes:

- Manual vs. automatic reconfiguration
- Homogeneous vs. heterogeneous modules
- Chain vs. lattice-type

As I mentioned above, self-reconfiguration is an essential feature of modular robots. Unfortunately, since automatic reconfiguration is very complex, the majority of physically realized chain-type robots is currently not self-reconfigurable. Note that a lot of issues in modular robot control can also be explored using manual reconfiguration.

Homogenous modular robots only use one module type while *heterogeneous* modular robots use several distinct module types. The term *semi-homogenous* can be used for robots that consist of exactly two module types.

Modular robots are usually referred to as being either lattice or chain-type. *Lattice-type* modular robots are generally homogenous and use highly symmetrical modules. Conceptually, the modules are organized in a grid. Modules have the ability to move to neighboring positions on the grid by rotating or sliding along adjacent modules. For reconfiguration, individual modules wander on the cluster to their final position. Lattice-type robots use cluster-flow (or water-flow) locomotion: In order to move, the robot

reconfigures in a way to continuously bring the modules from the back to the front giving the impression that the module cluster flows on the ground and around obstacles.

On the other hand, *chain-type*² robots reconfigure by extending limbs that can connect at the desired new position on the robot. For example, a walker can reconfigure to a snake by attaching one leg after the other to the tail. In contrast to lattice-type cluster-flow locomotion, chain-type robots reconfigure only in order to adapt to their environment or function. They locomote in a fix configuration, using powered joints for instance, which is faster and more energy efficient.

Murata et al. coined the term *hybrid-type* for chain-type robots that have the ability to reconfigure in a lattice-type way, i.e. by moving individual modules to neighboring positions on a grid [Murata et al. 2000]. Maybe because M-TRAN (see Chapter 2.2.2) is currently the only hybrid-type modular robot to my knowledge, this term is generally not used in the literature.

We are more interested in chain or hybrid-type than in lattice-type modular robots at the BIRG laboratory because, as I mentioned above, the latter ones have superior locomotion capabilities (faster and more efficient). Our modular robot hardware prototype YaMoR is chain-type (see Chapter 2.2.4). Consequently I focus on this kind of MR system in this thesis and include lattice-type modular robots only for the sake of completeness.

2.1.2 Promises and Applications

Versatility, adaptability, reliability and low cost are the four promises that are consistently advertised in modular robotics literature [e.g. Murata et al. 1994]. I add *modular design*, actually the most obvious feature of modular robots but often not explicitly mentioned, as a fifth promise:

1. *Versatility*. With the number of modules used, the number of possible configurations grows exponentially and the number of degrees of freedom linearly, making modular robots extremely versatile. In other words, the same system could be used for various tasks, which is especially useful when it is undesirable to build a special purpose robot for each task.
2. *Adaptability*. A self-reconfigurable modular robot can autonomously adapt to its environment. Such a robot could for example use an energy efficient rolling-loop configuration on flat ground and reconfigure to a walker when the terrain gets rough.
3. *Reliability* stems from redundancy. Modular robots that are controlled in a distributed manner are robust because failure of some modules potentially only degrades the overall function. Furthermore self-repair mechanisms can be implemented by ejecting damaged modules and replacing them with spare ones or by self-reconfiguring the robot around them. The more modules a robot has, the higher is its self-healing potential. On the other hand it is important to be aware that as the number of used modules grows, the probability of failure of one or several modules also grows and self-repair becomes a crucial issue.

² The terms string, linear or thread type are also found in the literature.

4. *Low cost.* Mass production could lower the price of single modules. Versatility and reliability are qualities of great commercial interest because the same robotic system could be used for diverse and changing tasks with low maintenance costs. However, one has to be careful about this promise because modules are needed in great numbers to form a single robot. For self-repair and distributed control modules must be self-contained (i.e. autonomous with respect to power, processing, communication, sensing, etc). It is not clear if such complex modules can be produced at sufficiently low price to build modular robots that are cheaper than special purpose robots.
5. *Modular design.* In a modular robot, the ratio between the number of module types and the total number of modules used is low. Such a system is called *n-modular*, *n* being the number of module types. “The low heterogeneity of the system is a design leverage point getting more functionality for a given amount of design. The analog in architecture is the building of a cathedral from many simple bricks. In nature, the analog is complex organisms like mammals, which have billions of cells, but only hundreds of cell types.” [Yim et al. 2000]

Naturally, a modular robot that fulfills all these promises would be interesting to use for any task that a traditional robot can do. For example, industrial modular robots working on an assembly line could simply be reconfigured to suit a new process. However, versatility, adaptability and reliability make modular robots especially well suited for harsh and unpredictable environments found in applications like space exploration, urban search and rescue (USAR) or undersea mining, where a complete set of desirable functions cannot be determined a priori.

It is estimated that in the past two decades, disasters like earthquakes, tornados, hurricanes, floods, fires, hazardous materials accidents etc. have caused about 3 million deaths worldwide, 800 million people adversely affected, and property damage exceeding US\$50 billion [Yim et al. 2000]. Tragically, in the context of population growth and global warming, natural disasters are expected to be more frequent and cause even higher damages in the future. The most common USAR scenario is the location, initial medical stabilization and rescue of victims trapped under collapsed buildings. Modular robots may be able to explore confined spaces deep in a rubble pile. The use of robots in USAR applications is especially important because collapsed structures are often unstable and dynamic and the safety of rescuers is at stake. [Yim et al. 2000]

Personally I believe that the first practical application of modular robots will be in space exploration. A MR system can be brought into space at relatively low price because the modules can be packed in a compact way. Once in space, the modular robot can be used for various tasks, for example in space station maintenance operations. Robustness is of crucial importance in space missions because they often have only one attempt to succeed at great cost. Furthermore, the absence of dust/dirt and zero gravity are favorable conditions for modular robots. Dust is problematic for the connection surfaces and gravity constrains the length of module chains [Yim et al. 2003]. Modular robots are also well suited for planetary exploration (consider that when a rover gets stuck or tips over this is a mission-ending disaster). A more futuristic vision of modular robots in space are extra terrestrial self-replicating robot colonies, e.g. for lunar resource development [Chirikjian et al. 2002].

I believe that besides from these practical applications, modular robots are also interesting from an academic point of view. Modular robotics is an ideal test bed for

research in distributed systems, self-organizing systems, multi-agent systems and robot control with many degrees of freedom (DOF). Furthermore, MR systems are a perfect framework to build one day a truly autonomous machine by applying bio-inspired methodologies [Marbach & Ijspeert 2004].

2.2 Module design

The success or failure of any MR system begins with the module design. The space of possible robot configurations and its theoretical locomotion and reconfiguration capabilities are determined by important design choices: The module's shape, number and position of connection surfaces, connection mechanism, joint types, number of DOF, global/local communication and processing capabilities, sensors, etc.

For example an M-TRAN like module can perform lattice-style reconfiguration while this is not possible with modules that have only two perpendicular rotational axes as CONRO does (see below). On the other hand, the latter ones might favor locomotion. Besides from kinematical constraints the module design also has a strong impact on the control algorithms that can be used (e.g. can a module only talk to its immediate neighbors or has it a wireless communication system?).

Given the importance of the module design, I did a comprehensive review of existing modular robot projects. In the next sections I discuss three leading chain-type MR systems in detail: CONRO, Polybot and M-TRAN. These robots are generally acknowledged to be state-of-the-art by the modular robotics community. Furthermore, they are interesting to compare because all three use very different but equally interesting approaches to both module design and modular robot control. Last but not least I present BIRG-module, the MR system that is currently being developed at our laboratory.

Lattice-type modular robots are not included here. They often use triangular and hexagonal shapes for planar robots and cubic or dodecahedral modules for 3-D. Some of the leading projects are: Telecube [Vassilvitskii et al. 2002], Crystalline [Vona & Rus 2000], I-Cubes [Unsal et al. 2000], Fractum [Tomita et al. 1999] or ATRON [Østergaard & Lund 2003].

2.2.1 CONRO

CONRO is a chain-type homogenous modular robot developed by the USC Information Science Institute and Computer Science Department. A CONRO module (Figure 2) has two degrees of freedom (yaw and pitch). It has one female connector on one side and three male connectors on the other side. The male connectors have two pins and can be attached to a female connector in any of two orientations: "Belly-up" or "belly-down". However, the IR transmitters/receivers that are used for communication are not correctly aligned if one module is "belly-up" and the other "belly-down" [Castano & Will 2001]. Therefore only one orientation can practically be used.

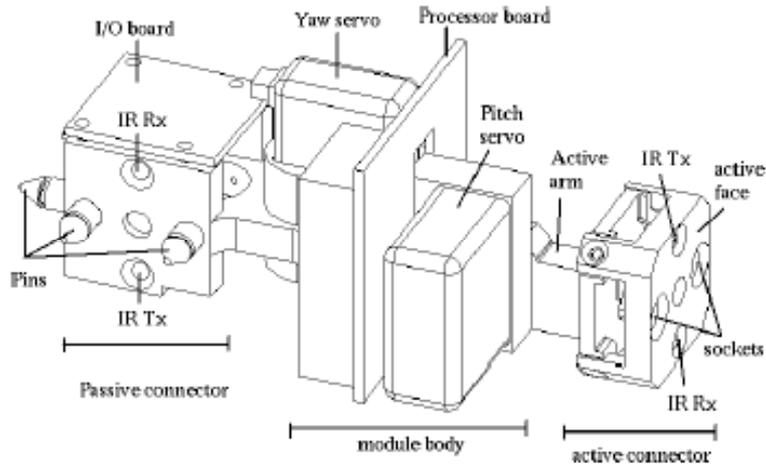


Figure 2: Schematic view of the CONRO module.

Compared to other modular robotics projects, the CONRO module is with two perpendicular rotational axes quite flexible. Even a single module has (limited) locomotion ability, which theoretically opens the door for self-assembly. CONRO has demonstrated some self-reconfiguration capability but the use of female/male connectors that can be attached together only in one orientation makes it very difficult. This lack of flexibility in docking might be one of the reasons why Shen et al. use manual reconfiguration in most experiments [e.g. Shen et al. 2002] and have not worked on a planning method for non-trivial reconfigurations for CONRO.

2.2.2 M-TRAN

The M-TRAN [Murata et al. 2002], standing for Modular TRANSformer, is developed by the Distributed System Design Research Group of the National Institute of Advanced Industrial Science and Technology of Japan (AIST).

The M-TRAN module consists of two semi-cylindrical boxes, connected together by a link as shown in Figure 3. The module has two degrees of freedom: Each box can rotate 180° independently. The servomotors are actually embedded in the connection link. The two rotational axes are parallel, in contrast to CONRO where they are perpendicular (yaw and pitch). This design allows M-TRAN to perform (limited) lattice-style reconfiguration.

The original approach of the M-TRAN module lies in its connection mechanism that uses permanent magnets. The module has an active and a passive box with three connection surfaces each. All connection surfaces of passive boxes use magnets of the same polarity, while the active boxes use the opposite polarity. Active boxes always meet passive ones when multiples of 90° are used for joint angles (check board parity). The active connection surfaces use shape memory alloys (SMA) and non-linear springs in their detachment mechanism, the passive ones simply consist of a plate with magnets. In addition, electrodes are placed on each connection surface for power supply and communication with neighboring modules or the host computer.

The use of magnets greatly simplifies autonomous docking because no exact alignment is needed as long as the modules are sufficiently close for the magnetic force to act. This is

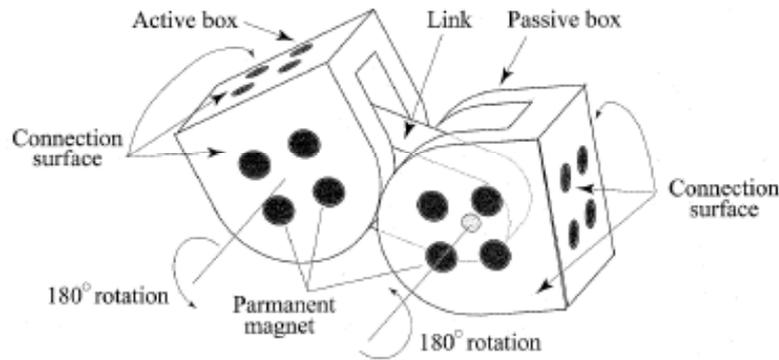


Figure 3: Schematic view of the M-TRAN module.

probably the main reason why M-TRAN has demonstrated superior self-reconfiguration capabilities than other advanced modular robotics projects such as Polybot or CONRO.

Recently a second prototype (M-TRAN II) has been built [Kurokawa et al. 2003]. The module now has an acceleration sensor and various improvements have been made (information processing ability, power consumption, connection mechanism, size).

2.2.3 PolyBot

The Palo Alto Research Center (PARC), a subsidiary of Xerox Corporation, develops PolyBot [Yim et al. 2000], the successor of the famous Polypod [Yim 1994]. The design philosophy behind PolyBot is to use very simple and small modules that should fit in a cube 5cm on a side, thus PolyBot uses cubic modules with only one DOF (see Figure 4). These segments have only two connection surfaces: One on the front and one on the back. Hence the need for a second module type (node) which is simply a cube with a connection surface on each side. Using segments and nodes, semi-homogenous modular robots can be built.

PolyBot G2 (second generation) is self-reconfigurable. The connection mechanism is hermaphroditic and 4 times redundant: Connection plates can mate in 90 degrees increment. Each connection surface has 4 LED's and 2 photo-diodes for 6 DOF position and orientation determination in autonomous docking. Processing capabilities of individual modules are strong thanks to an embedded Motorola PowerPC 555 processor with 1MB of RAM. Unfortunately, external power supply is used and sensing is limited to joint position in G2 [Duff et al. 2001].

The third generation of PolyBot is currently under construction. It mainly adds a brake/ratchet to the main actuation, is much more compact and has improved sensorial capabilities. It is planned to equip G3 modules with a wide range of sensors: 4 IR emitter-receiver pairs and 4 force sensors per connection surface, a potentiometer, joint angle sensor and two accelerometers [Zhang et al. 2002a].

PolyBot G2 was the first robot to demonstrate sequentially two topologically distinct locomotion modes by self-reconfiguration. With the G3 module, Yim et al. want to build a modular robot using an order of magnitude more modules (100+) than previous robots.

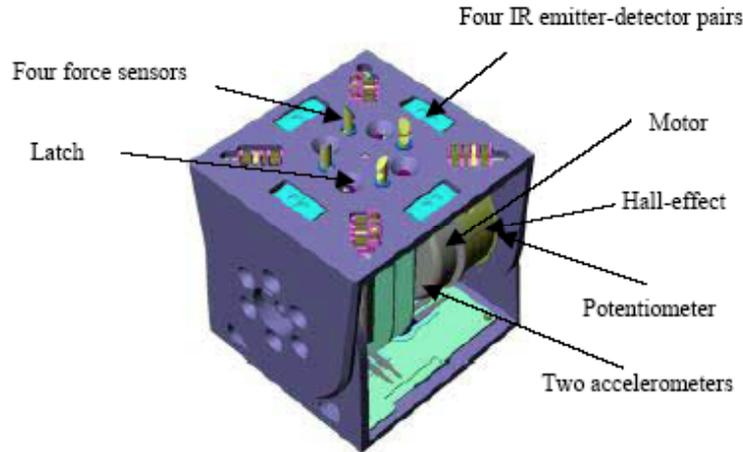


Figure 4: The Polybot G3 module with its sensors.

Note that it is possible to construct PolyBot meta-modules that are conceptually equivalent to M-TRAN modules by connecting two segments together in the same orientation (parallel axes of rotation) and adding a node on each side. Meta-modules that are similar to Conro can also be built: The segments are attached at 90 degrees shift (perpendicular axes of rotation) and only one node is added on the front side. Last but not least, PolyBot can also mimic YaMoR (see next section) with meta-modules that consist simply of a node attached to a segment.

I find the semi-homogenous design with simple, cubic units very appealing. Meta-modules that are conceptually equivalent to modules of other MR systems can be built with PolyBot. This fact indicates that PolyBot *may* be functionally superior to these systems (i.e. more versatile and adaptive). However, this flexibility comes at the price of heterogeneity that might in turn complicate control, especially with respect to self-reconfiguration. Only the future can show if homogenous systems using more complex modules or heterogeneous systems with simpler modules are better suited for particular applications in modular robotics.

2.2.4 YaMoR

YaMoR (Figure 5) is the MR hardware prototype that is currently being developed at our laboratory, the Biologically Inspired Robotics Group (BIRG) of the Swiss Federal Institute of Technology in Lausanne. YaMoR stands short for Yet another Modular Robot – a very humble name. Indeed, the short-term objective was to rapidly build a prototype at low cost. Even though YaMoR is technologically not as advanced as the modular robots introduced in the previous sections, it is perfectly well suited to investigate issues in locomotion control. For details, refer to [Dittrich 2004].

The design is inspired by modules such as M-TRAN or Polybot (see above). YaMoR has a single degree of freedom: The hinge of the U-shaped lever has a working range of a little bit more than 180°. It is driven by an RC-servo with a maximum torque of 73 Ncm, which is sufficient for a module to lift three others. The module weighs about 250 grams and it is 94mm long (including the lever) with a cross section of 45mm x 50mm.

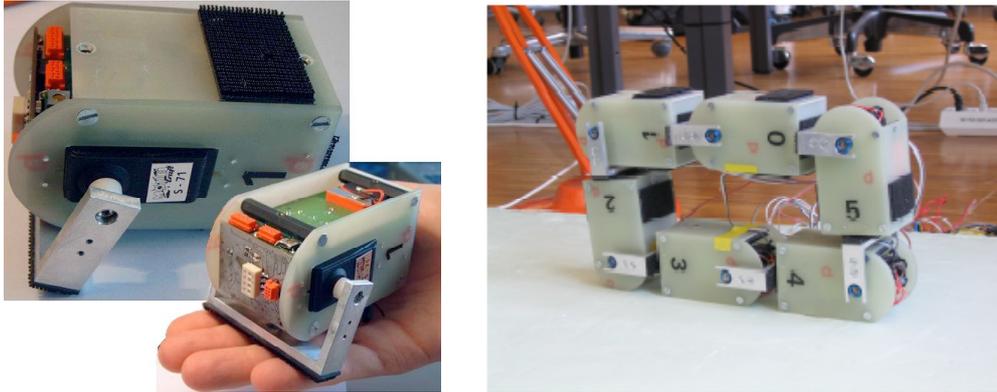


Figure 5: The YaMoR module. On the right, YaMoR in a rolling loop configuration.

The modules are self-contained, i.e. autonomous with respect to power, actuators, sensors and computation. Each module is equipped with a Field Programmable Gate Array (FPGA), a more flexible computational resource than common micro-controllers used in modular robots. No automatic docking mechanism has yet been designed. A special type of genderless Velcro® fastener is used to attach the modules together. Velcro proved to be a very simple, flexible and efficient way to manually connect modules together.

Comparing YaMoR with other modular robots, its most distinctive feature is the wireless communication via BlueTooth, which is not common in modular robotics. The advantage of wireless communication is, that there is no need for an electrical connection between the modules. This simplifies the design of the docking plates and avoids problems with dust/dirt that are likely to occur outdoors. Furthermore disjoint modules or groups of modules can communicate, for instance to self-assemble to a modular robot.

2.3 Modular Robot Control

Modular robot control is an extremely challenging field because modular robots generally have many more DOF and sensors (provided that the modules have sensorial capabilities) than ‘normal’ robots. Reconfiguration and locomotion are by themselves complex problems. The combination of both and the distributed nature of a modular robot further complicate things.

Consider for example PolyBot G3 (see chapter 2.2.3). It is planned to build robots with more than hundred modules. Such a robot would have about thousand force sensors and IR receivers/emitters (four per connection plate) and hundred potentiometers and joint angle sensors. The controller of the roughly 100 DOF has to be implemented on a distributed network with dynamic topology (due to reconfiguration) of about 100 embedded processors.

A modular robot is a distributed system and as such it can be controlled using a *centralized master-slave approach*, a *distributed approach* or a *combination of both*. In the centralized approach, at every time step the *central controller* sends action commands to all modules, usually identified by their id. The controller can be a host computer or one of

the modules. The centralized approach is problematic for several reasons: 1) It is not scalable because the central controller creates a communication bottleneck. 2) Master-slave control is not robust because if the controller fails, the whole system shuts down. 3) It is not well suited to deal with dynamical reconfiguration because it is impossible to know *a priori* where a specific module is in the current configuration [Salemi & Shen 2004]. In other words, the same module is not always on the same position in a specific configuration.

The problems mentioned above can be addressed by partially decentralizing the system: To resolve the scalability problem, the central controller can leave low-level control to the modules and take the role of a ‘coordinator’ [Yim 1994]. For example, the controller could be used to synchronize modules that know their proper action sequence. Such a partially decentralized system would also be better suited to implement dynamical reconfiguration because modules can locally choose their appropriate behavior based on their position in the configuration.

However, the system still has a single point of failure. Even though robustness could potentially be achieved if the ‘coordinator’ is a module and if other modules can take its place in case of failure, a completely decentralized approach that embraces the distributed nature of the MR system seems clearly more appealing.

One can distinguish between synchronous and asynchronous distributed control. In *synchronous distributed control*, the role of the ‘coordinator’ is taken by a distributed synchronization algorithm. Shen et al. take this approach in their hormone-based control algorithms (see Chapter 2.3.3). However insisting on strict global synchronization has a high cost in terms of efficiency [Støy et al. 2002].

Asynchronous distributed control emphasizes the autonomy of the modules. They are considered autonomous agents and the modular robot can be viewed as a *multi agent system*. The global behavior should emerge from local interaction between the modules [Støy et al. 2002, Murata et al. 1994]. Role-based control (see Chapter 2.3.2) and non-linear oscillators (see chapter 4.4) are examples of asynchronous distributed control.

In the following sections I review various approaches to modular robot control. Even though this is not a complete overview, the selected topics are a good introduction to the main issues in modular robot control. It is also important to be aware that not all topics address the same issues. The first three sections concern locomotion control. Then I present hormone-based and constraint-based control, which are both *control frameworks*, i.e. various types of controllers (e.g. locomotion, navigation, reconfiguration, etc) could be implemented within these frameworks. Since locomotion control with oscillators is the main focus of this work, it is not included here and presented more in detail in chapter 4.

Even though some of the presented concepts also apply to lattice-type robots, I focus on chain-type modular robots. Lattice-type robot control usually takes inspiration from cellular automata [e.g. Hosokawa et al. 1998, Butler et al. 2002]. ‘Virtual scents’ or other kinds of gradients are also used a lot [Bojinov et al. 2000, Støy 2004].

2.3.1 Gait control tables and phase automata

The most common and obvious approach to control locomotion of a chain-type robot in a specific configuration is a centralized *gait³ control table* [Yim 1994]. Each column of the table contains the sequence of actions for a specific module, usually identified with an id. Actions are generally just motor commands that set the desired angle. In mathematical terms gait control tables correspond to simplified finite state machines with a predefined sequence of actions for each module.

The gait control table is generally precomputed by a host that sends the commands of the next row to the modules at every time step. Alternatively, the host can also download the tables to the modules, which act as slaves and execute their table once received.

Besides from the problems inherent to the master-slave approach, namely scalability and reliability, gait control tables are a simple open loop behavior without sensory feedback. *Phase automata*, a generalization of gait tables, address this issue. A phase automaton is an event-driven discrete state machine with periodic behavior. The phase represents the automata's starting point in a continuous time domain. Events can be triggered by time or by sensors. Modules have a two-level control. The phase automata sets the control attributes (e.g. joint angle) and the lower level controller tracks the desired behavior [Yim 2003].

2.3.2 Role-based control

Role-based control has been proposed by Støy et al. for locomotion of chain-type robots [Støy et al. 2003]. It is an asynchronous distributed approach. Each module plays a role. A role consists of a periodic function $A(t)$ (e.g. a harmonic oscillator) that specifies the joint angle(s) of the module. Additionally, a delay for every child connector is given. The modules periodically send signals with the corresponding delay to their children and act as the master of their sub-trees. Therefore, synchronization is done locally with neighbors and global synchronization emerges over time. Even though the authors program the roles manually, one could also use an optimization algorithm.

Role-based control is an elegant approach to synchronize the system in a distributed manner but the algorithm has a major drawback: It is assumed that modules have one and only one parent connector. Modules can only be connected with this parent connector to child connectors of other modules. Conro modules can't be physically connected in a way that violates this constraint (the female connector being the parent) but if a genderless connection mechanism is used or if several male and female ports are available (e.g. M-TRAN) the algorithm can't be applied as is.

Another disadvantage is, that the synchronization procedure in role based control can lead to discrete jumps in the generated trajectories $A(t)$. Consider two connected modules that are not synchronized. When the parent sends the synchronization signal, the child will reset its clock t , leading to a brutal and uncontrolled change in the trajectory. In contrast, nonlinear oscillators always generate smooth trajectories (see Chapter 4).

³ A gait is a cycle of a pattern of motion that is used for locomotion

2.3.3 Hormone-based control

Shen et al. took inspiration from hormones to design reliable, distributed control algorithms that can deal with dynamic configuration changes. A digital hormone is, as its biological counterpart, a message that propagates in the network and triggers different actions from different receivers. In contrast to message broadcasting, a hormone may have a lifetime and can be modified or deleted by cells as it travels through the network [Shen et al. 2002].

As I mentioned above, a modular robot can be seen as a distributed system consisting of a set of interacting autonomous agents (the modules). It can be represented as a dynamic network where nodes are modules and links represent physical connections or communication channels between modules. The network is dynamic because its topology might change through reconfiguration. Modules must have some essential capabilities in order to perform complex tasks within this context [Salemi & Shen 2004]:

- *Distributed task negotiation*: The modules must agree on a global task to perform.
- *Distributed behavior collaboration*: Based on the global task each module has to choose its appropriate local behavior.
- *Synchronization*: To compensate the lack of global clocks, the local behaviors need to be synchronized between modules.
- *Topology monitor and discovery*: Modules must discover the current configuration and their position for correct action selection. Potentially topology monitoring might allow modules to detect damages and adapt their behavior accordingly (evtl. trigger self-reconfiguration).

A priori a module cannot know the current configuration of the robot. Topology discovery consists in representing the current configuration and potentially trying to map it to a known configuration in a database (i.e. one for which a controller is available). The problem is to find a good representation for configurations. A simple graph with a node for each module and links for physical connections is not sufficient because two modules can be attached together at various connection ports and/or with different orientations. To include information about the ports and the orientations that are used, Castano et al. use directed graphs with several vertexes for each module [Castano & Will 2001]. Configurations can then be compared by their adjacency matrixes.

Salemi et al. have proposed a distributed task negotiation algorithm for modular robots called DISTINCT [Salemi et al. 2003]. A module can originate a new task by propagating a specific hormone to build a Task Spanning Tree (TST). A TST is a growing tree of modules within the configuration that have all agreed on the same task. If different modules initiate multiple tasks a forest of partial TSTs might grow in the network. The partial TSTs gradually merge until there is only one TST covering all modules, the originator of the winning task being its root.

For a given task in a specific configuration, a module can generally select the correct behavior based on its local connection information. Salemi et al. have defined 32 types, covering all possible configurations of a CONRO module with its *immediate* neighbors [Salemi et al. 2001]. For example, if the module is attached to the ‘left’ port of another module with its ‘back’ port, its type is T5. If the position of the module within the configuration is not uniquely determined by the immediate neighborhood (e.g. in a snake

robot), an *extended neighborhood type* that includes connection information until distance n can be used [Salemi 2004]. A module can discover and monitor its type dynamically and autonomously based on the hormones it receives.

The correct local behavior is selected as a function of the extended type (i.e. the local configuration) and the selected behaviors of neighboring modules. Potentially, elapsed time or sensor input may also be considered. Algorithms for local behavior selection can be found in [Salemi & Shen 2004] and [Støy et al. 2003]. The latter one doesn't use the terminology of hormones but the approach is the same.

Last but not least, synchronization can also be achieved with hormone-based control. For example, a module can delay the propagation of a hormone or wait for special synchronization hormones from its active neighbors. Both serial and parallel hormone-based synchronization are discussed in [Salemi et al. 2001].

I agree with Støy et al. [Støy et al. 2003] that an important issue has not been addressed in the hormone-based protocols for Conro: A hormone might be lost on its way due to system failure or dynamic reconfiguration. If the sender waits for the message to propagate through the configuration tree and back this leads to a system stop with the algorithms proposed by Shen et al.

The algorithms discussed above address important issues in distributed control of modular robots. They are scalable, robust to dynamic reconfiguration changes and don't rely on global unique identifiers but I find the name hormone-based control somehow misleading. It suggests a novel, bio-inspired approach for communication in a distributed system but in practice, hormones are just content-based messages that are exchanged between modules using adequate protocols. It is true that these messages may have a lifetime that nodes can delete them, alter their content or insert a delay before routing them to the neighboring nodes, but this is true for other protocols as well and has not much to do with the functioning of biological hormones. Shen et al. have recently proposed a Digital Hormone Model (DHM) that is truly bio-inspired [Shen et al. 2004]. For example, stochastic reaction, dissipation and diffusion are modeled. Unfortunately the authors don't discuss how this model could practically be applied for modular robot control.

2.3.4 Constraint-based control

As I discussed above, in strictly asynchronous distributed control the robot is viewed as a multi-agent system. Modules are autonomous agents that exchange messages with other (generally neighboring) modules. It is important to keep in mind that this is only a model and that alternative views of a MR system are possible.

Zhang et al. have developed a constraint-based control framework to effectively program modular robots [Zhang et al. 2002b, Fromherz et al. 2001]. Constraint-based control is only a framework, thus various types of controllers could be implemented on top of it. Within this framework, the modular robot is not modeled as an aggregation of autonomous agents but as a large embedded distributed system with dynamic topology. In other words, the system is not viewed as N controllers that run on separate modules but as a single control system that is implemented on a network of N embedded processors.

In constraint-based control, the control problems are casted as constrained optimization problems. Thus, the controller consists of a set of constraint solving components

distributed over the embedded network. “Constraint solvers are goal oriented deliberative agents; they can be used as control regulators that drive the system to desired states or as information retrievers that extract current state information from sensed data.” [Zhang et al. 2002b]

The framework is built on an Attribute/Service Model (ASM). Constraint solvers are considered as services that can be triggered by time or by externally (hardware) or internally (software) generated events. Various components (services and shared attributes) of the control system may be physically located on the same or on different processors. ASM makes this transparent.

Zhang et al. have successfully applied constraint-based control to modular self-reconfigurable robots in simulation [Zhang et al. 2002b]. It is planned to use the framework with PolyBot G3 (see chapter 2.2.3). Unfortunately, the authors only focus on scalability and not on reliability. They mention that a state retriever can act as a diagnostic engine to detect failures, but it is not discussed how the system reacts when a whole node (module) fails.

2.3.5 Reconfiguration

A general reconfiguration planning algorithm would find the fastest reconfiguration sequence between two arbitrary configurations. Unfortunately it is computationally intractable to find the optimal solution [Chirikjian et al. 1996]. Various heuristic-based methods have been proposed for lattice-type robots: For instance, modules can be attracted to their final position with gradients as described in [Støy 2004] or [Bojinov et al. 2000]. The total distance that the modules traveled on the grid can be used as cost function for the reconfiguration sequence.

The main difficulty in chain-type reconfiguration planning is the numerous constraints that must be respected. For instance, one must take into account collisions with the ground, self-collisions, gravitational stability, actuation torque limits etc. [Murata et al. 2004]. Reconfiguration is much simpler for lattice-type modular robots since they limit the possible angles between elements. Furthermore, an elementary step in the reconfiguration sequence consists of a single module moving to a neighboring position on the grid. In contrast, chain-type modular robots must coordinate many modules for exact alignment and autonomous docking.

Initially I planed to design a new hybrid-type module with two DOF, inspired by the M-TRAN but using regular or rhombic dodecahedrons instead of cubes as basic shape for the modules. By doing so I hoped to overcome the limitations of M-TRAN with respect to lattice-style reconfiguration (see below) but finally the project took another direction. Nevertheless I discuss hybrid-type reconfiguration in the next section, as I believe it to be a very promising and sometimes over-looked approach.

2.3.5.1 Hybrid-type reconfiguration

Hybrid-type reconfiguration is an attempt to lower the complexity of reconfiguring chain-type modular robots. As mentioned above, various algorithms for reconfiguration planning with lattice-type robots have been proposed and alignment is easier because modules only

move to neighboring positions on the grid. Therefore it is desirable to reconfigure a chain-type robot in a lattice-type way, i.e. instead of reconfiguring whole chains of modules, individual modules wander on the robot body to their new position.

As I mentioned before, M-TRAN (generally considered chain-type) is currently the only modular robot that classifies as hybrid-type. However, the lattice-type reconfiguration capabilities of M-TRAN are limited because a module can only move to some of the neighboring positions on the grid. On a flat cluster, a module can travel in two modes: Forward roll and pivot mode (see Figure 6). A second module is needed for transition between these two modes. These limitations make it impossible to use general lattice-type reconfiguration algorithms for M-TRAN.

With the help of a graphical user interface for manual reconfiguration planning, Murata et al. have demonstrated a wide variety of reconfiguration experiments, both in simulation and with hardware. For example a crawler can self-assemble from a regular cluster of modules, then reconfigure first to a rolling tank and finally to a four-legged walker as illustrated in Figure 1.

Naturally, manual reconfiguration planning by trial and error has its limitations and is generally not acceptable for autonomous modular robots. Murata et al. have tackled the problem of finding a reconfiguration planning algorithm by introducing regularity in the module cluster. Regular clusters consist of four-module blocks, also called meta-modules. Three different types of such clusters have been designed and a centralized planning method for reconfiguration has been developed [Yoshida 2002] in order to achieve lattice-type locomotion (cluster flow) with M-TRAN modules. Locomotion is achieved by succeedingly moving tail meta-modules to the front. A global planner determines the desired position of the moving tail block. On a lower level, a motion scheme selector is responsible for planning the movements of individual modules based on a rule database. Clusters can move in various directions and climb over obstacles.

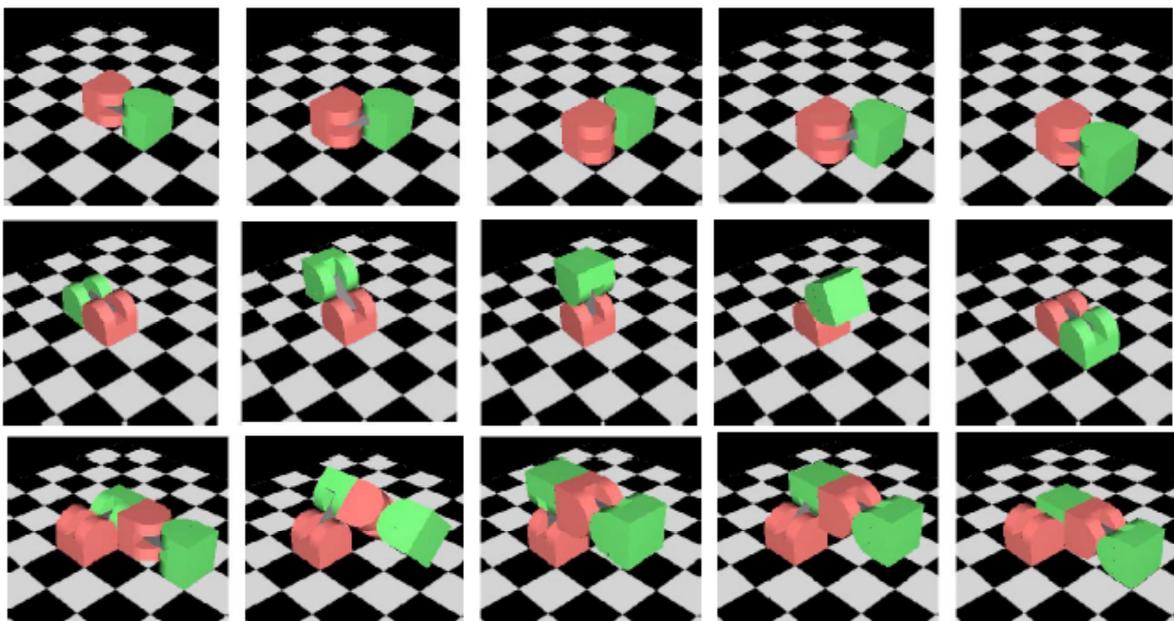


Figure 6: The M-TRAN module can move on a cluster of modules (here represented as the floor) almost in the same way as lattice-type modules do. If the module is lying on its side, it is in pivot mode (on the top), otherwise in forward-roll mode (in the middle). For conversion between these two modes, a second module is required (on the bottom).

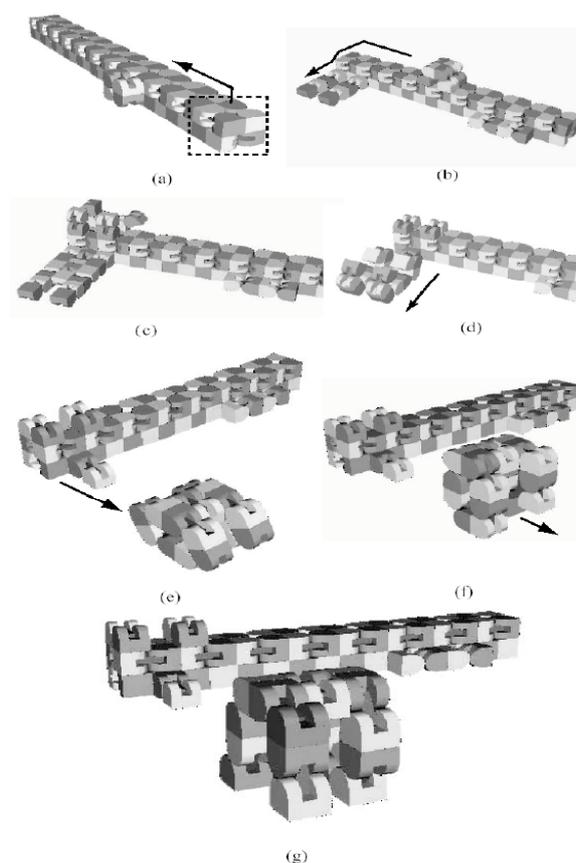


Figure 7: Hybrid-type reconfiguration capabilities of M-TRAN. A regular cluster of modules can locomote in a lattice-style way and/or assemble various types of walkers.

Kurokawa et al. have described an operation scenario for search and rescue where regular clusters locomote and/or self-assemble various types of walkers [Kurokawa et al. 2004]. These walkers could potentially reassemble into a cluster (refer to figure 7).

Even though this approach permitted astonishing demonstrations with M-TRAN, it relies heavily on manually programmed solutions. The proposed reconfiguration planner only concerns lattice-type locomotion of regular clusters. It is a centralized algorithm that uses a manually programmed rule database. More complex reconfiguration sequences, for example the walker generation from a cluster, have been completely hand programmed. As I discussed above, centralized control is problematic for modular robots because of scalability issues and robustness.

2.4 Sensors in modular robots

In order to display more complex behaviors than open-loop locomotion and to adapt to environmental changes, the robot obviously has to be able to sense its environment and internal state. Especially for their highly advertised locomotion capabilities in unstructured terrain, modular robots need strong sensorial capabilities.

It is striking how little attention has been paid to the use of sensors in modular robots. Previous research has consistently focused on 'blind' open-loop locomotion and reconfiguration. Consequently state-of-the-art MR systems generally only use joint position sensors and, in the case of self-reconfigurable robots, sensors for docking-aid.

Recently an effort has been done to improve this situation: For instance, M-TRAN II now includes an acceleration sensor and PolyBot G3 will be equipped with two acceleration sensors, a potentiometer and force sensors on the connection plates (besides from the IR sensors used for docking). However, these sensors will mainly improve autonomous docking and 'blind' locomotion. I conclude that not a single MR system currently uses modules with sensors that are suited for navigation in rough terrain or obstacle avoidance (e.g. IR distance sensors or photoreceptors).

Consequently, little research has been done on how to include sensor information in modular robot control. The use of sensors in modular robots differs from previous work in sensor systems and sensor fusion in that the position of a specific sensor is not fix (due to dynamic reconfiguration). The question is how to extract meaningful data from a dynamic network of sensors. Clearly one should look for answers in the field of distributed sensor networks where a lot of research has been done recently. See for instance [Esterin et al. 1999].

If the sensor information is used exclusively locally, this is not a problem. In a well-designed controller, every module should know its position in the current configuration and select the correct local behavior. The local behavior may be influenced by the local sensor input. Naturally, the local sensor information can also be used by the module to influence the global behavior, for example by releasing a specific hormone.

So far so good, but generally several sensors from different modules might be needed for correct global behavior selection. One solution is to send all sensor data to a master controller, e.g. a specific module. Such a controller would take global decisions based on the sensor input it gets from the modules and act as the 'brain' of the robot.

Støy et al. have tested a completely distributed approach with CONRO. Sensor information is first abstracted based on the position of the sensor in the robot and then propagated to all modules [Støy et al. 2002]. Each module independently selects the local behavior based on the propagated sensor values. The disadvantage of this solution with respect to the centralized 'brain' approach is slow reaction time and much communication. Obviously it is more efficient to directly send sensor information to a single controller than gradually propagating it to all modules.

Chapter 3

Bio-Inspired Locomotion Control

Clearly, locomotion is an essential skill of any autonomous robot. However, locomotion control is of particular importance to modular robots because their potential applications advertised in the literature always emphasize superior locomotion capabilities on rough terrain.

Unfortunately, many degree of freedom locomotion control is very complex and it turned out to be difficult to design such a controller with traditional engineering methodologies. Since nature was confronted with the same problem, it is interesting to study the solutions that emerged from evolution. Thus, in the first sub-chapter I give a brief overview of vertebrate locomotion.

3.1 Vertebrate locomotion

Efficient locomotion is mandatory for a species to survive. For example, animals rely on locomotion to find food, mate, populate new territories, catch prey or escape from predators. The locomotion strategy of a particular species must be well adapted to its environment and ecological niche. Evolution has led to a wide variety of different types of locomotion such as walking, hopping, crawling, flying, swimming etc, providing thereby a rich source of inspiration for engineers.

Despite the large variety of locomotion strategies that can be observed in nature, we can identify some common characteristics that underlie locomotion of vertebrates. In the subsequent sections I shall discuss these fundamental characteristics of vertebrate locomotion and their impact on bio-inspired locomotion control of robots in detail.

In Sections 3.1.1 – 3.1.4 animal locomotion is analyzed qualitatively, i.e. from a high level point of view. As you will see, locomotion control is extremely complex. Specifically, many oscillatory signals need to be coordinated and synchronized. In Sections 3.1.5 and 3.1.6 I discuss how these signals are generated at a neurophysiologic level. I will focus on the building blocks and architecture of the locomotor circuit, as these findings are most interesting for designing a bio-inspired robot controller.

3.1.1 Locomotion through rhythmic activities

In nature, locomotion is generally achieved with periodic activities. Rhythmic contractions of muscles lead to undulations of the body and/or oscillations of legs, wings or fins. On the other hand, man-made machines generally rely on continuous movement of wheels or propellers. One must be aware that the preference of periodic activity in nature doesn't imply that this approach is always superior to continuous activity for locomotion. However, in the case of locomotion on unstructured terrain the strategies adopted by nature seem clearly superior to what can be achieved with wheeled robots.

The drawback of locomotion based on oscillations is the complexity of the control system. For example, a differential wheel robot can be controlled using only two actuators: The speed of the left and the right wheel. Furthermore, the effects of the actuators on the movement of the robot are easy to understand. On the other side, locomotion through periodic movements generally implies many more degrees of freedom. The signals that induce the oscillations must be well coordinated in order to achieve efficient locomotion and the effect of a single actuator on the global behavior is far from trivial.

Locomotory control is further complicated in the case of dynamically stable gaits of walkers. In a dynamically stable gait, the center of gravity is not maintained at all time within the polygon formed by the contact points of the limbs with the ground (in contrast to statically stable gaits, where this is the case). Therefore the limbs are not only used for locomotion per se, but also to avoid falling over by balancing the momentum, gravitational and inertial forces.

3.1.2 The importance of mechanical dynamics

Clearly, the dynamics of the biomechanical system are as important to efficient locomotion as the dynamics of the neural control. First of all, the body determines what movements are physically possible. Furthermore, the interplay between the control signals, the biomechanical system and its environment leads to the emergence of very complex, nonlinear dynamics. For example, the effect of a particular command depends on the position, momentums and inertial forces acting on the body at that specific time. On the other hand, each command has a long-term effect on the mechanical dynamics and on the future sensory feedback.

For the biologist this implies that the neural system can't be understood without taking into account it's interaction with the body. For the engineer it implies that an effective locomotion controller cannot be designed separately from the body. Over the last decade, embodied cognition has received a lot of interest. Embodiment stresses the importance of the dynamics that emerge from the interplay between the controller, the body and its environment. The importance and complexity of this interplay are strong arguments in favor of co-evolution of morphology and control.

3.1.3 Symmetry – a fundamental characteristic of locomotion

I can't think of a vertebrate whose axis of symmetry doesn't coincide with its preferred direction of locomotion⁴. In order to move on a straight line, the sum of the rhythmical forces that are applied to the environment by the limbs on the left and on the right side must be symmetric with respect to the direction of locomotion, otherwise the animal would turn in a circle. Even though symmetric forces could theoretically also be produced by an asymmetric structure, symmetry obviously favors efficient locomotion.

Symmetry in nature is not limited to the morphology; it also extends to the architecture of the controller. Clearly, the locomotory control circuit of your left leg has the same architecture as the one of your right leg. It wouldn't be advantageous for evolution to come up with two different control systems to achieve the same basic functionalities.

As I will discuss in Chapter 4.3.1, my experiments with a genotype-to-phenotype mapping that implies symmetric morphology and control structures indicate that symmetry is indeed a fundamental characteristic of efficient locomotion.

3.1.4 Locomotion gaits

A gait is a cycle of a pattern of motion that is used for locomotion. Observing animal locomotion at a high level, a gait can be described by the phase differences of the limbs and/or body segments.

For example, typical quadruped walking gaits are walk, trot and bound. The walking gait is characterized by different phases of all legs, i.e. the four legs touch the ground at different times. Trotting is when the diagonally opposed legs touch the ground at the same time and in bound, the hind and the forelegs hit the ground simultaneously.

Generally animals have the capability to switch between various gaits. Transition between gaits is fast (often within one cycle) and smooth. For example, the salamander swims like a lamprey with a traveling wave along the body. On ground, it changes to a trotting gait with a standing wave along the body [Ijspeert & Cabelguen 2003].

Another important observation from animal locomotion is that the frequencies of the rhythmic movements are equal. Also at a lower level of description, there seems to be no evidence that the various oscillators along the spinal cord (see next section) use different frequencies [Ijspeert 1998].

⁴ There are also animals that move sideways, like crabs or some snakes.

3.1.5 Neurophysiology of vertebrate locomotion

An important question to neurobiologists was where the oscillatory patterns are generated. In a famous experiment Shik et al. observed gait transitions in a decerebrated cat [Shik et al. 1966]. A walking gait could be induced by an electrical stimulus of the brainstem. If the amplitude of the stimulus was increased, the gait switched to trot and finally to gallop. This demonstrates that most of the vertebrate locomotory circuit is located in the brainstem and the spinal cord and that it can be controlled with simple stimuli from the higher neural centers. These high-level commands can for example be used to control the speed of motion.

Neurobiologists have extensively studied the locomotor circuit of some species. Central pattern generators (CPGs) are circuits that generate oscillatory output from tonic input. Nowadays we know that CPGs in the spinal cord and in other oscillatory centers are responsible for transforming simple stimuli from higher neural centers into complex patterns of rhythmic signals needed for locomotion⁵.

The locomotor CPG is controlled via multiple descending pathways from the motor cortex. These tracts can either be direct or relayed by centers in the brainstem. For example, the corticospinal tracts used for visuomotor coordination by mammals directly link spinal neurons while the reticulospinal tract, which is essential for locomotion of all vertebrates, is relayed by the reticular nuclei in the brainstem. In general the descending pathways are surprisingly conservative. Recent studies indicate that modifications of the CPGs and of the morphology (hence also of the sensory feedback), might be much more frequent than changes in the descending tracts in the evolution of species [Donkelaar, 2001].

The oscillatory centers (or sub-CPGs) are coupled together to control the phase difference of their periodic output. For example, the swimming CPG of a lamprey consists of roughly 100 segmental oscillators along the body. The swimming gait is a traveling wave from head to tail. Oscillatory neural activity can be measured from an isolated spinal cord in an excitatory bath. The phase difference between two segments remains constant even if increasing the pharmacological stimulation changes the frequency of the oscillations. These oscillations can also be observed from only two isolated lateral segments, hence the oscillatory centers seem to be interconnected with their neighbors. In tetrapods there is evidence of a hierarchical decomposition of the CPG into different oscillatory centers on the level of the limbs, the joints and even the individual extensor and flexor muscles.

In summary the most important properties of CPGs are: a) Rhythmic neural activity without external stimuli or induced by very simple input signals; b) Capability of generating distinct patterns in function of the input; c) Hierarchical decomposition into coupled oscillators.

⁵ Note that CPGs are not only used for locomotion but also for other rhythmic activities like breathing.

3.1.6 The role of sensory feedback

In the previous chapter I have mentioned experiments where rhythmic neural activity has been observed from completely isolated spinal cords, i.e. in the absence of sensory feedback. Other studies have shown that animals whose oscillatory centers had been surgically or pharmacologically isolated from the sensory information were still able to produce similar patterns of neural activity as recorded from intact locomotion.

These findings rejected the theory of peripheral control, which supposed that locomotion is a chain of reflexes based on sensory feedback. It seems that the CPG is able to generate the basic patterns of neural activity without sensory information, but that it strongly relies on sensory receptors in joints and muscles in order to shape these patterns for efficient locomotion. Coordination of mechanical and neural activity is of particular importance within this context. This can be demonstrated very nice by rhythmically moving a limb of a decerebrated vertebrate which is often enough to start the oscillatory activity of the CPG.

Most interestingly, the tracts of the reflexes often share many interneurons with the oscillatory centers of the CPG. In other words, the effect of a reflex depends on the state of the CPG and may even be shaped by higher-level commands. From the embodied cognition point of view the capability of the CPG to influence the future sensory input with the signals it projects to the motoneurons deserves particular attention. The sensory-motor loop does not constitute the basis for the CPG to work, as this would correspond to peripheral control. However, it may still play an essential role for efficient locomotion.

3.2 Nonlinear oscillators with balanced couplings

3.2.1 Introduction

The coupled nonlinear oscillators presented in the subsequent sections belong to a broad class of mathematical problems, called perturbed nonlinear dynamical systems:

$$\dot{\mathbf{q}} = F(\mathbf{q}) + \mathbf{p}$$

where \mathbf{q} is the vector of state variables (x and v in our case) and \mathbf{p} a perturbation vector. An oscillator is defined as a dynamical system that converges to a periodic solution if it is unperturbed ($\mathbf{p} = \mathbf{0}$). Any oscillator may be transformed into a polar coordinate system:

$$\begin{cases} \dot{\theta} = \omega_0 + p_\theta \\ \dot{r} = F_r(r, \theta) + p_r \end{cases}$$

where ω_0 is the natural frequency of the unperturbed oscillator. The evolution of the radius r is defined by the dynamical system F_r . The perturbation \mathbf{p} has been decomposed into two components, one acting on the phase (p_θ) and the other acting in the direction of the radius (p_r). In order to understand the effect of the perturbation \mathbf{p} it is useful to look at the limit cycle (i.e. the set of \mathbf{q} that the system periodically visits after convergence) from a geometrical point of view. As illustrated on figure 8, p_θ modifies the phase but p_r will be damped out and have a negligible effect [Buchli & Ijspeert 2004].

Multiple oscillators may be coupled together by defining a perturbation that is a function of the state of the other oscillators. Before turning to the general case, suppose we have a system that consists of only two coupled oscillators:

$$p_{\theta,1} = 0$$

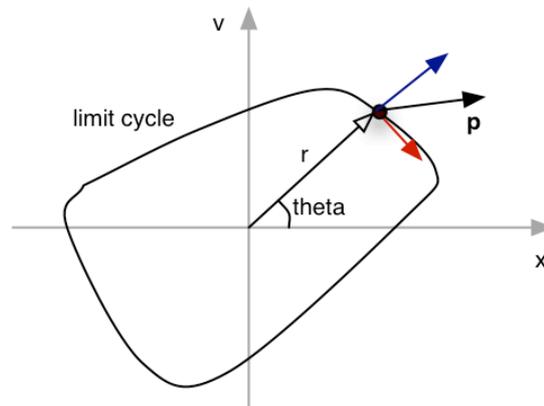


Figure 8: Sketch of a limit cycle. This oscillator has state variables x and v . It may be transformed into a polar coordinate system θ - r . The perturbation can be decomposed into p_θ (in red) and p_r (in blue). The latter one is generally damped out and has no effect on the phase.

$$p_{\theta,2} = f(\mathbf{q}_1)$$

The coupling can result in various dynamics, chaos or stabilization at a constant phase difference ϕ are the most common cases. We are obviously interested in the latter one. As I will discuss in the next chapter, it is essential that the phase difference after convergence (ϕ) can be predicted for many reasons. In other words, we would like to find a function g that gives us ϕ in function of \mathbf{p} . Actually it's the inverse that is most useful: A specific phase difference ξ could be set if a function h was known that perturbs the system such that it converges to $\phi = \xi$. Suppose \mathbf{p}_1 is zero, i.e. there is a unidirectional coupling from oscillator 1 to oscillator 2:

$$\begin{aligned} g(\mathbf{p}_2) &= \phi \\ \mathbf{p}_2 &= h(\xi, \mathbf{q}_1) \quad \Rightarrow \quad \phi = \xi \end{aligned}$$

Unfortunately the relation between the perturbation and the resulting phase difference is far from trivial and for some oscillators it may even be impossible to derive it analytically. For the interested reader, the paper by Buchli and Ijspeert is a good starting point [Buchli & Ijspeert 2004]. The authors discuss phase sensitivity analysis of amplitude controlled phase oscillators. The methodology involves transforming the system into polar coordinates and deriving a differential equation for the phase difference. This method might also be applicable to the nonlinear oscillator that I present in the subsequent sections, but I follow a more intuitive and less formal approach.

Through numerical experiments and with the help of graphical plots I gain insight into the dynamics of the system. Based on the understanding of the dynamics I succeed „guessing“ the functions g and h . Finally I prove the correctness of my solutions both numerically and analytically (luckily, the analytical prove of correctness of a known solution is trivial compared to actually deriving the solution from the system).

As mentioned above, there are different types of oscillators. From now on, when I use the term nonlinear oscillator or just oscillator, I always refer to the specific type that is presented in the following sections.

3.2.2 Standalone nonlinear oscillator

Before tackling the problem of designing coupled nonlinear oscillators that can be set to a specific phase difference, let's consider the simple case of a single nonlinear oscillator without couplings. Experience shows that sinusoidal signals are well suited for locomotion control:

$$x = A \sin(2\pi ft + \varphi)$$

where A is the amplitude, f the frequency and φ the phase. By taking the first and the second derivative of x , a first order differential equation can be derived:

$$\begin{cases} \tau \dot{x} = v \\ \tau \dot{v} = -x \end{cases}$$

$$\text{with } \tau = \frac{1}{2\pi f}$$

The amplitude A is only implicitly defined by this linear differential equation. A depends on the initial conditions and perturbations of the state variables will have an effect on it (analogous to the perturbation of an ideal spring). We must therefore add a new term that drives the system to a limit cycle with precise amplitude:

$$\begin{cases} \tau \dot{x} = v \\ \tau \dot{v} = -\alpha \frac{x^2 + v^2 - E}{E} v - x \end{cases} \quad (1)$$

The expression $x^2 + v^2$ represents the actual energy of the oscillator and $x^2 + v^2 - E$ is the energy error of the oscillator. Therefore, the nonlinear term may be understood as the normalized energy error multiplied by α and v . The positive constant α can be used for tuning the attracting force to the limit cycle. The bigger α , the faster the convergence. The

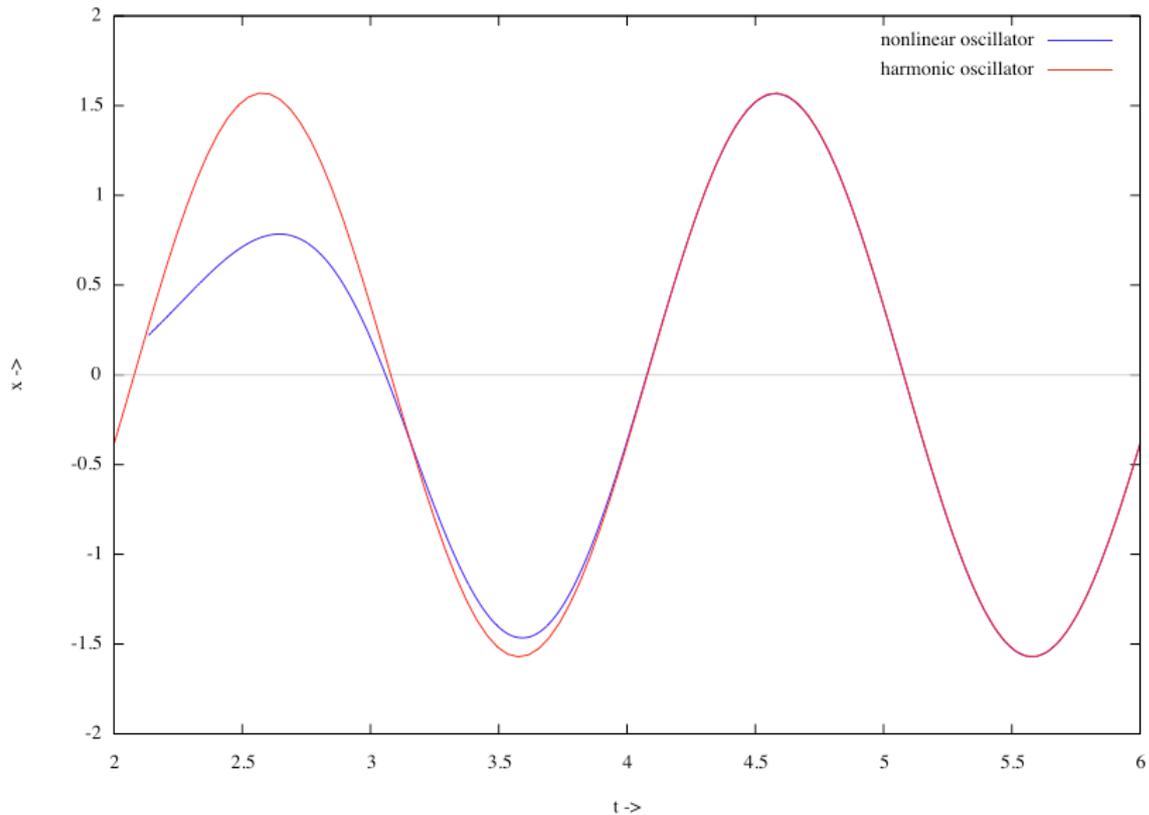


Figure 9: Convergence of a standalone nonlinear oscillator (equation 1). The energy has been set to $\pi/4$ and τ is $1/\pi$, consequently x converges to a sinus with amplitude $\pi/2$ and frequency $1/2$ (see equation 2). The phase depends on the initial conditions and has been tuned manually in the harmonic oscillator to match the NOL. The initial state is set randomly. Note that I also initialize each NOL with a different time in order to simulate the asynchronous clocks of the modules.

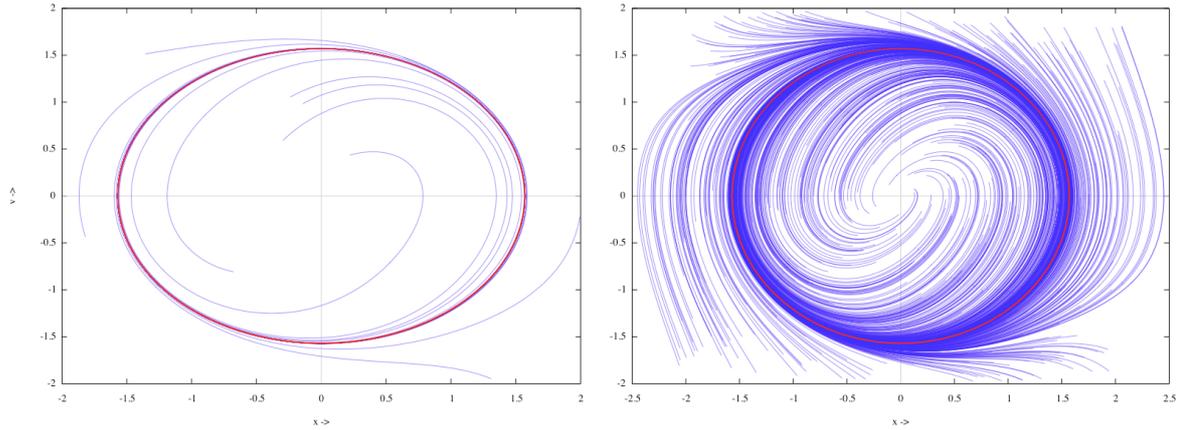


Figure 10: Limit cycle of a standalone nonlinear oscillator (equation 1). Each line corresponds to a separate run with random initial conditions as illustrated on figure 1. Ten runs are plotted on the left, 500 on the right.

multiplication by v implies faster convergence when the velocity is high. As illustrated on Figure 9 and 10, on the limit cycle of (1) x and v converge to:

$$\begin{cases} \tilde{x} = A \sin\left(\frac{t}{\tau} + \varphi\right) \\ \tilde{v} = A \cos\left(\frac{t}{\tau} + \varphi\right) \end{cases} \quad (2)$$

with $A = \sqrt{E}$ and $f = \frac{1}{2\pi\tau}$

The phase φ depends on the initial conditions. You can easily verify that this is indeed a solution of (1) by replacing x and v by \tilde{x} and \tilde{v} in the equations. Convergence to the limit cycle is guaranteed for arbitrary initial conditions except for the singularity when both x_0 and v_0 equal zero. Refer to figure 10.

3.2.3 Coupled nonlinear oscillators

I will now explain how several nonlinear oscillators can be coupled together in order to synchronize at a constant phase difference. Following [Ijspeert, et al. 2003] I started with a simple coupling strategy which consists of projecting signals proportional to the x and v states. The perturbation \mathbf{p}_i of oscillator i is:

$$\begin{cases} p_{x,i} = 0 \\ p_{v,i} = \sum_j (a_{ij}x_j + b_{ij}v_j) \end{cases} \quad (3)$$

where a_{ij} and b_{ij} are positive constants that imply a specific phase difference on the limit cycle. As illustrated on figure 11, one can classify the coupling scheme of two oscillators as:

- a) Uncoupled: $a_{ij} = b_{ij} = a_{ji} = b_{ji} = 0$. Obviously the phase difference can't be controlled.
- b) Unidirectional: a_{ji} and b_{ji} are both zero. For any a_{ij} and b_{ij} (not both zero) the oscillators converge to a constant phase difference.
- c) Bidirectional with two free parameters: $a_{ji} = -a_{ij}$ and $b_{ji} = b_{ij}$. In this case, the links in both directions drive the system to the same phase difference, thereby accelerating convergence (see next section).
- d) Bidirectional with four free parameters: if none of the above applies. Generally, the two links are conflicting, i.e. the two couplings try to drive the system to two diverging phase differences. The oscillators may stabilize at a phase that balances the conflicting perturbations or fall into chaotic behavior.

To model a CPG a network of nonlinear oscillators is needed (see Figure 12). If the network is a tree, i.e. a fully connected and acyclic graph, the phase differences can be analyzed pair wise. In other words, the coupling between two oscillators alone determines their phase difference. On the other hand, in cyclic networks complex dynamics emerge from conflicting forces analogously to the case of bidirectional couplings with four free parameters (which are also a cycle) mentioned above.

Trees of nonlinear oscillators always converge to constant phase differences. Since chaotic dynamics slow down both the genetic algorithm and the online optimization I assume that the networks are always trees in this project. Further investigations whether this assumption is biologically plausible would be very interesting within this context.

With the simple coupling given by (3), the perturbation is strongly influenced by the energies E_j of the coupled oscillators. For example, if the amplitude of oscillator j is very small, the coupling will be weak (unless compensated with high values of a_{ij} and b_{ij}) which results in a slow convergence. As we want the strength of the coupling to be independent from the amplitude of the emitting oscillator, the perturbation must be normalized by the *actual* amplitude $(x_j^2 + v_j^2)^{1/2}$ (which might be very different from $E_j^{1/2}$ before convergence). Hence the general equation for the coupled oscillator i is:

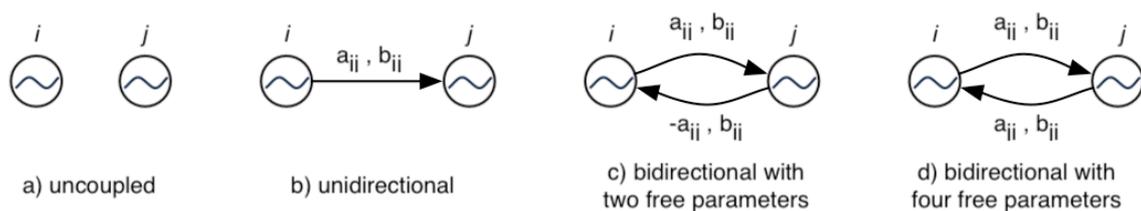


Figure 11: Classification of coupling schemes between two nonlinear oscillators.

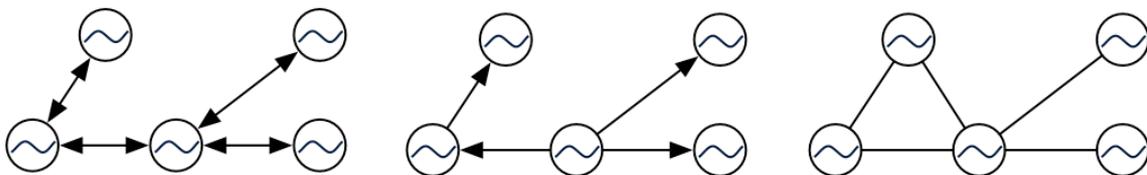


Figure 12: In an acyclic network the phase difference between two coupled oscillators is determined only by the coupling between these two oscillators (first two examples). Cycles result in more complex dynamics and a pair wise analysis of the phase difference is not sufficient (on the right).

$$\begin{cases} \tau \dot{x}_i = v_i \\ \tau \dot{v}_i = -\alpha \frac{x_i^2 + v_i^2 - E_i}{E_i} v_i - x_i + \sum_j \frac{a_{ij} x_j + b_{ij} v_j}{\sqrt{x_j^2 + v_j^2}} \end{cases} \quad (4)$$

The attentive reader might have noticed that the same argument also holds for oscillator i . In equation (3), the coupling is normalized with respect to the amplitudes of the emitting oscillators but the amplitude of oscillator i is not taken into account. In other words, if the amplitude A_i is halved, the influence of the coupling will be twice as strong. I observed this phenomenon in form of varying convergence times for equivalent initial conditions with different values of the energy parameters. A remedy would be to multiply the coupling term with the actual amplitude of oscillator i :

$$p_{v,i} = \sqrt{x_i^2 + v_i^2} \cdot \sum_j \frac{a_{ij} x_j + b_{ij} v_j}{\sqrt{x_j^2 + v_j^2}} \quad (5)$$

However I decided not to use this form (see also equation 14). Hence, the following sections build upon equation (4).

3.2.4 Predicting the phase difference of two oscillators

In the previous section I explained how two oscillators can be coupled together to synchronize at a constant phase difference. However, efficient locomotion requires synchronization at a specific phase. When using genetic algorithms or other optimization methods it is not necessary to understand the effect of the coupling parameters on the phase. However, with phase prediction we can drastically improve these algorithms by limiting and structuring the search space.

First of all I measured the phase difference in function of the coupling parameters in a series of experiments. Obviously, the phase difference can only be measured after convergence of the system. Correctly detecting convergence and measuring the phase is not as easy as one might think and apparent irregularities in the phase are sometimes falsely interpreted as a consequence of the nonlinearity of the system.

As you can see in figure 13, the relation between the coupling parameters is much more regular than previously expected. It seems that on any ray parting from the origin, the phase is constant. Furthermore, the phase appears to be a linear function of the angle that the ray forms with one of the axes. The key to phase prediction is a transformation of a and b into polar coordinates (refer to figure 14):

$$\begin{aligned} a &= r \cos(\xi) \\ b &= r \sin(\xi) \end{aligned} \quad (6)$$

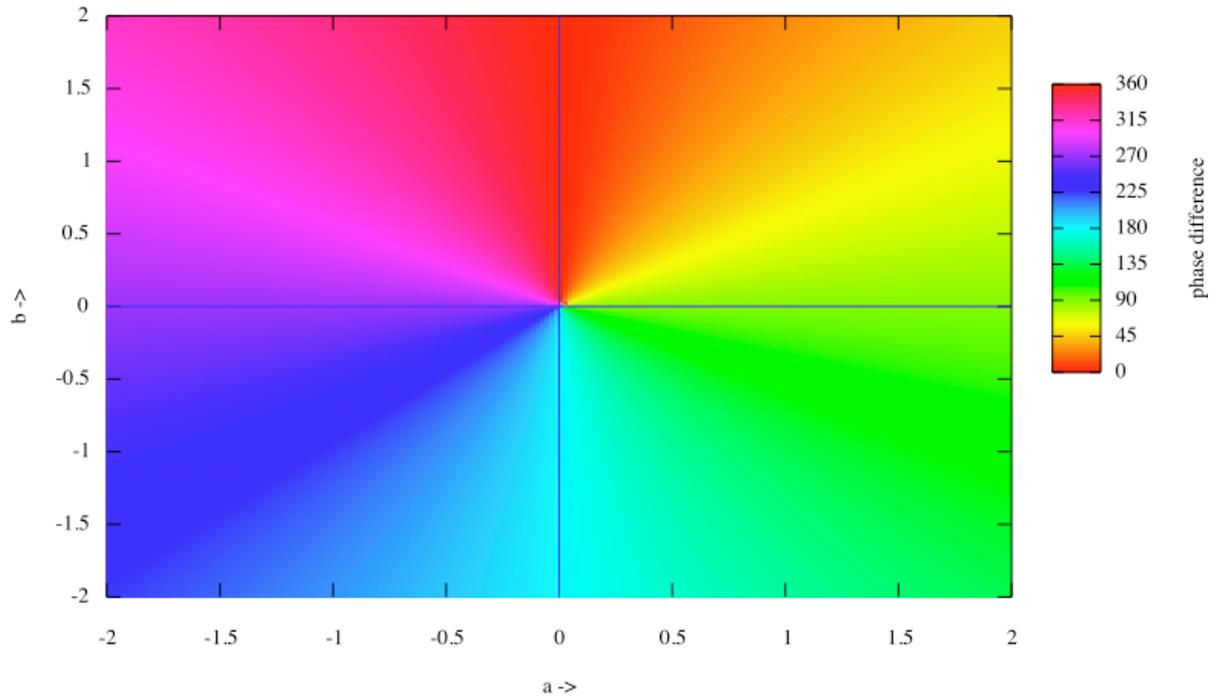


Figure 13: Phase difference between two coupled oscillators. Each combination of the coupling parameters a and b has been tested (by increments of 0.02) using random values for the initial conditions *and* the energy parameters. As expected, the results are identical for both unidirectional and bidirectional couplings with two free parameters.

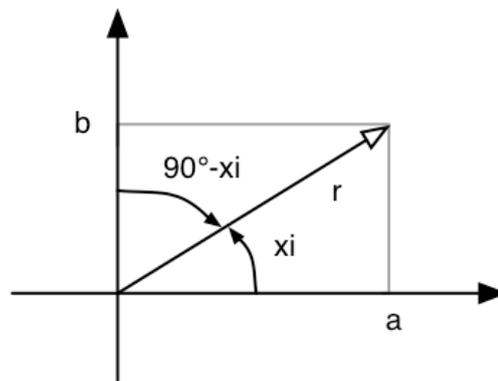


Figure 14: Transformation of the coupling parameters into a polar coordinate system r - ξ .
The plot on figure 6 suggests that the phase difference grows linearly with $\pi/2 - \xi$.

Note that this is not the transformation of the oscillator into polar coordinates that has been mentioned in Section 3.2.1. It is just a different notation for the coupling parameters. However, with this notation we can easily guess the relation between the phase difference ϕ and the coupling.

Based on figure 13 I hypothesize that the function g that predicts the phase from the perturbation is:

In polar coordinates:
$$g(r, \xi) = \frac{\pi}{2} - \xi \tag{7}$$

In Cartesian coordinates:
$$g(a, b) = \begin{cases} \pi/2 & (a > 0 \wedge b = 0) \\ \arctan\left(\frac{a}{b}\right) & (b \neq 0) \\ -\pi/2 & (a < 0 \wedge b = 0) \end{cases} \tag{8}$$

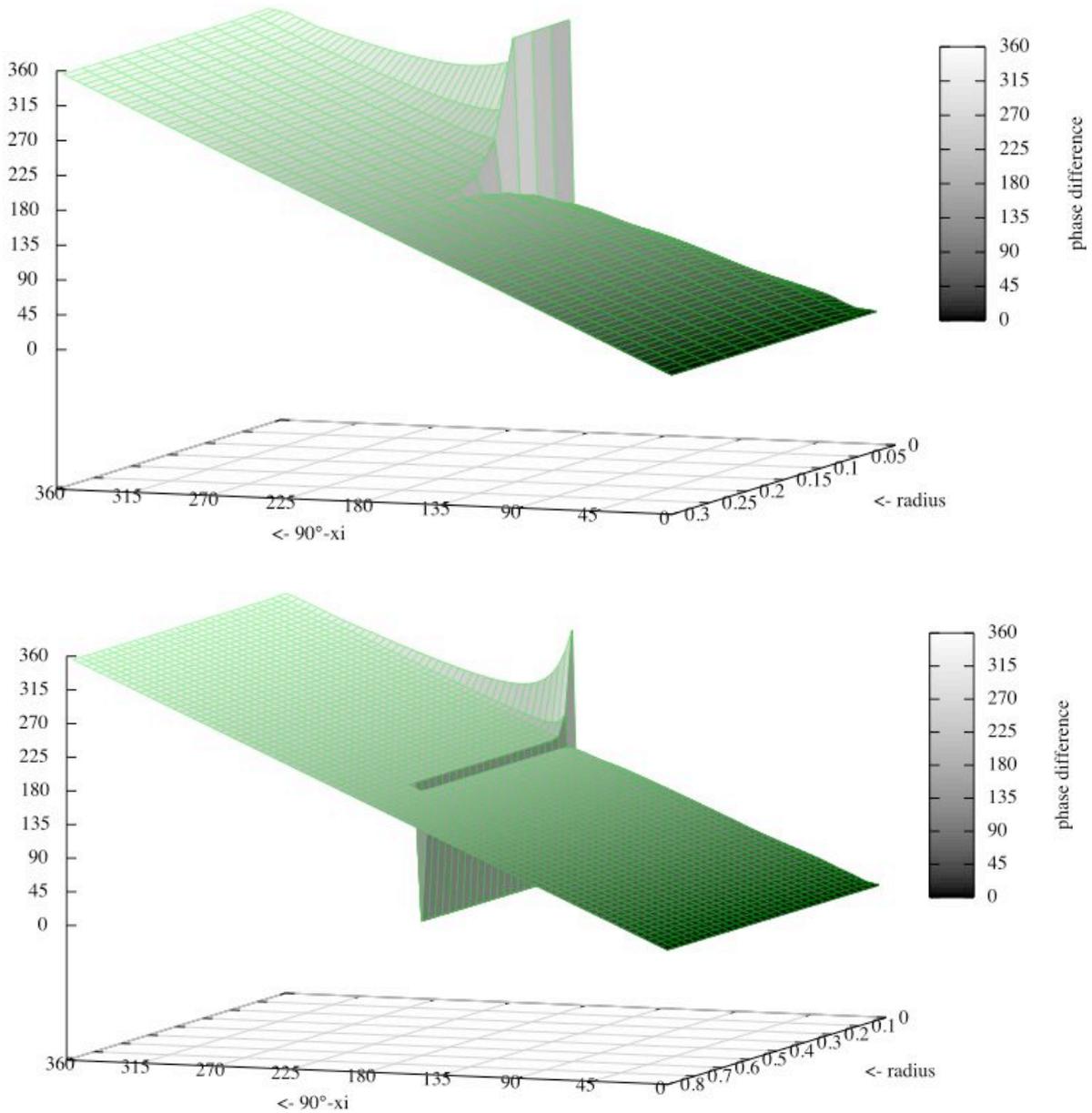


Figure 15: Plots of the phase difference (vertical axis) in function ξ and r . For the upper diagram unidirectional couplings were used, for the lower bidirectional couplings with two free parameters. The ,canyon' in the second plot is a typical example of an artifact (in the phase plots of figure 6 I eliminated it by using much higher precision and smaller time steps).

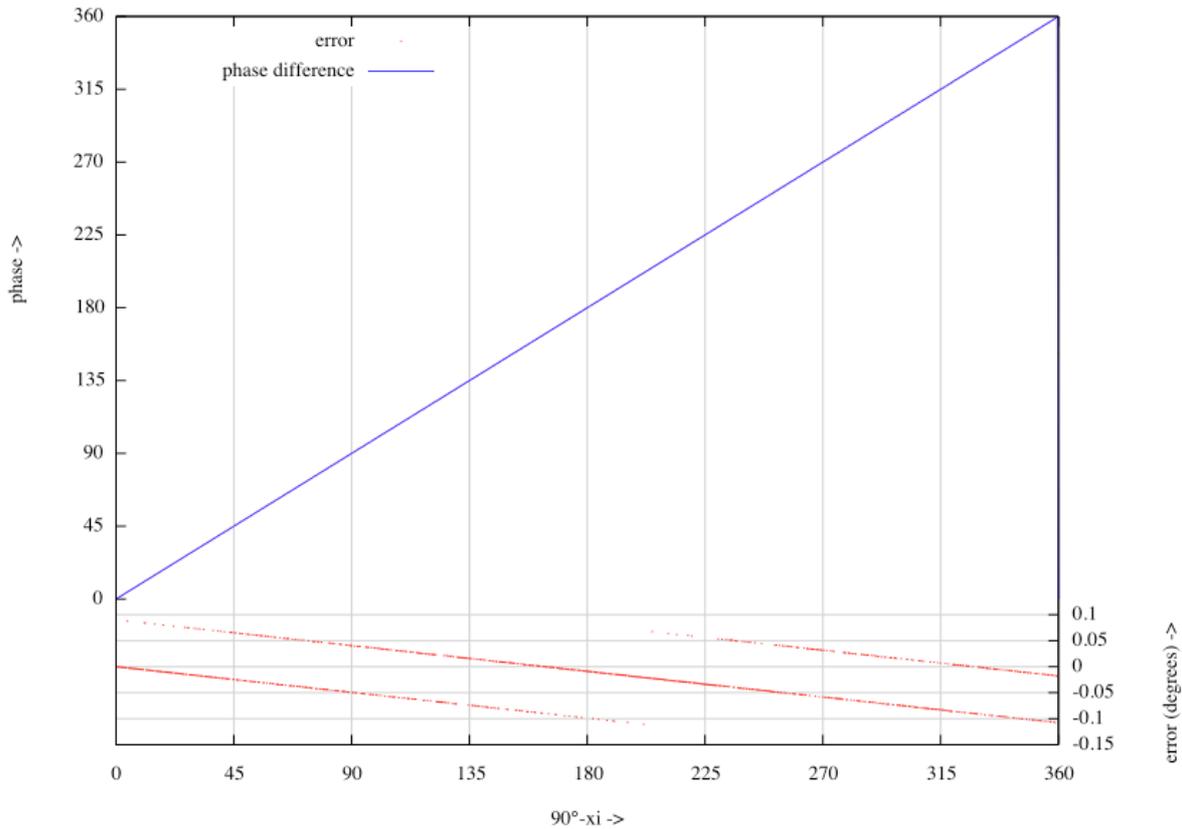


Figure 16: Error of the phase prediction function g (equations 7 and 8). For this plot, bidirectional couplings with two free parameters were used. Unidirectional couplings gave the same results. To speed up the experiments I only used a precision of 0.01 radians (approximately 0.5 degrees) for detecting the convergence, thus the error lies within the precision of the phase measurement and could be reduced by increasing the precision.

Before trying to prove this hypothesis analytically, we can test it numerically. In order to do so I first plotted the phase difference in function of ξ and r using both unidirectional and bidirectional couplings with two free parameters. Indeed, the plots of figure 8 confirm equations (7) and (8) for sufficiently big r (i.e. for sufficiently strong couplings).

In another experiment I calculated the error of the phase prediction function g for a constant radius r . Again, the hypothesis was confirmed. As shown in figure 16, 0.1 and -0.15 degrees bound the error, which is sufficiently small for our purposes. Furthermore, as explained in figure 16 the error is only a numerical artifact explained by the use of a relatively low precision (0.01 radians) for convergence detection.

By rewriting the coupling term $p_{i,v}$ of the nonlinear oscillator i (equation 4) using the new notation (equation 6) we can understand the effect of the coupling parameters:

$$p_{v,i} = \sum_j \frac{a_{ij}x_j + b_{ij}v_j}{\sqrt{x_j^2 + v_j^2}} = \sum_j \left(r_{ij} \cdot \frac{\cos(\xi_{ij})x_j + \sin(\xi_{ij})v_j}{\sqrt{x_j^2 + v_j^2}} \right) \quad (9)$$

Now suppose that x_j and v_j converged to:

$$\begin{cases} \tilde{x}_j = A_j \sin\left(\frac{t}{\tau} + \theta_j\right) \\ \tilde{v}_j = A_j \cos\left(\frac{t}{\tau} + \theta_j\right) \end{cases}$$

then, by replacing x_j and v_j in (9) we get:

$$\begin{aligned} p_{v,i} &= \sum_j \left(r_{ij} \cdot \frac{\cos(\xi_{ij}) \cdot A_j \sin\left(\frac{t}{\tau} + \theta_j\right) + \sin(\xi_{ij}) \cdot A_j \cos\left(\frac{t}{\tau} + \theta_j\right)}{A_j} \right) \\ &= \sum_j \left(r_{ij} \sin\left(\frac{t}{\tau} + \theta_j + \xi_{ij}\right) \right) \\ &= \sum_j \left(r_{ij} \cos\left(\frac{t}{\tau} + \theta_j - \left(\frac{\pi}{2} - \xi_{ij}\right)\right) \right) \end{aligned} \quad (10)$$

The first simplification is achieved with a trigonometric addition theorem, then we apply the reduction formula $\sin(\pi/2 + \alpha) = \cos(\alpha)$. Now suppose that on the limit cycle:

$$\tilde{v}_i = \tilde{A}_i \cos\left(\frac{t}{\tau} + \theta_i\right) \quad (11)$$

Note that $\tilde{A}_i \neq \sqrt{E_i}$. The hypothesis (7) implies that:

$$\phi_{ij} = \theta_j - \theta_i = \frac{\pi}{2} - \xi_{ij} \quad \Rightarrow \quad \theta_i = \theta_j - \left(\frac{\pi}{2} - \xi_{ij}\right) \quad (12)$$

Hence, the cosine in (10) is the normalized velocity of oscillator i on the limit cycle:

$$p_{v,i} = \sum_j \left(r_{ij} \frac{\tilde{v}_i}{\tilde{A}_i} \right) \quad (13)$$

If the same radius (strength of the coupling) is used for all couplings, the perturbation is:

$$p_{v,i} = kr \frac{\tilde{v}_i}{\tilde{A}_i} \quad (14)$$

where k is the number of couplings. The coupling perturbs the oscillator with a signal that is proportional to the normalized 'target' velocity \tilde{v} ! Note that this signal \tilde{v} is different from v before convergence. The effect of such a perturbation is that v synchronizes with \tilde{v} .

The speed of convergence is proportional to the number of couplings k and to the strength of the couplings r . The signal \tilde{v}_i is normalized, thus the force of the perturbation is not proportional to the amplitude of the oscillator. Therefore, the same coupling results in either faster or slower convergence if the energy of the oscillator is decreased or

increased. This is the exact same observation that has already been made in Section 3.2.3. Now we see that multiplying the coupling term with the actual amplitude of the oscillator (equation 5) would indeed lead to a convergence independent from the energy. However, slower convergence for oscillations with big amplitudes generally makes sense in a robotics application: oscillations with small amplitudes can usually be changed faster by the motor than oscillations with big amplitudes because a smaller torque is needed in the first case. This is the reason why I decided not to use equation (5).

Note that, even though verified numerically, I have still not proved the hypothesis about function g (equations 7, 8). Specifically, equations (13, 14) could only be derived under the assumption that the hypothesis is true. Luckily this is the case, as I will prove in Section 3.2.7.

Finally, yet most importantly, we can use equations (7, 8) to define a coupling that sets the phase difference ϕ_{ij} between two oscillators to a specific phase $\tilde{\phi}_{ij}$:

$$p_{v,ij} = r_{ij} \cdot \frac{\cos\left(\frac{\pi}{2} - \tilde{\phi}_{ij}\right)x_j + \sin\left(\frac{\pi}{2} - \tilde{\phi}_{ij}\right)v_j}{\sqrt{x_j^2 + v_j^2}} \Rightarrow \phi_{ij} = \tilde{\phi}_{ij} \quad (15)$$

3.2.5 Convergence of two coupled nonlinear oscillators

Two oscillators have converged if their phase difference remains constant. In the previous section we have seen that the radius gives the strength of a coupling:

$$r = \sqrt{a^2 + b^2}$$

Choosing an appropriate r is essential for the optimization algorithms: Convergence should be quick to speed up the algorithm but ideally the trajectories should not change faster than the mechanical dynamics allow. Furthermore it is important to know an upper bound for the time it takes the system to converge with arbitrary initial conditions.

In order to study the effect of r I measured the number of periods until convergence in function of the coupling parameters. As illustrated in figure 10, one should choose at least $r > 0.75$ in the case of unidirectional couplings and $r > 0.5$ in the case of bidirectional couplings with two free parameters in order to stay out of the red and purple areas where convergence is impracticably slow for locomotion.

One could imagine that choosing initial positions on the limit cycle might accelerate convergence. Figure 18 shows the number of periods until convergence for initial conditions $x_0 = 0$ and $v_0 = A$. Comparing figures 17 and 18 we see that convergence is actually faster for random initial conditions. Geometrically speaking, a possible explanation is that the trajectories can take 'shortcuts' from the initial positions and directly approach the limit cycle with the correct phase (remember Figure 10). This is apparently faster than changing to the correct phase difference on the limit cycle itself. Finally, to choose optimal values for r I defined a measure of the convergence in function of r . This measure is given by:

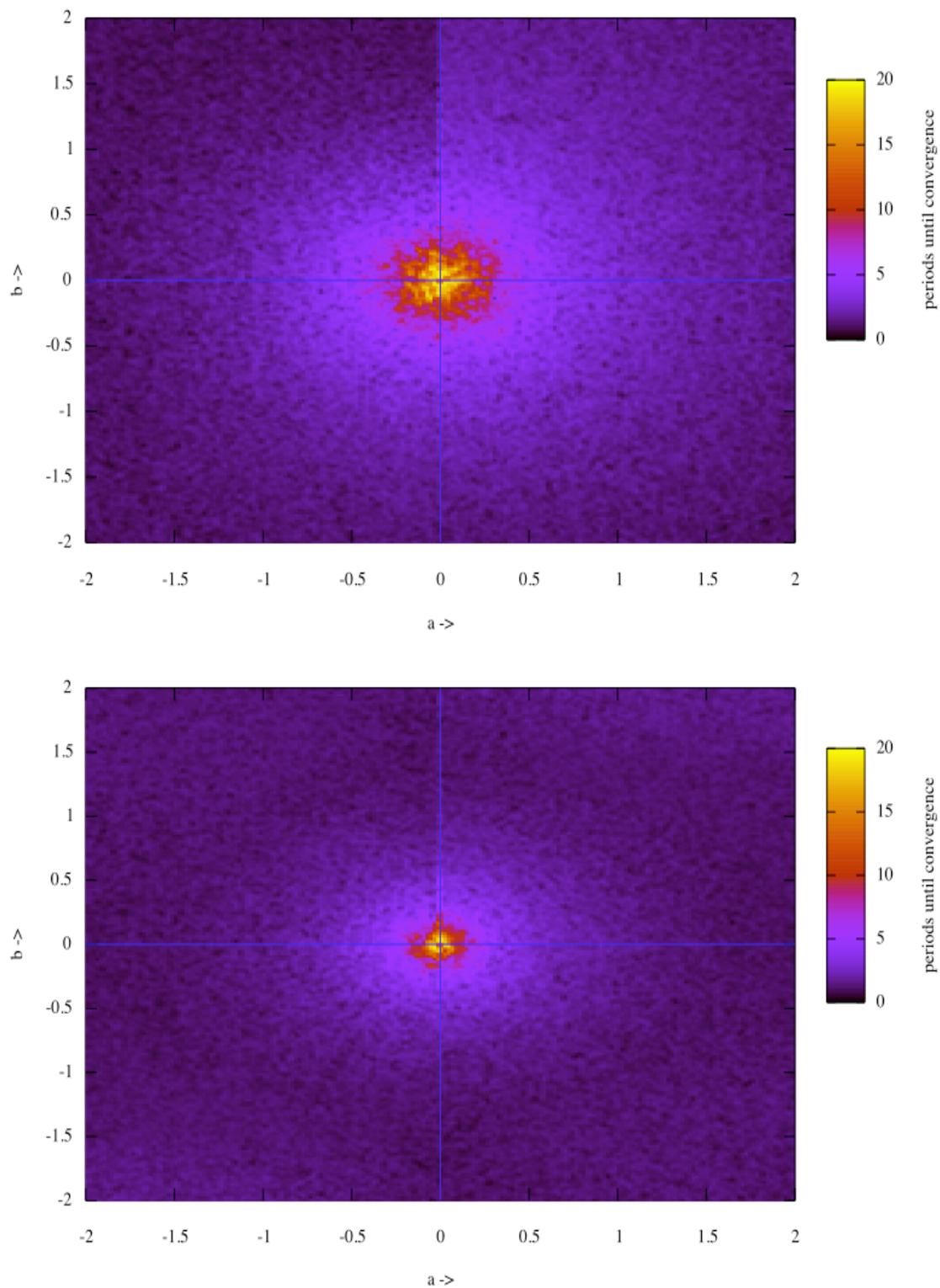


Figure 17: Number of periods until convergence in function of the coupling parameters a and b . For each run random initial conditions are set, this explains the noise. On the top, a unidirectional coupling scheme is used. The apparent faster convergence on the left side of the b -axis compared to the right side is just an artifact (it's because I measure the convergence in whole numbers). As expected, convergence is faster for bidirectional couplings with two free parameters (below).

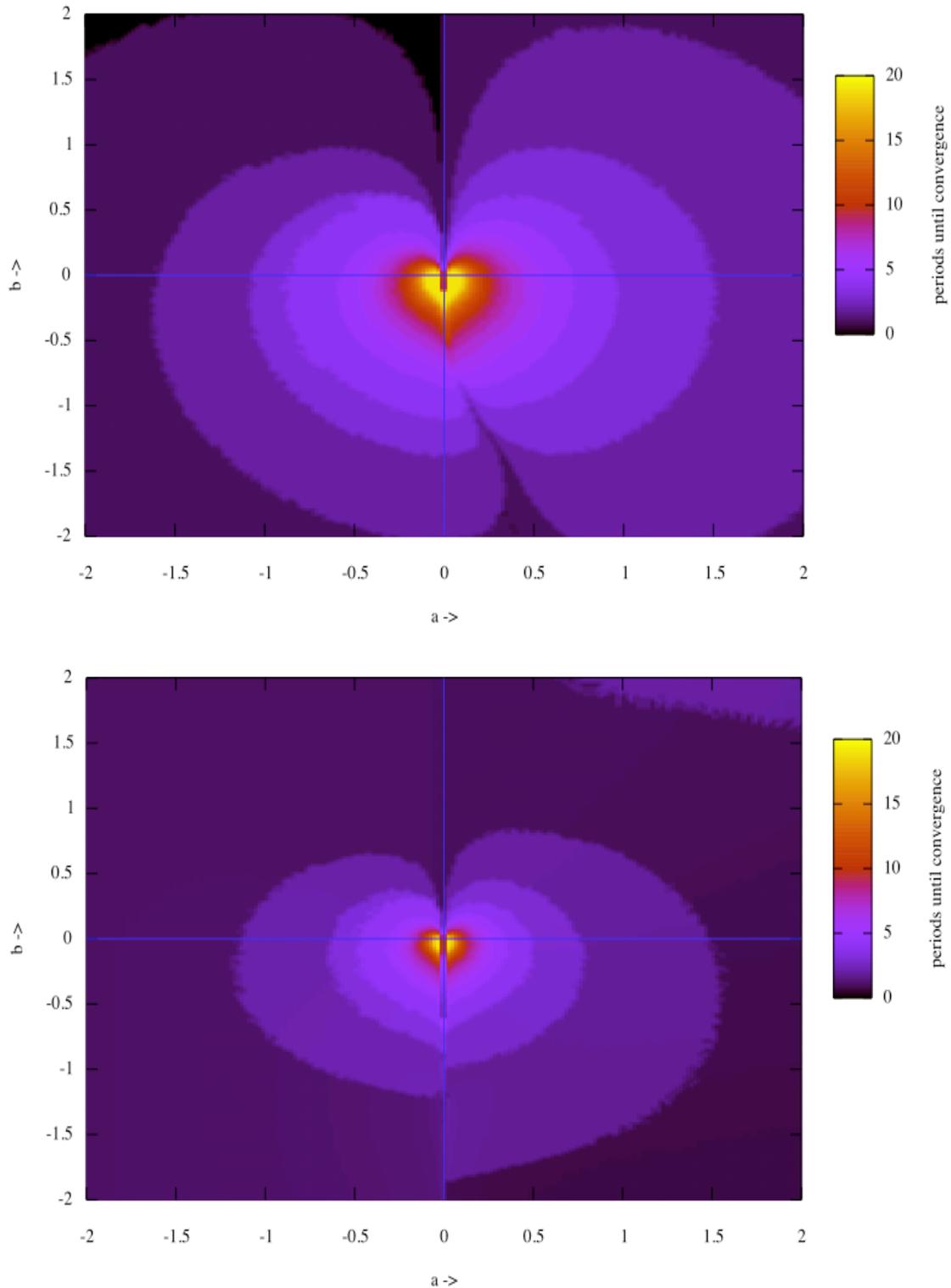


Figure 18: Number of periods until convergence in function of the coupling parameters a and b for constant initial conditions $x_0 = 0$ and $v_0 = A$. Again, on the top a unidirectional coupling scheme is used and below bidirectional couplings with two free parameters. Surprisingly, convergence slowed down for big r in the upper left corner of the second plot, thereby indicating that convergence time might increase if the coupling is too strong. This observation is confirmed in figure 19.

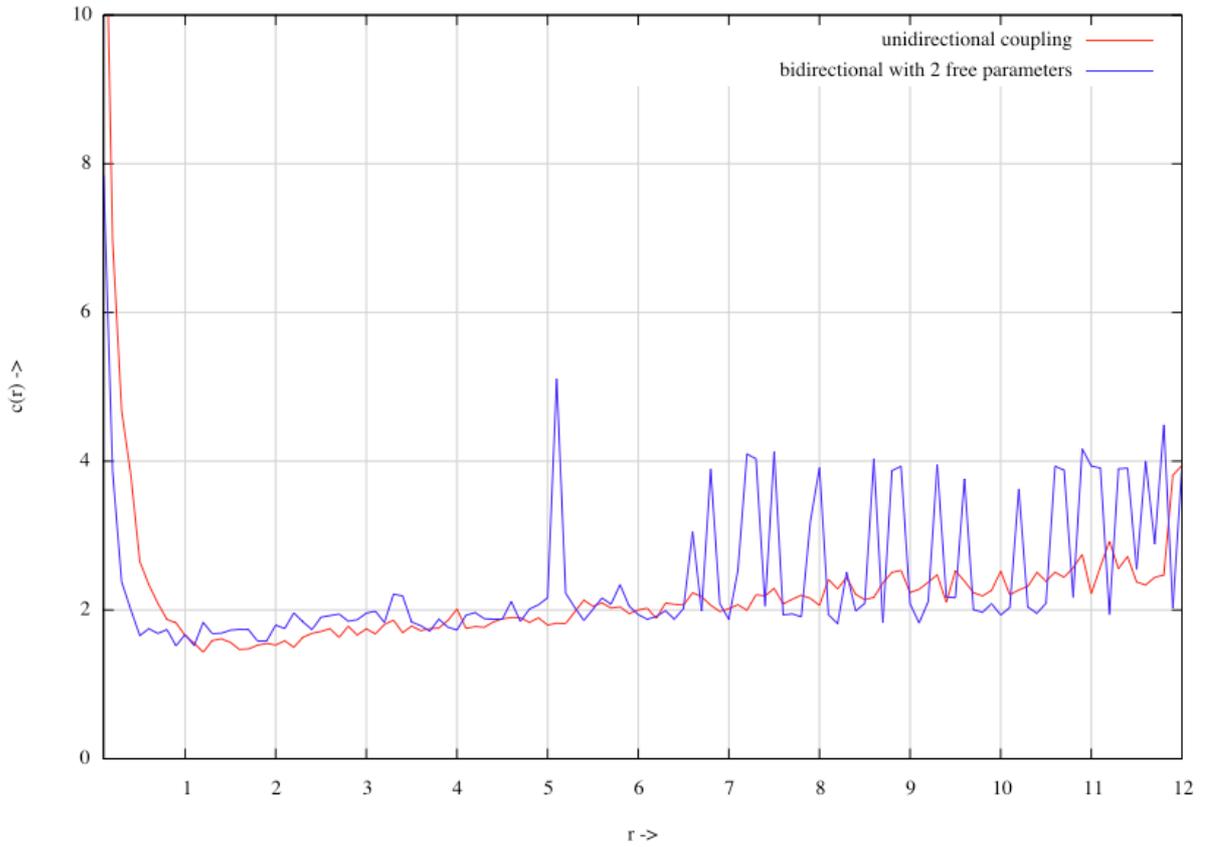


Figure 19: For each radius r , the number of periods until convergence p of 100 separate runs is measured. The angle ξ is incremented for each run, starting at 0 for the first run and going until 2π for the last run. Random initial conditions *and* random amplitudes are set for every run. The convergence for a specific radius $c(r)$ (vertical axis) is defined as being the square mean of the hundred p that have been measured for r (see equation 16). Interestingly, the convergence gets slower for very strong couplings. The spikes that occur in the case of bidirectional couplings indicate individual runs that converge very slowly.

$$c(r) = \sqrt{\frac{p_0^2 + p_1^2 + \dots + p_N^2}{N}} \quad (16)$$

where p_i is the number of periods until convergence with an angle of $\xi_i = i \frac{2\pi}{N}$. For each p_i (i.e. for each run) random initial conditions and random amplitudes are set for both oscillators. Taking the square mean instead of the arithmetic mean makes $c(r)$ more sensible to runs that take exceedingly long to converge. The results are given in figure 12. Interestingly, the convergence slows down as r increases after reaching an optimum at roughly:

$$r_{opt} = 1.5 \quad (\text{for unidirectional couplings})$$

$$r_{opt} = 1.0 \quad (\text{for bidirectional couplings with two free parameters})$$

3.2.6 The amplitude difference

As explained above, a standalone nonlinear oscillator converges to a sinusoidal limit cycle with an amplitude equal to the square root of the energy parameter. Unfortunately, the perturbation of the coupling does not only have an effect on the phase but also on the amplitude \tilde{A}_i on the limit cycle of oscillator i :

$$\tilde{A}_i > \sqrt{E_i}$$

Intuitively, we can understand that the signal of the coupling increases the energy in the oscillator. I define the amplitude difference e_A as being:

$$e_A = \tilde{A}_i - \sqrt{E_i}$$

As you can see in figure 20, the amplitude difference increases with the radius, i.e. the strength of the coupling. The difference is significant and can certainly not be neglected. Actually, the influence of the coupling on the amplitude is so strong that oscillations with amplitudes smaller than approximately 20° are not possible with couplings of reasonable strength. Making matters worse, the difference of a coupled oscillator i depends not only on the strength of the coupling but also on its energy E_i (refer to figure 21).

I spent some time trying to find the relation between the amplitude difference e_A , r and E_i , but I didn't succeed. Finally I found a much better solution for the problem: In the next section I introduce a modified coupling term that implies a zero amplitude difference.

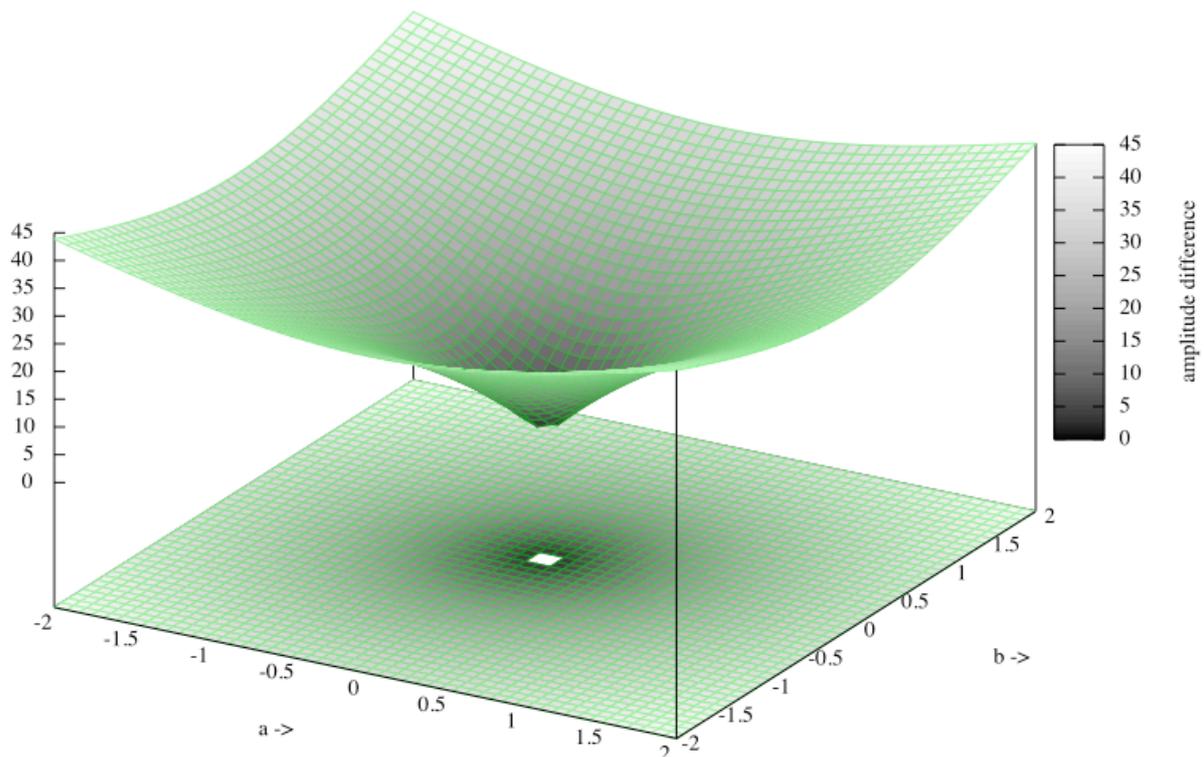


Figure 20: The amplitude difference (vertical axis) of a coupled oscillator increases (not linearly) with the strength of the coupling.

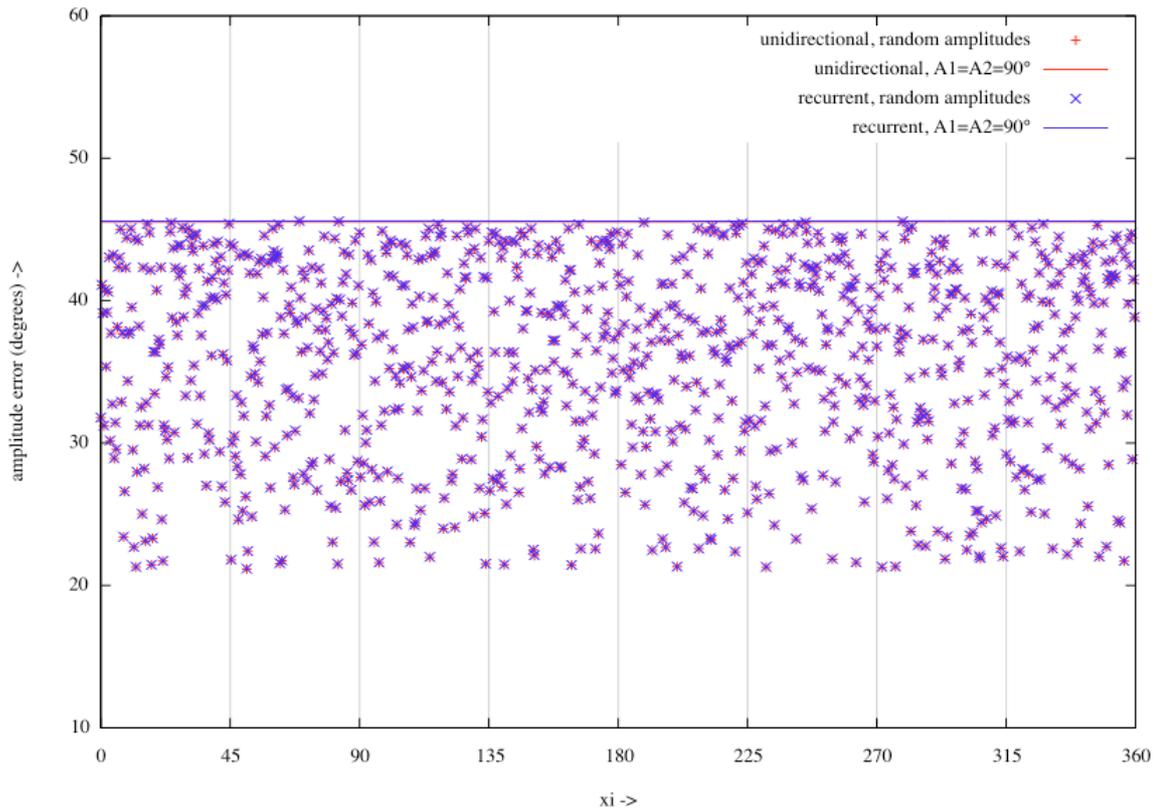


Figure 21: The amplitude difference (vertical axis) for a constant radius. In a first set of experiments, energy parameters of both coupled oscillators were the same for all runs (this corresponds to the line on the top). In a second series, random amplitudes were set for both oscillators (the points). The same random seed was used for the series with the unidirectional (red) and the bidirectional couplings with two free parameters (blue). Note that the points always coincide; hence the amplitude difference is the same for unidirectional and bidirectional couplings.

3.2.7 Nonlinear oscillators with energy balanced couplings

The amplitude difference discussed in the previous section is problematic for locomotion control. First of all because small oscillations become impossible for coupled oscillators but also because the difference depends on the strength of the coupling r . As mentioned above, r can be used to tune the speed of convergence. For locomotion control, r is an important parameter, especially in the context of gait transitions. For example, it may be useful to experiment with different coupling strengths for a specific gait transition. One can even imagine a robot that actively controls the speed of the gait transition via the parameter r . However, since r strongly influences the amplitudes, modifying the strength of the couplings will generally have a very negative impact on the locomotion gait (they are very sensible not only to the phases, but also to the amplitudes).

As the amplitude difference stems from an increase of the actual energy caused by the coupling, the most obvious solution is to add a new term to the coupling that balances the additional energy out (hence the name *energy balanced couplings*). Thus I propose to introduce an energy balancing term Γ in the standard equation (4) for the coupled nonlinear oscillators. After experimenting with different terms, I succeeded finding a term Γ that works. Using polar coordinates for the coupling parameters, the energy balanced coupling is:

$$p_{v,i} = \sum_j \left(r_{ij} \cdot \frac{\cos(\xi_{ij})x_j + \sin(\xi_{ij})v_j}{\sqrt{x_j^2 + v_j^2}} + \Gamma_{ij} \right) \quad (16)$$

$$\text{with } \Gamma_{ij} = -r_{ij} \frac{v_i}{\sqrt{x_i^2 + v_i^2}}$$

As explained below, the energy balanced coupling actually represents the phase error of oscillator i . The following two equations define the nonlinear oscillators with energy-balanced couplings. In equation (17) I use polar coordinates, in equation (18) Cartesian coordinates for the coupling parameters (i.e. the two equations are equivalent):

$$\begin{cases} \tau \dot{x}_i = v_i \\ \tau \dot{v}_i = -\alpha \frac{x_i^2 + v_i^2 - E_i}{E_i} v_i - x_i + \sum_j \left(r_{ij} \cdot \left(\frac{\cos(\xi_{ij})x_j + \sin(\xi_{ij})v_j}{\sqrt{x_j^2 + v_j^2}} - \frac{v_i}{\sqrt{x_i^2 + v_i^2}} \right) \right) \end{cases} \quad (17)$$

$$\begin{cases} \tau \dot{x}_i = v_i \\ \tau \dot{v}_i = -\alpha \frac{x_i^2 + v_i^2 - E_i}{E_i} v_i - x_i + \sum_j \left(\frac{a_{ij}x_j + b_{ij}v_j}{\sqrt{x_j^2 + v_j^2}} - \sqrt{a_{ij}^2 + b_{ij}^2} \frac{v_i}{\sqrt{x_i^2 + v_i^2}} \right) \end{cases} \quad (18)$$

Unlike the standard nonlinear oscillators discussed in the previous sections, an energy balanced oscillator i converges exactly to:

$$\begin{cases} \tilde{x}_i = A_i \sin\left(\frac{t}{\tau} + \theta_i\right) \\ \tilde{v}_i = A_i \cos\left(\frac{t}{\tau} + \theta_i\right) \end{cases} \quad (19)$$

with $\theta_i = \theta_j - \left(\frac{\pi}{2} - \xi_{ij}\right)$

By replacing x_i, v_i with \tilde{x}_i, \tilde{v}_i in equation (17) we can prove that (19) is indeed a solution. These calculations are not included here because they are trivial to do. I shall only show that the energy balanced coupling term $p_{v,i}$ indeed becomes zero after convergence as this helps understanding why $p_{v,i}$ represents the phase error of the oscillator:

$$\begin{aligned} \tilde{p}_{v,i} &= \sum_j \left(r_{ij} \cdot \left(\frac{\cos(\xi_{ij})\tilde{x}_j + \sin(\xi_{ij})\tilde{v}_j}{\sqrt{\tilde{x}_j^2 + \tilde{v}_j^2}} - \frac{\tilde{v}_i}{\sqrt{\tilde{x}_i^2 + \tilde{v}_i^2}} \right) \right) \\ &= \sum_j \left(r_{ij} \cdot \left(\cos\left(\frac{t}{\tau} + \theta_j - \left(\frac{\pi}{2} - \xi_{ij}\right)\right) - \frac{\tilde{v}_i}{A_i} \right) \right) \\ &= \sum_j \left(r_{ij} \cdot \left(\frac{\tilde{v}_i}{A_i} - \frac{\tilde{v}_i}{A_i} \right) \right) = 0 \end{aligned}$$

The simplification of the first term in the sum to a cosine is achieved with trigonometric theorems as explained for equation (10). Note that the discussion of nonlinear oscillators in section 3.2.4 was based on the hypothesis about the phase predicting function g and on the *assumption* that the oscillator converges to a sinusoidal limit cycle. The approach in this section is fundamentally different: As mentioned in the previous paragraph, we can prove that the oscillator converges to \tilde{x}_i, \tilde{v}_i (equation 19) by showing that it is a solution to the differential equation (17).

The phase predicting function g for energy-balanced couplings is the same as the one given in equation (7) for standard couplings. The difference is that in the case of energy balanced couplings the g of equation (7) is not just a hypothesis because it can actually be derived from the solution to the differential equation (19) which is proven true.

The fact that \tilde{x}_i, \tilde{v}_i is the only solution and global attractor for energy balanced coupled oscillators has been confirmed numerically (Figure 22). The phase error is lower than the precision used to measure it and the amplitude error is indeed zero for random amplitudes and initial conditions.

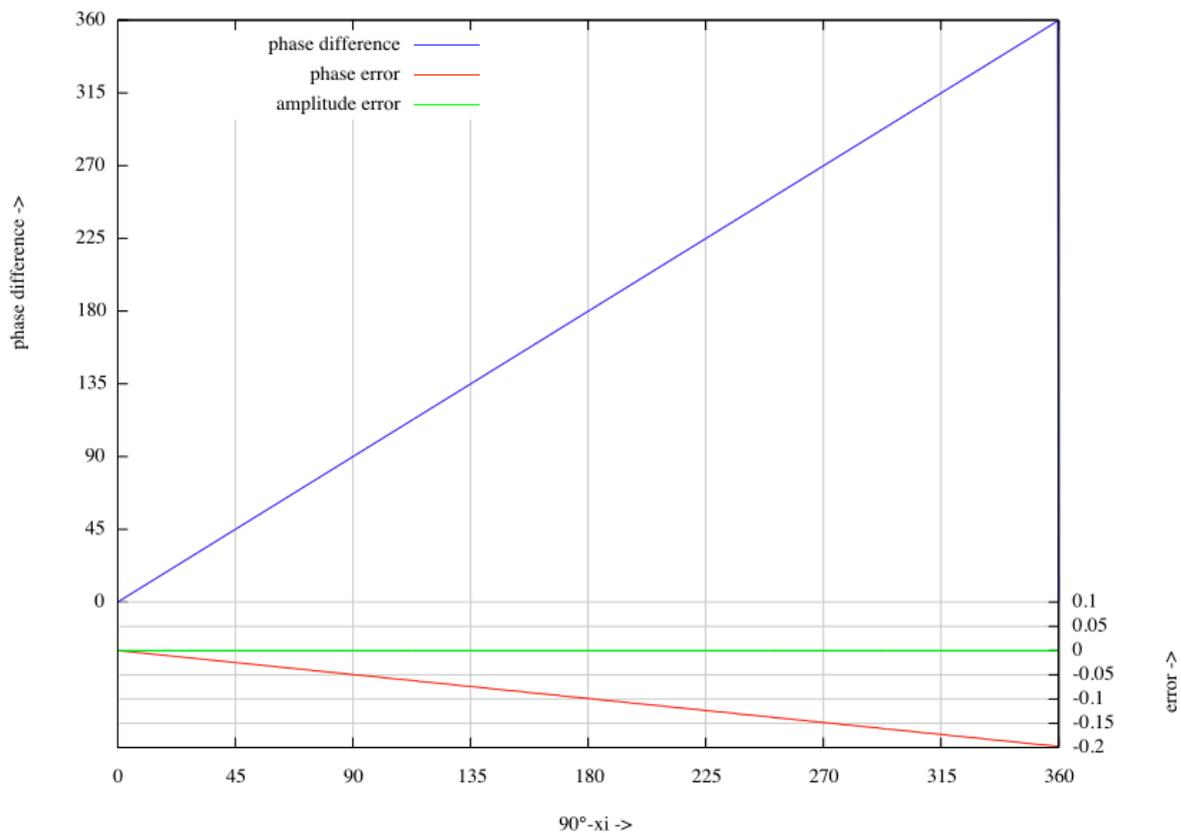


Figure 22: The amplitude and phase error of energy balanced oscillators for a constant radius. For each run, random initial conditions and random amplitudes are set. As expected, the amplitude error is zero, independently from the amplitudes and ξ . As explained in figure 9, the phase error is due to the relatively low precision (0.01 radians) used for detection of convergence.

Chapter 4

Co-evolution of configuration and control for locomotion

4.1 Introduction

In Chapter 3.1 I discussed the complexity of many DOF locomotion control. In summary, the oscillatory signals must be well coordinated for efficient locomotion and the effect of a single actuator on the global behavior cannot be easily predicted. Furthermore, the dynamics that emerge from the interplay between the CPG and the mechanical system must be taken into account. For all these reasons, it is very difficult to design a locomotion controller with traditional engineering methodologies (i.e. there is no analytical approach to the problem) - hence the need for heuristic optimization algorithms to set the parameters of the CPG.

In modular robotics, the arguments considered above also speak against designing the configurations of the robot by hand. There are two additional reasons why it is inadmissible to manually design the configurations and locomotion controllers: a) A modular robot is meant to operate in many different configurations. Manually designing all these configurations and their CPGs demands too much time and effort; b) The complexity of designing the configuration and the CPG grows exponentially with the number of modules used. Manually programming the locomotion controller is not scalable, it might become virtually impossible if several dozens or hundreds of modules are used.

In the next section I review various approaches to self-organization of locomotion in chain-type modular robots. Then I discuss previous research in the field of co-evolution of morphology and control. Finally I shall present how Adam (my modular robot simulation and evolution tool) co-evolves configurations and CPGs for locomoting chain-type modular robots. As you will see, the evolved configurations are more complex and the locomotion more elegant and efficient compared to what has been achieved in previous research.

4.1.1 Evolutionary approaches to modular robot control

As mentioned above, there has been a lot of research on automatic generation of biped or quadruped locomotion using various CPG models. Interestingly, to the best of my knowledge the research group behind the M-TRAN project and the BIRG laboratory are currently the only ones applying these principles to chain-type modular robots.

4.1.1.1 Evolutionary motion synthesis method for M-TRAN

An evolutionary motion synthesis method for M-TRAN that includes both locomotion and reconfiguration has been proposed in [Yoshida et al. 2003]. The behavior of the robot is described as a motion sequence. Each segment of the sequence contains various motor commands for the modules. There are two basic commands: One to set the angles of a specific module and one to keep a connection with another module. If not explicitly specified with the connection command, two neighboring modules are detached.

Yoshida et al. applied a Genetic Algorithm to evolve motion sequences for arbitrary generated initial configurations. With other words, they evolved gait control tables. Even though they argue that this is the first method that involves both locomotion and reconfiguration, the connection command was not used in the genetic algorithm. Therefore, only locomotion with static configurations was evolved. However, to evolve locomotion without reconfiguration other approaches are better suited (see below). This method only uses angles that are multiples of 30° and since it relies on a fix series of motor commands it is not scalable to evolve more complex behaviors (e.g. using sensors). Also it is not clear to me how reconfiguration could be evolved like this (e.g. what fitness to use). Furthermore control is centralized and the modules need unique identifiers.

4.1.1.2 Automatic locomotion pattern generation for M-TRAN

Kamimura et al. recently proposed an Automatic Locomotion Pattern Generation (ALPG) method that seems clearly superior to the evolutionary motion synthesis method described above [Kamimura et al. 2003]. Neural oscillators are used as a model of the CPG. Analogously to Mesot and Bourquin (see below) they apply a GA to optimize the parameters of the CPG for a specific, fixed configuration. The evolved gaits were successfully transferred from simulation to the M-TRAN hardware.

Kamimura et al. use a full coupling scheme, i.e. all oscillators are connected with each other unless the GA specifically sets the connection weights to zero. Clearly, this approach was only successful because the tested configurations were all quite simple, consisting of less than ten modules. With a full connection scheme, the number of couplings grows exponentially with the number of modules, thus it is not scalable. Furthermore, the principle of strictly local interaction is abandoned. For these reasons I believe that nearest neighbor couplings are better suited for modular robot CPGs.

Another weak point of the proposed method is, that the GA evolves the initial conditions of the oscillators. In other words, all initial conditions must be set to the evolved values before starting the controller. Thus, the locomotion controller is not asynchronous.

4.1.1.3 Previous research at BIRG

To explore self-organization of locomotion, Mesot simulated modules with stick-wheels that wander around and make connections with other modules that they bump into coincidentally [Mesot 2004]. This leads to random assemblies of modules. For a specific (fix) configuration, Mesot applied a GA to optimize the parameters of the nonlinear oscillators for locomotion. Furthermore he proposed an adaptive algorithm that applies gradient descent to find the coupling parameters for a specific phase difference. Note that if we know the phase predicting function, such an adaptive algorithm is not necessary anymore as we can directly set any desired phase difference between two oscillators. The most interesting part of Mesot's research concerns online optimization of locomotion using simulated annealing, which is discussed in chapter 5.

Bourquin applied different heuristic algorithms to optimize the locomotion of several fix configurations [Bourquin 2004]. Unlike Mesot, who used nearest neighbor couplings for the nonlinear oscillators, the coupling scheme was manually designed. Efficient locomotion gaits were generated by all tested optimization algorithms (particle swarm optimization, genetic algorithm, simulated annealing), even though particle swarm optimization seemed to perform slightly better than the others.

4.1.2 Co-evolution of morphology and control

Even though to the best of my knowledge Adam is the only project co-evolving configuration and control of modular robots, co-evolutionary algorithms have successfully been applied in other fields.

Sims co-evolved body and brain for locomoting and competing block creatures [Sims 1994]. Ventrella used co-evolution to generate walking stick creatures [Ventrella 1994]. With Framsticks, an artificial life project, the user can co-evolve morphology and control of virtual stick creatures with various genotypes and fitness functions [Komosinsky & Rotaru-Varga 2001]. Bongard and Pfeifer evolve gene expression rules and simulate an ontogenetic process during which the body grows by successive cell divisions [Bongard & Pfeiffer 2001]. Hornby co-evolved morphology and control of locomoting robots as illustrated on figure 23 [Hornby & Pollack 2001]. Using L-systems as generative encoding

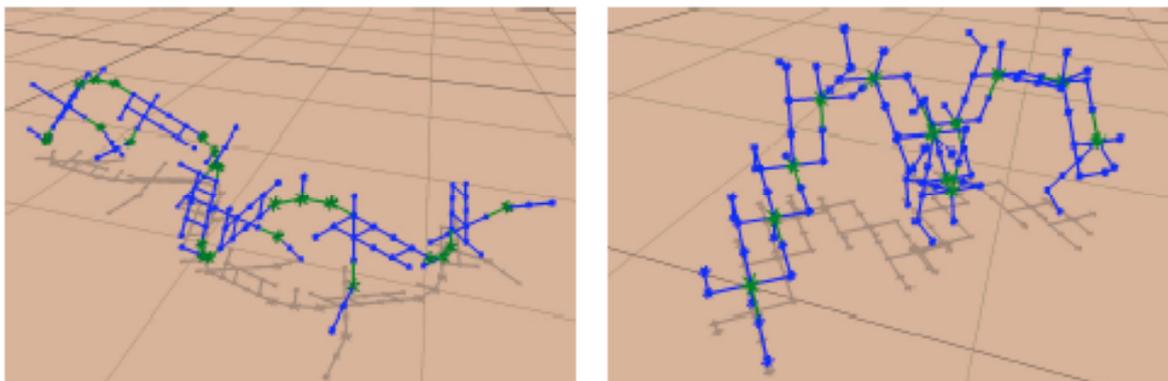


Figure 23: Using L-systems as generative encoding, Hornby co-evolved morphology and control of locomoting robots that are more complex than those in previous work.

for the GA he succeeded evolving robots that are more complex and use more parts than those in previous work.

However, the projects mentioned above all use stick-like body-segments that are themselves subject to evolution and don't simulate existing hardware. By using a realistic simulation, the evolved solutions can be built in reality as done by Hornby or even be generated automatically with 3D printing technology [Lipson & Pollack 2000]. However, the modules must still be produced especially for every robot. In contrast, robots evolved by Adam can be assembled and tested with the predefined module type (YaMoR). Similar work has been done with LEGO robots [e.g. Lund 2001], but these robots use many different kinds of elements and are not well suited to implement key features of modular robots like self-reconfiguration.

4.2 The simulation environment

Before discussing the genetic algorithm used by Adam to co-evolve configuration and control of modular robots, I shall briefly present the simulation environment that the robots are tested in. For the implementation I tried out two different frameworks: The Open Dynamics Engine (ODE) and Webots.

ODE is an open source physics simulation library [Smith]. It accurately models rigid body dynamics (kinematics, gravity, friction, collisions etc.). ODE is well suited for evolutionary robotics because it is very stable and fast. Webots [Michel 2004], commercial robot simulation software developed by Cyberbotics Ltd in collaboration with EPFL, is actually built on top of ODE. In contrast to ODE it is a higher-level framework specifically designed for robot simulation. The user can build robots and environments with a graphical user interface. The advantages of Webots are numerous: It is possible to design a robot simulation in short time, without worrying about the details of the lower-level physics simulation library. Furthermore, common types of sensors and actuators are available and the user can benefit from many additional features such as video export. The disadvantage of Webots is a certain loss of flexibility that is inherent to any high-level software framework. For this reason and also because of problems with the Mac version of Webots, I decided to use ODE for the simulation.

The simulated module corresponds exactly to the specifications of the real hardware (see chapter 2.2.4). As you can see on figure 24, I left aside the cylindrical front part of the module in the ODE simulation. I did so to increase the speed and stability of the simulation (cylinders take considerably more time than cubes for collision detection and they sometimes cause stability problems in ODE). Even though there are few collisions with the cylindrical front part as the lever protects it, we should add it to the simulation before transferring evolved robots to the hardware. The total mass of the module is uniformly distributed in the simulated bodies. Again, before transfer to the real world one should implement a more detailed model.

Powered hinge joints are used to simulate the RC-servo of YaMoR. A PD controller sets the torques that are necessary to follow the trajectories generated by the CPG. The

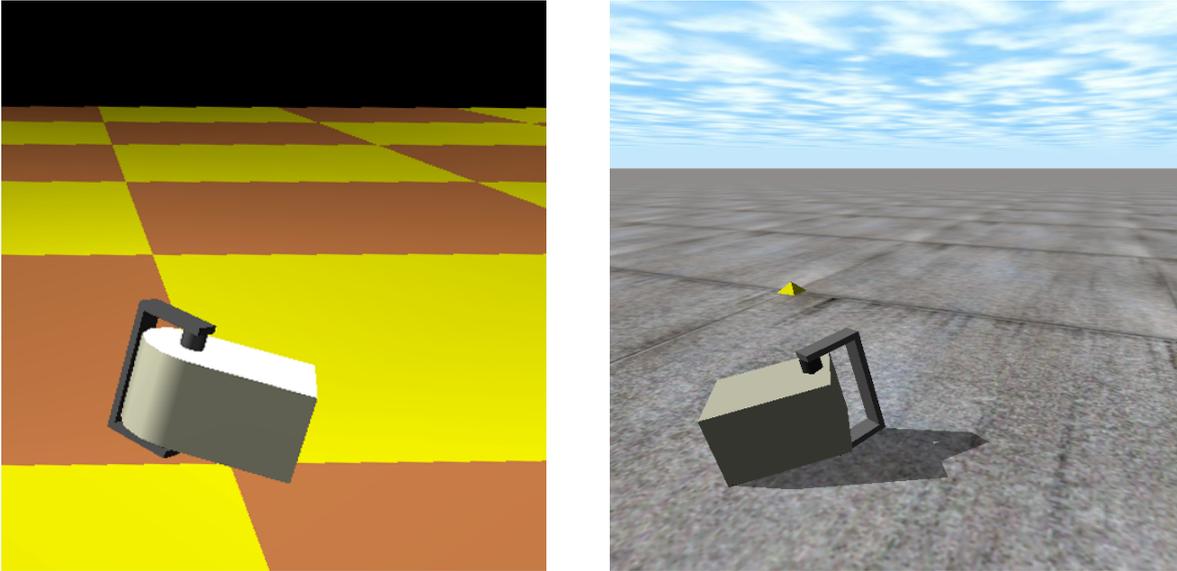


Figure 24: Simulation of the YaMoR module with Webots (left) and ODE (right).
The environment is for the moment an infinite plane.

torque T_i that is applied to the hinge i depends on the desired angle θ_i , the actual angle θ_a , and the angular rate ω_a :

$$T_i = \alpha(\theta_i - \theta_a) - \beta\omega_a$$

Experience showed, that it is better to use a more advanced controller that is part of ODE. The input for this controller is the target angular rate ω_d .

$$\omega_d = \alpha(\theta_i - \theta_a)$$

Again, α is a positive constant that must be determined experimentally. I agree with Mesot [Mesot 2004] that there is no need for a second term that depends on the actual angular rate ω_a (like in the first equation) because the same effect can be achieved with the parameters of the ODE controller. Most importantly, these parameters can also be used to set the maximum torque and the maximum speed of the powered hinge to respect the specifications of YaMoR's RC-servo.

4.3 Genetic Algorithm

For the reader who is not familiar with genetic algorithms, I give here a very brief introduction. A genetic algorithm (GA) acts on a population of individuals. In this case, the individuals are modular robots. In analogy to Darwinian evolution in nature, fitter individuals have a higher probability to reproduce and a lower probability to die than weaker individuals. As in nature, the offspring is affected by mutations and/or crossover. The fitness of an individual is given by the fitness function. An example of a simple fitness function that promotes locomotion is the average speed of the individual over a certain

amount of time. An individual is defined by its genome. In order to determine the fitness of an individual its genome must first be decoded. The function that decodes the genome is called the genotype-to-phenotype mapping. In the case of Adam, the phenotypes are modular robots in the simulation world. The set of all possible genotypes and phenotypes are called the genotype and phenotype space of the GA.

4.3.1 Genetic encoding

Defining an adequate genotype-to-phenotype mapping is by far the most difficult step when designing a GA that acts on a complex phenotype space. The most common genetic encodings for co-evolution of morphology and control are directed graphs and L-systems. Directed graphs are direct encodings, generally the nodes represent body segments and the links physical connections. On the other hand, L-systems are generative (or developmental) encodings that describe a growing process of the morphology. They result in more complex and better structured, but not necessarily fitter solutions. For Adam I chose trees, a subset of directed graphs, as genotype.

The genetic encoding defines which phenotypes are close genetically (i.e. separated by only a few mutations) thereby structuring the phenotype space. The fitness function induces another topology with respect to fitness values (sometimes called the fitness landscape). The idea behind direct encodings is to correlate these two topologies. In other words, related genotypes should correspond to similar phenotypes with similar fitness values. The genotype that is presented next strictly follows this philosophy. In contrast, the second genotype (section 4.3.1.3), which encodes symmetric configurations, is a compromise between a developmental and a direct encoding.

4.3.1.1 Encoding the configuration of YaMoR robots

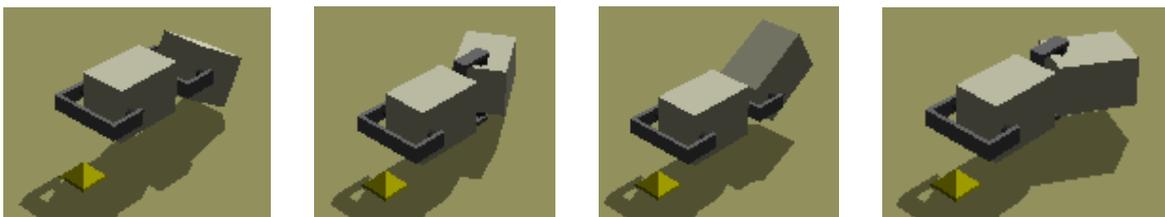
The Adam genotype is a tree. Each node represents a module and links correspond to physical connections between the modules. The root of the tree is referred to as the head of the robot. Even though YaMoR has a genderless connection mechanism, I use a male/female connection scheme for the genotype: The docking plate on the U-shaped lever is male and the docking plates of the body are female. The disadvantage of this choice is a smaller phenotype space, i.e. not all configurations that are physically possible can be encoded with the genotype. The advantages are:

- Children are always attached to their parents with the lever, thus the only free lever in the configuration is the one of the head. In other words, a limb always ends with a body of a module and not a lever. This is desirable because the body is more robust and provides more friction with the ground. Clearly, the lever is not well suited as a foot, especially in unstructured terrain where it could get stuck easily.
- The control algorithm is simplified because the modules always know that their parent is connected at the lever and all other attached modules are children. As I will explain below, modules need to know who their parent is to correctly communicate the state of their nonlinear oscillator.

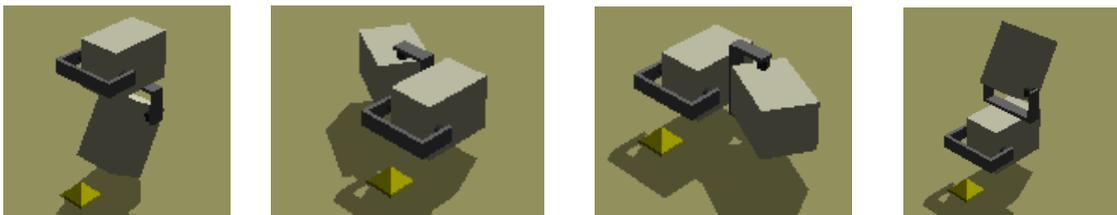
- No automatic docking mechanism has yet been designed for YaMoR. Promising a genderless connection mechanism is always easier than implementing it. If my evolutionary experiments show that the locomotion capability is not impaired by the male/female connections, we can do without a genderless docking mechanism in the hardware.
- The implementation of the genotype-to-phenotype mapping is simplified. Decoding the tree genome and building the corresponding modular robot in the simulation world is highly complex. Despite my previous experience, this part of the project took about two months.
- The genetic algorithm benefits from the smaller phenotype space so that evolution might be faster.

Even though the phenotype space has been restricted, it is still huge. As illustrated on figure 25, two modules can be attached together in twenty different ways: There are four distinct orientations for each of the five docking positions. Using three modules, there are already 720 different configurations (even though some of them might be functionally equivalent). When more than ten modules are involved, the number of configurations quickly grows beyond all measures. Considering this big phenotype space, I don't believe that using male/female connectors limits the locomotion capabilities of evolved robots.

Besides from the orientations and the docking positions, the initial angles of the hinge joints are also an important parameter of the configuration. Powered modules oscillate around the initial angle and the joints of rigid modules are locked at the initial angle. Mesot and Bourquin consider the initial angle as a parameter of the controller, which works fine for the specific configurations that they worked with. However, if one is confronted with randomly generated configurations, the initial angles must be taken into account before starting the simulation in order to determine if the structure is legal or not. I define a structure to be illegal if it has intersecting body parts. The initial angles significantly alter the shape of large configurations. If zero initial angles were used when building the



a) A child (the module with an initial angle of 30 degrees) can be connected with the parent in four distinct orientations, labeled north, east, south and west (from left to right).



b) The child can be attached to the parent at five different connection plates. They are labeled BACK (illustrated in the upper four figures), DOWN, LEFT, RIGHT and UP (from left to right).

Figure 25: The genotype limits the configurations of two modules to four orientations (a) and five connection plates (b). Therefore, two modules can be attached together in twenty different ways.

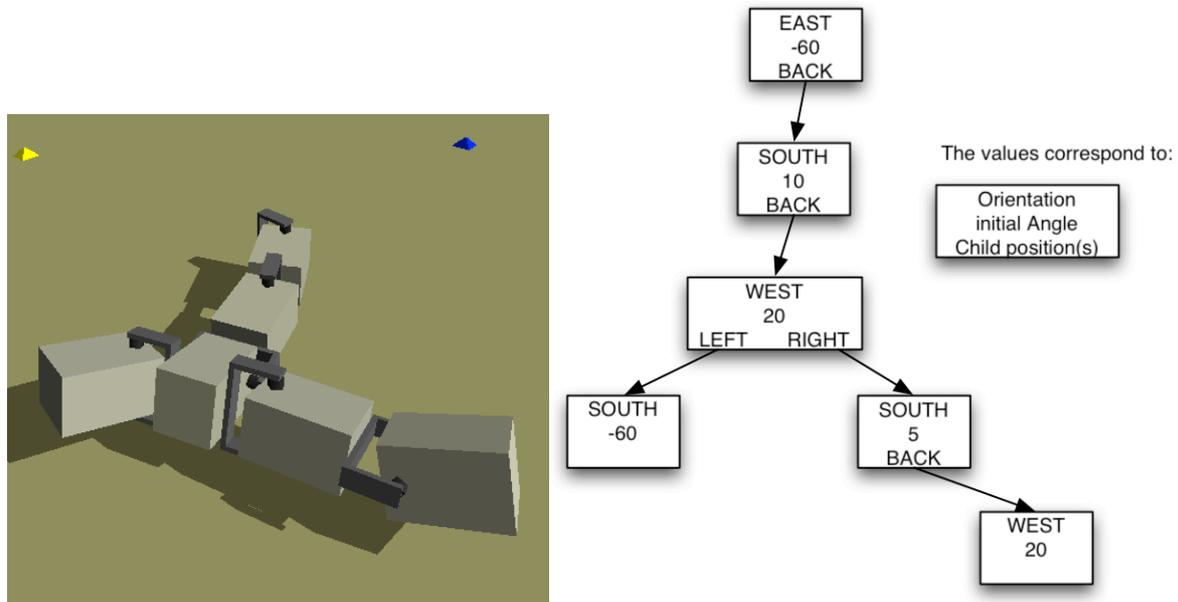


Figure 26: The configuration in the simulation world (phenotype) and the corresponding genotype. The head of this structure (on the top in both figures) has an initial angle of -60° , its child is attached to the BACK position with orientation SOUTH.

structure, a configuration might be considered legal even if it is problematic with the true initial angles. Or it might be discarded because of intersecting body parts but with the true initial angles the problem would not have occurred.

In figure 26, you can see a simple configuration and its genotype. Each node encodes its orientation and initial angle. Furthermore, each node specifies at which positions the children are attached. Note that these positions must not be conflicting, i.e. there can't be two children attached at the same position. I refer to these variables as structural parameters; they are summarized in table 1.

I conclude this section with some implementation details of the genotype-to-phenotype mapping regarding the configuration. In order to set the correct position of the various bodies and joints in the physics simulation, one must know the coordinate transformation from the global (world) coordinate system to the local coordinate system of each module. The procedure starts with the head and recursively descends the tree. The coordinate transformation of any module can be calculated with a series of matrix multiplications of the parents coordinate transformation matrix. The matrix multiplications must take into account the initial angle of the parent as well as the docking position and the orientation of the child. Once the structure is built, it is tested for intersecting body parts with ODE's

Parameter	Range	Description
Orientation	{NORTH, EAST, SOUTH, WEST}	The orientation of the module
Initial angle	$(-\pi/2, \pi/2]$	The initial angle of the hinge joint.
Child position(s)	{BACK, LEFT, RIGHT, UP, DOWN}	The docking position for every child

Table 1: The structural parameters. Each node contains at least it's orientation and initial angle. Additionally, it must encode the docking position of each child. I use the term structural parameters for these variables because they encode properties of the structure. On the other hand, the control parameters (tables 2 and 3) encode properties of the controller.

built in collision detection. Furthermore, the height difference between the lowest module and the head is determined. The whole structure is then destroyed and, if it was legal, rebuilt at the correct height in the real simulation world.

4.3.1.2 Encoding the controllers

Luckily, encoding the controllers is much simpler than encoding the configurations. The parameters of a module's controller must be encoded in the respective node together with the structural parameters (orientation, child position(s) and initial angle). This is because structural mutations, for example attaching a limb (i.e. a sub-tree) at a new position, should not affect the controllers. In other words, the nodes encapsulate the controller of the module and protect it from mutations that affect the structure.

Encoding harmonic oscillators is straightforward. The free parameters defined in table 1 are simply added to the structural parameters of the nodes. The oscillation will take place around the initial angle defined in the structural parameters.

The approach is the same when using nonlinear oscillators. The control parameters (table 2) are encoded in the nodes, together with the orientation, initial angle and child position(s) of the module. However, the couplings require some special attention. Kamimura et al. use a full coupling scheme for their neural oscillators [Kamimura et al. 2003], i.e. every oscillator is coupled with all other oscillators unless the weights of some connections are explicitly set to zero. In section 4.1.1 I argued that this approach is not scalable and that the principle of strictly local interaction shouldn't be abandoned. For these reasons I use nearest neighbor couplings as illustrated in figure 27. The children are coupled with their parents. A rigid module has no active oscillator, thus it simply relays the couplings from its children to the parent. The head must always have an active oscillator (even if it is set rigid) to ensure that the network of coupled oscillators is a spanning tree over the whole configuration.

Parameter	Range	Description
IS_RIGID	{true, false}	Determines if the module is rigid or not.
A	$(0, \pi/2]$	The amplitude.
phi	$[0, 2\pi]$	The phase.

Table 1: Control parameters of a harmonic oscillator. The frequency is constant and the same for all modules. The oscillation takes place around the initial angle defined in the structural parameters.

Parameter	Range	Description
IS_RIGID	{true, false}	Determines if the module is rigid or not.
E	$(0, \pi/4]$	The energy parameter of the nonlinear oscillator.
a _{ij}	$[-2, 2]$	The weights of the coupling from the parent to this oscillator.
b _{ij}	$[-2, 2]$	
a _{ji}	$[-2, 2]$	The weights of the coupling from this oscillator to the parent (only for bidirectional couplings with four free parameters).
b _{ji}	$[-2, 2]$	

Table 2: Free parameters of a coupled nonlinear oscillator. The last two parameters are only required when using bidirectional couplings with four free parameters.

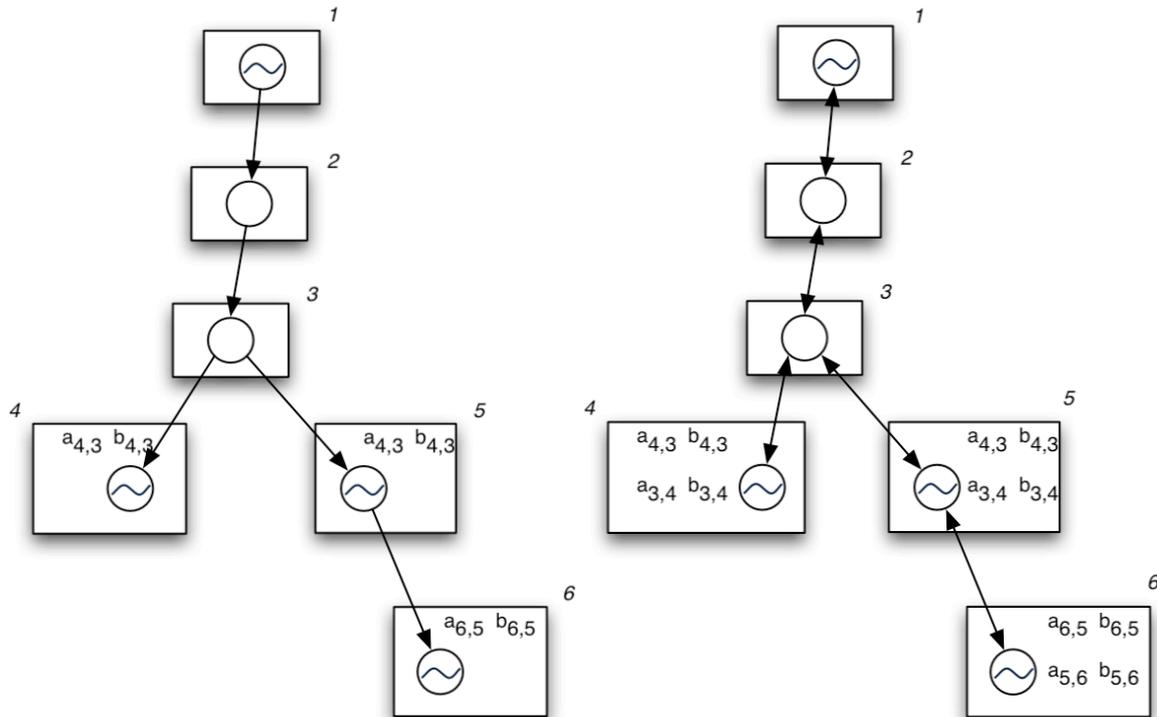


Figure 27: Nearest neighbor couplings for the example configuration of figure 26. The oscillators of rigid modules are inactive (represented with an empty circle). Rigid modules relay the couplings between their parents and children as described below. Unidirectional couplings are used on the left, bidirectional couplings on the right. The child nodes encapsulate the coupling parameters a_{ij} and b_{ij} of the coupling with their parent.

The parameters a_{ij} and b_{ij} of a coupling between a parent and a child module are encoded in the child. Thus, if a structural mutation detaches a sub-tree (i.e. a limb) from its previous position to connect it to a new module, the limb still operates with the same phase difference. For example, if the sub-tree of module 5 in figure 27 is detached from module 3 and reattached to the head, it still has the same phase difference because the node encapsulates the coupling parameters.

4.3.1.3 A genetic encoding for symmetric configurations

Robots that have been evolved with the genotype presented above are often quite symmetric. It seems that the phenotype space is too big and unstructured for the GA to find truly symmetric configurations that would perform even better. In chapter 3.1.3 I reasoned about symmetry in nature and concluded that symmetry might be a fundamental characteristic of efficient locomotion. For all these reasons it is interesting to restrict the phenotypes to symmetric configurations.

The basic idea is to mirror the configuration along the spine. The spine is the robots axis of symmetry (refer to figure 28). The symmetry is not encoded in the genotype, the only difference with the genotype presented above is that each node must not only encode its own phase, but also the phase of its mirrored counterpart. As in nature, the amplitudes of corresponding joints in left and right limbs are the same. Thus, an additional phase parameter must be added for harmonic oscillators and an additional set of coupling parameters when nonlinear oscillators are used. Refer to figure 29.

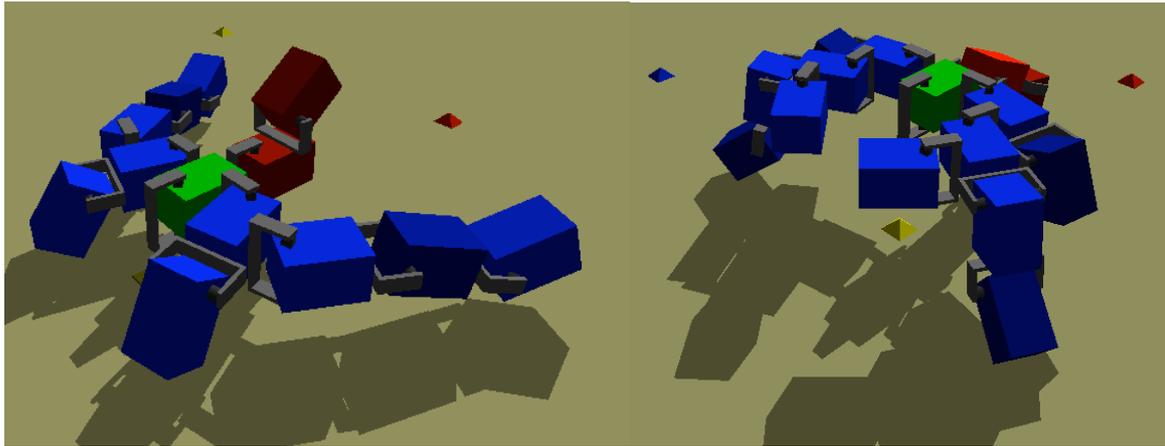


Figure 28: Two robots that have been evolved with the symmetric encoding. The head is colored green, spine modules red and limbs blue.

First of all, the genotype-to-phenotype mapping detects which modules lie in the axis of symmetry. These are the spine modules. If the initial angle of a spine module would cause the spine to be asymmetric, it must be overruled and set to zero. For example, the left robot of figure 28 must have a zero angle for the first spine module, but not for the second one.

In a second step, the limbs on the right are mirrored to the left and vice-versa. The mirrored limb has the same tree structure as the original, but some of the structural parameters must be mirrored in some cases. For example, depending on the relative position of the module within the limb, the initial angle must sometimes be multiplied by -1 , sometimes not. Analogously, the child positions must sometimes be switched. A detailed description of the procedure goes beyond the scope of this report.

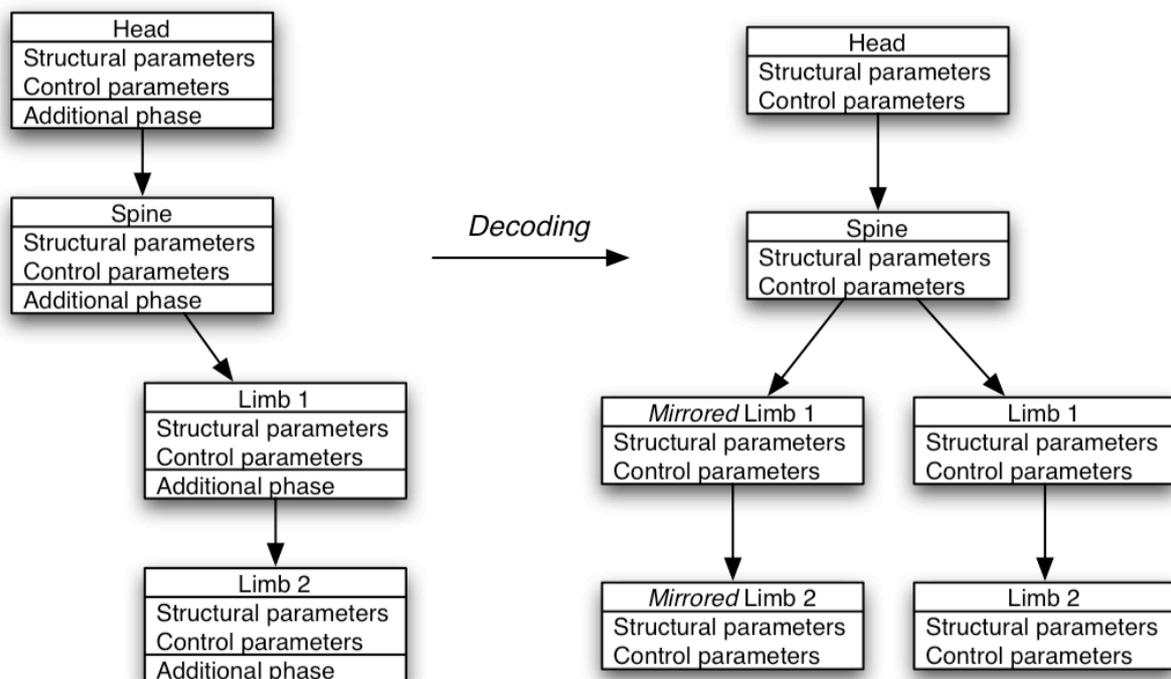


Figure 29: Genotype for symmetric configurations. On the left: An example genome. The phase of the mirrored module is added to the parameters defined in the previous sections. On the right: The same genome after the symmetric decoding. The limb has been mirrored and the additional phase integrated in the controllers of the mirrored modules.

4.3.2 Genetic operators

Genetic operators include mutation and crossover. Both operators act on the structure as well as the control parameters. Mutation of real-valued parameters is done by adding a random number from a Gaussian distribution to the previous value. The mean of the Gaussian distribution is zero, the standard deviation σ depends on the parameters range:

$$\sigma = \frac{x_{\max} - x_{\min}}{k}$$

where k is a positive integer, x_{\max} is the upper and x_{\min} the lower bound of the parameter. k is a parameter of the GA that must be set intuitively. I found that values between 10 and 60 gave good results.

Parameters that are not real numbers, namely the orientation, the child positions and the Boolean variable that specifies if the module is rigid, are mutated by randomly selecting another value from the set of possible values.

The tree structure of the genotype may be mutated in several ways. New modules can be added to the structure at any free face. Modules or whole sub-trees can also be deleted by mutation. Furthermore, sub-trees or single nodes of the tree may be swapped.

I use single-point crossover. A child is formed by copying the mother and then replacing one of its sub trees with a sub tree of the father. Generally this corresponds to a macro-mutation, i.e. the child is not close to the parents in the phenotype space. This is important to explore new areas of the phenotype space but also problematic because the area around the parents might not be searched enough. Thus it is advisable to use rather low probabilities for the crossover operator.

Before doing crossover I check if the tree of the mother's and the father's genotype are identical. When comparing the trees, the parameter values are ignored. Only the structure of the tree is considered, i.e. the nodes and the links that connect them. Note that identical trees may correspond to different (but similar) configurations because the nodes encapsulate the structural parameters. If the trees of the mother and the father are identical, the crossover operator swaps identical sub trees. In other words, the child has the same tree structure as the parents, but part of the tree is from the mother and the other part from the father. This corresponds better to sexual reproduction in nature: The child is similar to the parents and inherits qualities from both parts. As the GA converges to a particular solution, the population becomes more and more homogenous and the probability for the tree structure of the parents to be identical becomes quite high. In advanced generations, approximately half of the individuals have the same tree structure.

4.3.3 Fitness function

I reward locomotion with a very simple but effective fitness function. The fitness of a robot is determined by how far it travels from the starting point in a certain amount of time. Note that this is not the total distance that the robot traveled. This simple fitness function implicitly promotes locomotion in a straight line. I didn't get better results with more complex fitness functions that explicitly reward movement in a straight line.

The time of simulation is crucial to achieve good results. The principle is „you get what you ask for“. If the time of simulation used for fitness evaluation is too short, one can't expect stable locomotion of evolved robots after that exact amount of time. A common problem is also the evolution of high, instable configurations that get a positive fitness score just by falling over at the beginning of the simulation.

I found that letting the simulation run a certain amount of time before starting the fitness evaluation solved these problems. Generally I used the position of the robot after 4 simulated seconds as starting point and the position after 14 seconds as end point to measure the distance.

4.3.4 Initialization and detection of convergence

The population of the GA is initialized with randomly generated genomes. I use a maximum depth of 5 for the initialization of the tree genomes. Note that during the evolution, the size of the genome is unlimited.

I use a convergence test to decide when to stop the GA. The convergence of the GA at generation g is defined as the ratio of the N th previous best-of-generation fitness $f_{\max}(g - N)$ to the current best-of-generation score $f_{\max}(g)$:

$$c(g) = \frac{f_{\max}(g)}{f_{\max}(g - N)}$$

where N is a positive integer. The GA is stopped when the convergence of the current generation passes a user specified bound, for example if: $c(g) > 0.95$.

N should be set in accordance with the type of genetic algorithm that is used. One should choose high values for N when using an incremental GA, where at each generation only two new individuals are produced. In contrast, N should be set to a rather low value for a GA with multiple, migrating populations where hundreds of new individuals are evaluated at every generation.

When comparing the performance of different types of GAs (chapter 6) I chose N such that 1000 individuals are evaluated in N generations. For example, an incremental GA has $N = 500$ because two new robots are evaluated in one generation.

4.3.5 Selection and replacement

I shall now review the different types of genetic algorithms that I tested. All algorithms use the same genetic encoding, genetic operators, fitness function, initialization and stopping procedure. The algorithms only differ in how they manage the population, namely how offspring is created and inserted into the population.

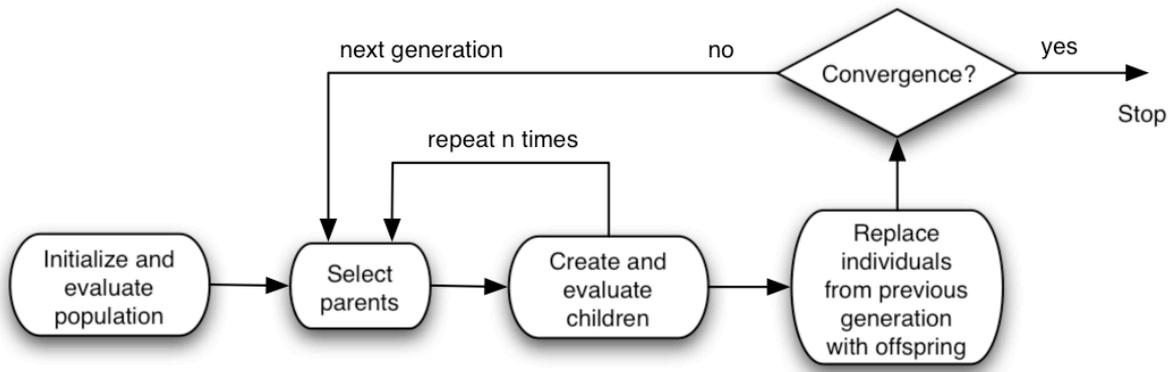


Figure 30: Flow chart of a genetic algorithm. In contrast to the steady state GA, the incremental GA does not repeat the selection of parents and creation of children in a single generation.

The incremental GA only creates two new individuals per generation. For each generation, two children are generated from two selected parents. The children then replace two individuals from the previous generation. Refer to figure 30. I tested two different replacement strategies. The first one always replaces the worst individuals of the population. If the newly generated offspring has a lower fitness value than the worst individual, it is not inserted into the population. In the hope to promote diversity in the population I also tested a random replacement method, where the children replace randomly selected individuals. The best robot is never removed (elitism).

I obtained best results with a rank-proportional roulette wheel method for selecting parents. The robot with the highest fitness value has rank zero, the worst robot has rank $N-1$, where N is the population size. The probability $p_s(i)$ for an individual i to be selected is inverse proportional to its rank $r(i)$:

$$p_s(i) = \frac{N - r(i)}{\sum_i (r(i) + 1)}$$

The steady state GA follows the same procedure as the incremental GA, but much more offspring is generated at each iteration. As you can see in figure 30, two children are not directly inserted into the population. Replacement only takes place after all offspring of the generation has been created. The disadvantage of this approach with respect to the incremental GA is that the convergence is slowed down. On the other hand, the slower convergence allows the GA to better explore the search space and potentially evolve fitter individuals.

Finally I also tested a GA with multiple, independent populations. Each population is evolved using a steady state GA as described above. At each generation, the populations migrate a fixed number of their best individuals to the neighbor population (deterministic stepping-stone migration algorithm). Additionally, a master population is updated each generation with the best individuals from every population. The performance of the incremental GA, steady state GA and the algorithm with migrating populations is compared in chapter 6.

4.4 Asynchronous distributed control

In chapter 1 I have discussed the importance of asynchronous distributed control in modular robotics. Hormone-based control (section 2.3.3) and role-based control (section 2.3.2) are examples of distributed asynchronous control algorithms. I shall now briefly outline such an algorithm for modular robots that have been evolved with the GA presented in the previous section. With minor adaptations, the algorithm could be implemented on any modular robot that uses coupled nonlinear oscillators with nearest neighbor couplings.

The modules are autonomous, i.e. they have their own computational resource. Furthermore, they must be able to communicate with the modules that they are connected with (the nearest neighbors). It is assumed that the modules know the genome and their position within the configuration. Thus, they can extract the appropriate genes from the genome to correctly set the energy and coupling parameters of their nonlinear oscillator.

If unidirectional couplings are used, each module periodically communicates its state x_i and v_i to all children. The children compute the perturbation of their nonlinear oscillator based on the state of the parents and the coupling parameters that are encoded in their genome. Rigid modules simply forward the state information from their parent to all children.

In the case of bidirectional couplings, in addition to the communication from parents to children, the children must also send their state to the parent. Actually, since all parameters of the bidirectional coupling are encoded in the children genome, the child j may directly send its contribution s_j to the perturbation of the parent oscillator i .

$$s_j = \frac{a_{ij}x_j + b_{ij}v_j}{\sqrt{x_j^2 + v_j^2}}$$

Again, rigid modules relay the s_j from the children to the parents. In order to reduce the communication load, the rigid module may already compute the sum of the s_j from its children and only send this sum to the parent node. Now suppose that the nonlinear oscillators have converged to constant phase differences. If energy balanced couplings are used, the perturbations are all zero at that point (remember that the perturbation represents the phase error). In other words, the couplings have no effect anymore. Thus, if one or several couplings are temporarily cut because of a communication failure, the behavior of the robot is not affected. Actually, in the absence of sensory feedback or higher-level control, the couplings could be completely disabled after convergence of the nonlinear oscillators. The phase differences would diverge only very slowly because of numerical errors (I have tested this in simulation).

If a module doesn't get a message from a neighbor in time, it assumes a communication failure and integrates the nonlinear oscillator without the input from that neighbor. With standard nonlinear oscillators, this would cause the modules amplitude to drop significantly. But when using energy-balanced couplings, the modules continue to operate with the same phase differences and the same amplitudes. Thus, the control algorithm is resistant to temporary communication failures.

In summary, I have presented an approach for a reliable, asynchronous, distributed control algorithm that could be implemented in the YaMoR modules or other modular

robots. In contrast to hormone-based and role-based control, neighboring modules *smoothly* synchronize to the desired phase difference.

Chapter 5

Online optimization of locomotion

The genetic algorithm presented above is an offline optimization method. Only if the modules could autonomously self-assemble to the evolved configurations and evaluate the fitness of the robot, the GA could potentially be implemented online, i.e. on the hardware itself and not just in simulation. Clearly, this is not possible with today's technology. Furthermore, the evolution of fit individuals would take years.

Online optimization (we could also say learning or online adaptation) of modular robot locomotion is interesting for two scenarios. The first scenario has been considered by Mesot [Mesot 2004]: Self-assembling modules build random configurations. Using an online optimization algorithm, locomotion can be learned by such random structures. Even though highly interesting from a theoretical point of view, this scenario is not very practical. Obviously, random configurations are almost never well suited for locomotion. Therefore, even if the online optimization method finds the global optimum (i.e. the perfect locomotion gait for this configuration) the robot will still perform poorly compared to robots with configurations that have been designed for locomotion.

The second, more practical scenario is the online adaptation of a specific locomotion gait of a modular robot. For example, when building a robot that has been evolved in simulation, it always performs worse in reality than in simulation. This is because the simulation is not an exact model of the real world. Learning robots favor a smooth transfer from simulation to reality because the locomotion gait is being adapted to the environmental constraints by the online optimization algorithm. This is useful not only for transfer from simulation to reality, but whenever the robot needs to adapt to new environmental conditions.

To the best of my knowledge, Mesot is the first one to tackle the problem of online optimization of locomotion for chain-type modular robots [Mesot 2004]. He uses simulated annealing to optimize the parameters of the controllers. His approach is discussed in the next section. The algorithm I chose for online optimization is Powell's method. It is based on a one-dimensional optimization procedure like Brent's method. Unfortunately the implementation is not finished at the time of writing. Numerical recipes in C gives an excellent overview of both methods:

<http://www.library.cornell.edu/nr/bookcpdf.html>

5.1 Introduction to optimization

The goal of optimization is to find an extremum (maximum or minimum point) of a function f that depends on one or several variables. An extremum can be global or local. A local extremum is an optimum in a finite neighborhood. The global extremum is truly the highest or lowest function value. From now on I just consider the problem of function maximization. Note that function minimization is trivially related because one can maximize $-f$.

In our case, f is the fitness function and the variables are the parameters of the controller. The fitness function is evaluated by setting the new control parameters, waiting until the nonlinear oscillators converged and then measuring the average speed or the distance covered over a certain amount of time. For now I just assume that the robot can estimate its speed, even though YaMoR is not yet equipped with the necessary sensors.

There are two classes of optimization algorithms: Those that only require evaluations of the function and those that also require evaluations of the derivative (the gradient in the multidimensional case). Unfortunately we don't know the gradient of the fitness function f . Even though methods using the gradient are more powerful, numerically estimating the gradient of the f would take too much time because it involves many evaluations of f .

The most obvious way to optimize f is to crawl uphill on the fitness landscape. Indeed, there is a simple but very effective algorithm that follows this approach: The *downhill simplex method*. This algorithm is not well suited for online optimization of locomotion because it can be extremely slow. Other multidimensional optimization methods that do without the gradient are *heuristic optimization methods* (e.g. simulated annealing), and the so-called *direction-set methods*.

Mesot used simulated annealing for online optimization [Mesot 2004]. On the heart of simulated annealing [Kirkpatrick 1983] lies an analogy to how crystals are obtained from a melt. If the temperature in a melt is gradually reduced crystals form. In a crystal, billions of atoms are ordered in a completely regular structure that corresponds to the lowest possible energy state of the system, i.e. the global optimum. In simulated annealing, the atoms correspond to the parameters, and the energy to the function that is being optimized. The essence is that the system may always jump to a state of higher energy (i.e. lower fitness). Thus, the method heuristically explores the search space and does not always crawl uphill. As the temperature (a parameter of the algorithm) is decreased, the probability that the system jumps to a higher state of energy becomes smaller and smaller.

In summary, simulated annealing is a heuristic optimization method that avoids small local optima if the temperature is decreased slowly. Unfortunately, the optimization process takes so long (hundreds of simulated hours in Mesot's experiments) that the 'online' optimization could never be used online in reality. Two other heuristic optimization algorithms are Particle Swarm Optimization (PSO) and genetic algorithms. Bourquin obtained better results with these two algorithms than with simulated annealing for offline optimization [Bourquin 2004]. This indicates, that GAs or PSO probably also perform better for online optimization. Robot controllers for wheeled robots have successfully been evolved online with genetic algorithms but the optimization took days.

By using a quadratically converging direction-set method instead of a heuristic search algorithm I take a radically different approach. For the second scenario mentioned above,

which is online adaptation of locomotion, a heuristic optimization algorithm is clearly not appropriate because the gait that is being adapted is relatively close to the optimum that we are looking for. Therefore it is unnecessary to heuristically explore wide areas of the search space. For the online optimization of locomotion from scratch (first scenario mentioned above) the performance of a deterministic optimization method depends heavily on the topology of the fitness landscape. If the search space is full of small, local optima, the probability of finding a good locomotion gait is small. Only experiments can show if my approach is well suited for online optimization from scratch, but I am confident that it is the right choice for online adaptation of locomotion.

Chapter 6

Results

6.1 Fitness function

First of all, I would like to illustrate the problems that arise when using an evaluation time that is too short. Some genetic algorithms evolve high and instable structures that get a positive fitness just by falling over at the beginning of the simulation (figure 31). Other robots take advantage of their initial position, i.e. the specific position where the structure

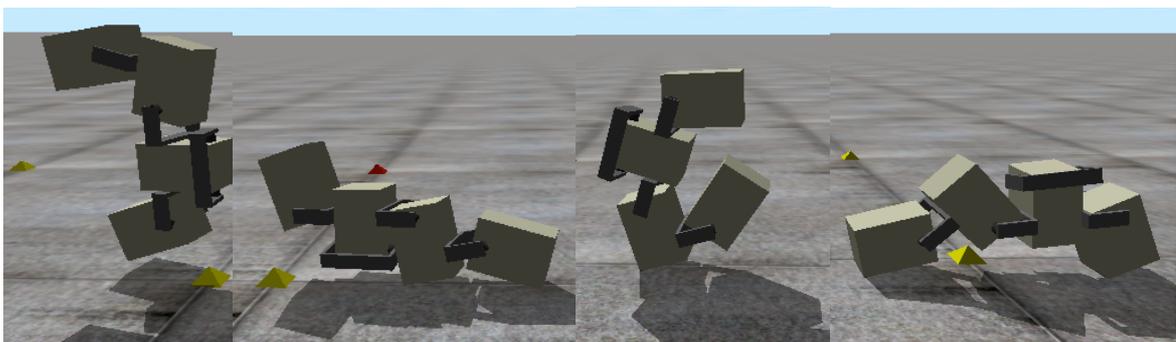
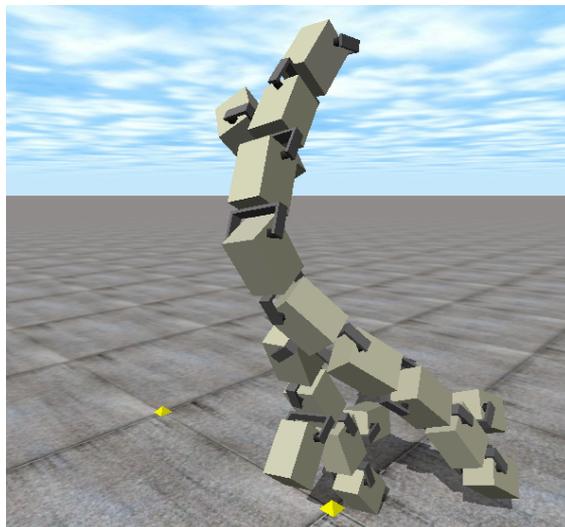


Figure 31: The structure on the upper figure gets a positive fitness just by falling over at the beginning of the simulation. The second robot actively takes advantage of the momentum he gets from rolling out of the initial position. Both behaviors can be avoided by increasing the evaluation time

is built at the beginning of the simulation. The robot in figure 31 takes advantage of the initial position to roll forward one time. In contrast to the big structure that gets fitness just by falling, this robot continues to locomote after the initial forward roll. When the evaluation time is too short, almost all robots take advantage of their potential kinematical energy from the initial position.

To avoid these problems I increased the evaluation time from 6 to 14 simulated seconds. Furthermore, I started the evaluation only after 4 seconds. Therefore, robots are not encouraged to take advantage of their initial position. Unfortunately, increasing the evaluation time considerably slows down the genetic algorithm but I do not see an alternative.

An interesting detail can be observed on the big structure of figure 31. The two limbs that touch ground are almost identical. Even though there is no gene duplication operator, the part of the genome that encodes this limb has clearly been reused. This may happen if crossover takes place between two individuals that have a common ancestor. This kind of gene duplication or gene reuse is a very beneficial effect of the crossover operator.

6.2 Genetic encoding

Before comparing the different kinds of couplings that I tested, I compare the general encoding with the encoding that restricts the phenotype space to symmetric configurations.

As expected, the symmetric encoding outperforms the general encoding by far. Even though fit individuals are also obtained with the general encoding, a lot of these robots are very simple. The majority are rollers. The configuration of these robots is often a linear chain of modules and locomotion is achieved by rolling sideways. The problem is, that this kind of solution is both easy to find by the GA and very fit. In fact, a good roller defeats even the fittest walkers. As you can see in the examples of evolved robots given at the end of the chapter, the general encoding also evolves interesting robots. Locomotion strategies include rolling, walking, crawling, hopping, etc. It is striking, that fit individuals are always kind of symmetric. As mentioned above, it seems that the GA 'tries' to evolve symmetric configurations but it converges to local optima that are not exactly symmetric because the phenotype space is too big and too unstructured.

The genetic encoding that limits evolved robots to symmetric configurations exceeded my expectations. Using the convergence test to stop the GA, an evolutionary run takes generally about two hours on a 1.25 GHz G4 Mac computer. In this short time, highly interesting and fit robots are evolved in almost every run. The majority of the robots are crawlers or walkers. These chain-type robots are more complex and the evolved gaits are more dynamic and sophisticated than those from previous research. Refer to the example robots at the end of the chapter. Unfortunately the elegance of the evolved robots is not visible on the images. Movies are available on my project web page at:

<http://birg.epfl.ch/page32031.html>

In order to compare the performance of the GA with the general and the symmetric encoding I did 15 evolutionary runs for both encodings, using the convergence test described in chapter 4.3.4 to stop the GAs. For each run, I recorded the number of fitness evaluations (i.e. the total number of generated children) and of course the maximum

Encoding	Evaluations	Max. fitness
General	2435.66	2.74
Symmetric	2208.21	2.91

Table 3: Average number of evaluations and average of the maximum fitness for both encodings. The average has been computed from 15 genetic algorithms. GAs using the encoding that restricts the phenotype space to symmetric configurations evolve fitter individuals in shorter time.

fitness. The averages of both measures are compared in table 3. Due to the smaller and better structured phenotype space, GAs using the symmetric encoding evolved fitter individuals in shorter time. However, the difference is not as big as one might expect. This is explained by the fact, that the general encoding usually also finds fit individuals, but they are much simpler and less interesting than those evolved with the symmetric encoding.

6.3 Nonlinear vs. Harmonic oscillators

I have also evaluated the performance of the GA for the different controllers. For this experiment, I used a steady state GA with a population size of 100 robots. At each generation, 50 children were generated. Again, I used the convergence test to stop the GA and recorded the maximum fitness and the number of generations. I did 14 evolutionary runs for each type of controller. The controllers included harmonic oscillators, nonlinear oscillators with unidirectional couplings and nonlinear oscillator with bidirectional couplings. The bidirectional couplings have four free parameters; I did not test bidirectional couplings with two free parameters because I expect them to perform similarly to unidirectional couplings.

As you can see on table 4, bidirectional couplings with four free parameters didn't give satisfactory results. This is explained by the larger search space and problems with convergence of the coupled nonlinear oscillators. Indeed, some of the evolved robots had conflicting couplings and their oscillators did not converge to stable phase differences.

To my surprise, the nonlinear oscillators with unidirectional couplings outperform the harmonic oscillators even though their search space is bigger. This is surprising because the main advantages of nonlinear oscillators such as asynchronous distributed control and smooth gait transitions are not relevant for the fitness in simulation. A possible explanation are the higher amplitudes caused by the couplings (I did not use the energy balanced couplings). I will repeat this experiment with: a) Energy balanced couplings; b) Harmonic

Controller	Generations	Max. fitness
Harmonic	66.69	2.94
Unidirectional	67.38	3.17
Bidirectional	75.09	2.53

Table 4: Average number of generations and the average of the maximum fitness for different types of controllers: Harmonic oscillators, nonlinear oscillators with unidirectional couplings and nonlinear oscillators with bidirectional couplings and four free parameters. The average has been computed from 14 GAs for each type of controller.

oscillators with higher amplitudes, so that the range of the amplitude matches the actual range of the nonlinear oscillators (the upper limit of the amplitude is boosted by the couplings in the case of nonlinear oscillators).

6.4 Genetic Algorithms

As explained in chapter 4.3, I implemented three different kinds of genetic algorithms: Incremental GA, steady state GA and a genetic algorithm with multiple, migrating populations. The GA with multiple populations performed poorly. An evolutionary run takes very long and the results are not better than those evolved with the incremental and steady state GA. This is because the probability is quite high, that a simple locomoting robot is generated by the random initialization procedure in one of the populations. By migration, this simple solution (that corresponds to a local optimum) then ‘poisons’ the other populations.

I believe that for the same reason, large populations don’t perform well for incremental and steady state GAs (table 5). As expected, the incremental GA converges faster than the steady state GA. On the other hand I was surprised that the incremental GA also evolves individuals that are at least as fit as those from the steady state GA.

The random replacement strategy mentioned in chapter 4.3 proved to be too destructive, i.e. the average and maximum fitness of the population could not be improved because too

	Incremental GA		Steady state GA	
Population size	Evaluations	Max. fitness	Evaluations	Max. fitness
50	1990	3.37	2066	2.88
100	2090	3.23	2398	3.20
200	1785	2.89	2330	2.64

Table 5: Performance of incremental and steady state GAs for various population sizes. The incremental GA with a population size of 50 performs best.

Replacement	Evaluations	Max. fitness
Worst	2177.14	2.88
Random	1120.00	1.14

Table 6: The random replacement strategy proved to be too destructive. The averages are computed from 12 GAs.

Scaling	Evaluations	Max. fitness
Rank	1883.33	3.00
Linear	2568.88	2.72

Table 7: The rank-proportional roulette-wheel method performed much better than the more traditional fitness-proportional roulette-wheel method with linear scaling of the fitness values (averages from 12 GAs).

many fit individuals were being replaced by the offspring (table 6).

I can highly recommend the rank-proportional roulette-wheel method for selection (table 7). Not only does it perform much better than the common fitness-proportional roulette-wheel method with linear scaling of the fitness values, but also is it much easier to implement.

6.5 Energy balanced couplings

My analysis of the nonlinear oscillators is rewarded with a significant reduction of the parameter space. Instead of optimizing both weights a_{ij} and b_{ij} , a single parameter ξ may be optimized. This is beneficial for the GA, but especially important in the context of online optimization, where the parameter space must be reduced to a minimum.

Even with quadratically converging optimization algorithms such as Powell's method, the fitness function must be evaluated many times. The number of evaluations needed to find the optimum grows exponentially with the number of parameters. Initially, online optimization of locomotion in a reasonable time seemed impossible to me. Only with phase prediction and the resulting limitation of the search space do I believe online optimization to be feasible within dozens of minutes.

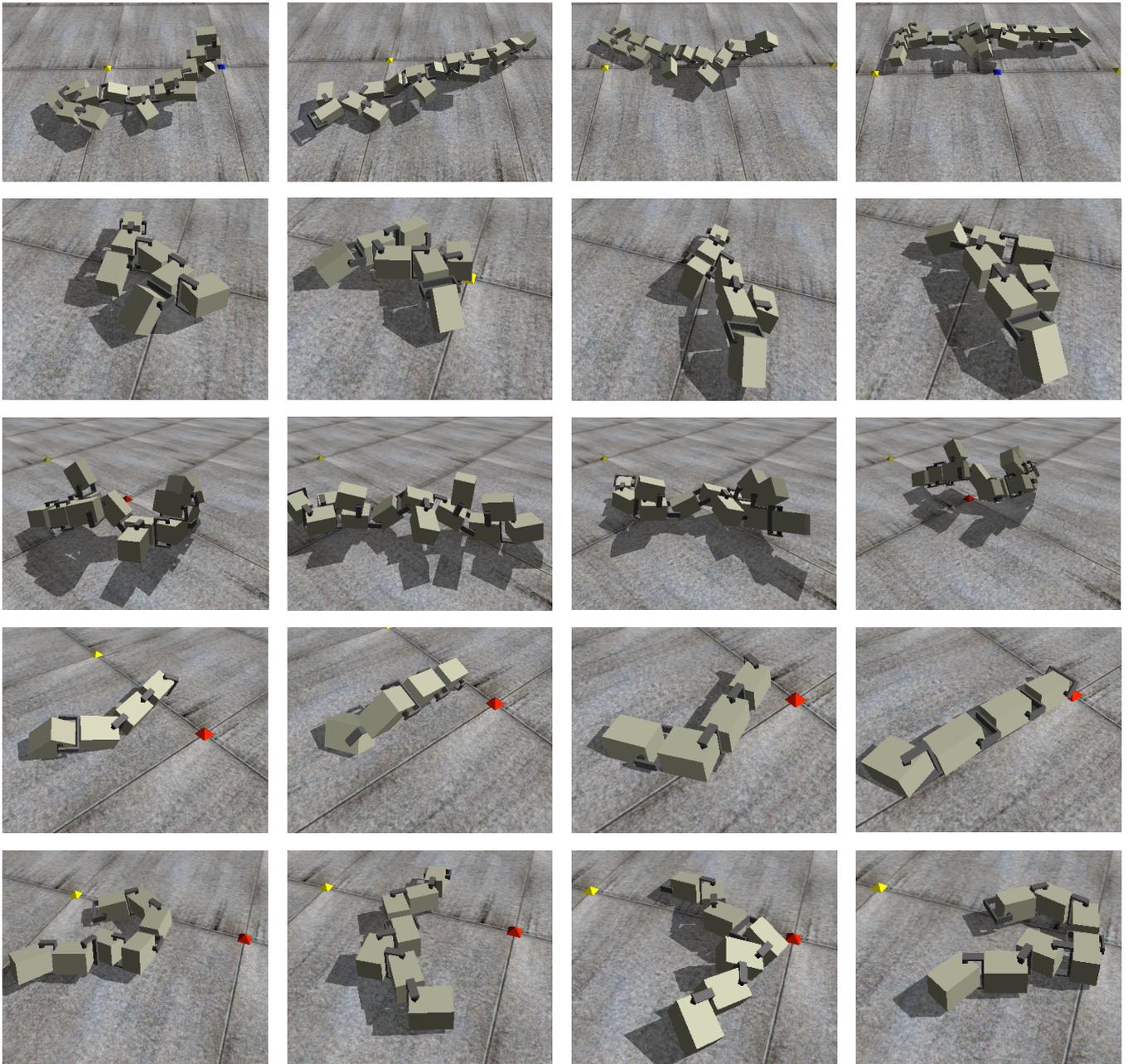
I have estimated that it takes about one simulated minute to optimize the fitness function in one dimension. An iteration of Powell's method involves N one-dimensional optimizations, where N is the number of dimensions (i.e. the number of parameters). If the fitness landscape is not full of small, local optima, we may indeed dream that online optimization of locomotion could be very fast.

Unfortunately, I did not finish the implementation of Powell's method at the time of writing. Thus, I have neither positive nor negative results. Please visit my project web page for the latest news regarding online optimization:

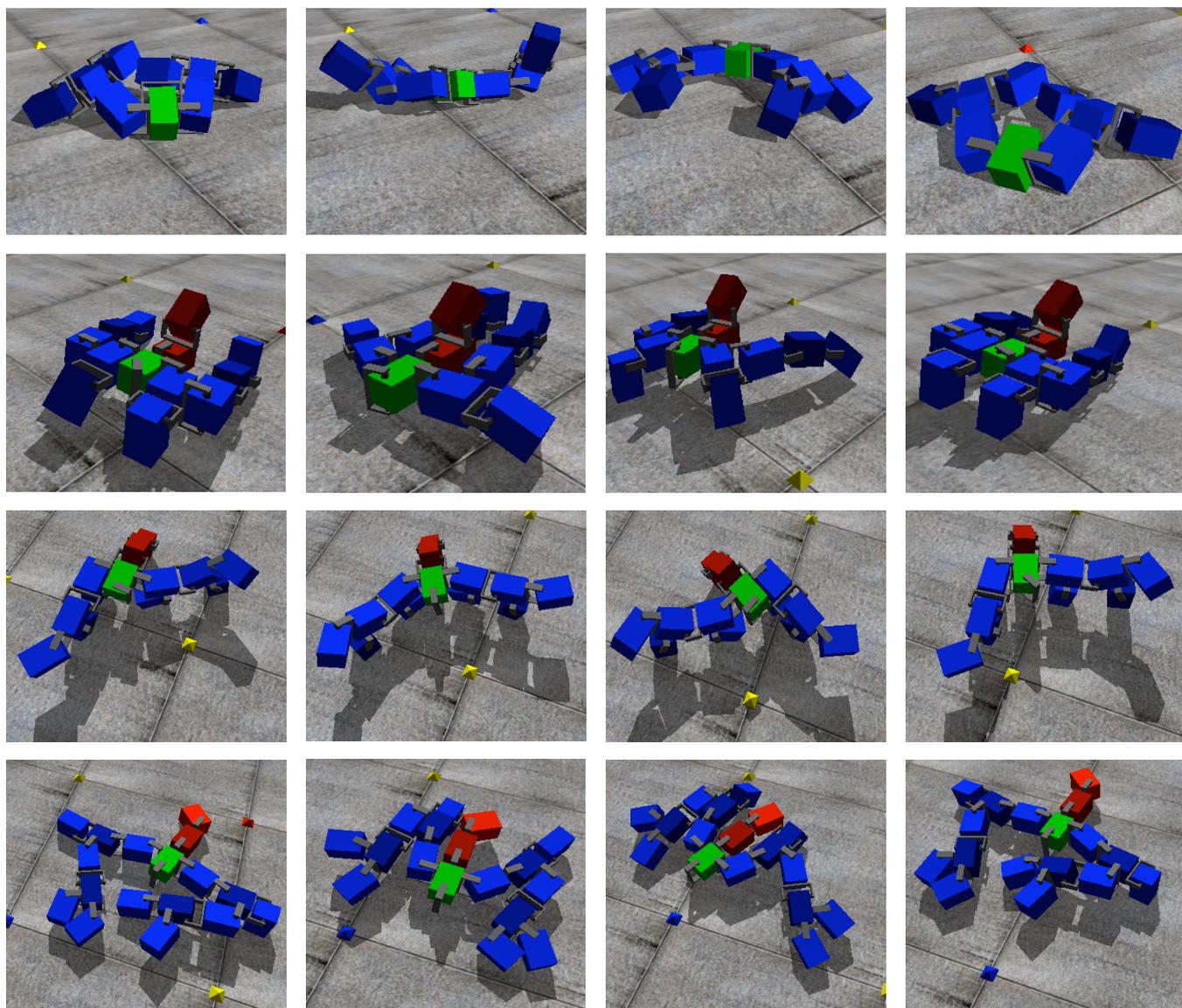
<http://birg.epfl.ch/page32031.html>

Thanks to the energy-balanced couplings, it is possible to use nonlinear oscillators with robots that have been evolved using harmonic oscillators. I have successfully tested this in simulation. The phases of the harmonic oscillators must simply be converted to the coupling parameters that imply this phase. Indeed, the robots then have the exact same behavior when using harmonic oscillators or nonlinear oscillators. This is only possible when using energy-balanced couplings. Otherwise, the couplings cause a significant amplitude difference of the oscillators that completely changes the gait.

6.6 Examples of evolved robots



Asymmetric robots



Some examples of robots that have been evolved with the genetic encoding that restricts the phenotype space to symmetric configurations. The robot on the top uses bidirectional couplings with four free parameters, the second robot harmonic oscillators and the two robots on the bottom unidirectional couplings.

Chapter 7

Conclusions and future work

I have presented a bio-inspired approach for self-organization of locomotion for modular robots. In contrast to previous research in self-organization of chain-type modular robot locomotion, I co-evolve the configurations with the CPGs in analogy to how the bodies co-evolve with the nervous system.

Inspired by symmetry in nature and the tendency of the GA to evolve quasi-symmetric individuals, I designed a genetic encoding that restricts the phenotype space to symmetric configurations. The results of this encoding exceeded my expectations. Evolved robots use more modules and are more complex than those in previous research. The gaits are more elegant and sophisticated than those of any other chain-type modular robots that I have seen so far. Furthermore, the GA evolves these individuals in a surprisingly short time (about two hours on a high-end PC). In summary, fitter and more complex robots are evolved in shorter time with the symmetric encoding.

We may philosophize about the origin and purpose of symmetry in nature based on these results. In chapter 3.1.3 I have argued that symmetry might be a fundamental characteristic of efficient locomotion. Even if this was true, it is no explanation why the whole body of animals is symmetric and not just the limbs that are used for locomotion. After the complicated implementation of the symmetric genotype-to-phenotype mapping, I am amazed how the body succeeds growing almost perfectly symmetrical limbs. Obviously, there must be a mechanism at the level of the developmental gene regulatory networks [Hinman et al. 2003] that is responsible for the bilateral symmetry. I hypothesize, that the ‘purpose’ of symmetry in nature is the same as of my genetic encoding: A restriction and structurization of the phenotype space. Indeed, nature is confronted with the same problem as my genetic algorithm: A huge and unstructured search space. Now suppose that a developmental gene regulatory network that implies a symmetric morphology is evolved per coincidence, through mutation. The evolution of such a species would be boosted because of the limited and structured phenotype space. If the analogy with my genetic encoding holds, ‘symmetric species’ would soon become more sophisticated and fitter than ‘asymmetric species’ and finally extinguish them. Note that these considerations are purely philosophical, it would be interesting to further investigate the role of developmental gene regulatory network within this context.

With respect to the different types of genetic algorithms that I tested, I can conclude that the performance of the evolved robots cannot be improved with more sophisticated algorithms such as the GA with migrating populations. Therefore, to evolve more complex modular robots, one should test a developmental coding based on L-systems for instance.

I found that the nearest neighbor coupling scheme is well adapted for a modular robot CPG. Note that with the symmetric encoding, the couplings are relayed along the spine to the limbs, analogously to the architecture of the neural CPG in vertebrates. I believe that the nonlinear oscillators are well suited as canonical subsystems to model the neural oscillatory centres and that there is no benefit in using more complex models for open loop locomotion. Clearly, we should investigate if the dynamics of the nonlinear oscillators are rich enough to integrate sensory feedback.

In chapter 4.4 I have outlined a distributed, asynchronous control algorithm for implementing the CPG on the hardware. The algorithm is based on strictly local interaction and is robust in face of temporary communication failures.

Using polar coordinates for the coupling parameters, one can easily predict the phase difference between two coupled oscillators, which has been considered impossible before. Instead of optimizing both weights a_{ij} and b_{ij} of a coupling, a single parameter ξ may be optimized. This is beneficial for both the GA and the online adaptation method. I have also proposed a new coupling term that ensures exact convergence to a sine. The couplings have no effect on the amplitude when using this term. This is interesting for the distributed control algorithm mentioned above, because only with energy-balanced couplings is it resistant to temporary communication failures.

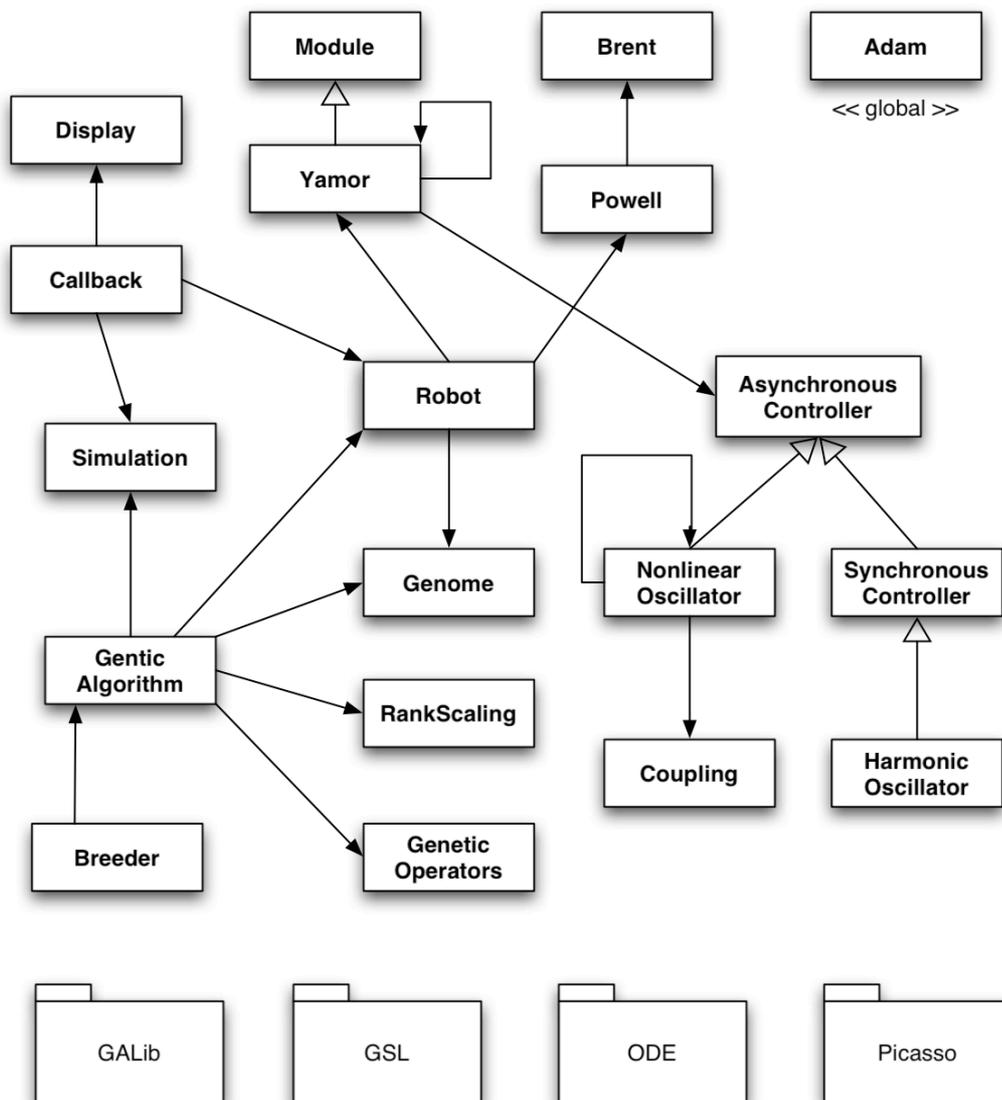
Unfortunately the online adaptation algorithm is not finished at the time of writing. Success or failure of my approach depends on the topology of the fitness landscape. Only experiments can show if efficient locomotion is obtained with randomly assembled structures. However, I am confident that online adaptation of evolved individuals is possible with Powell's method.

Appendix - implementation

The source code is more than 8000 lines. The description of the class interfaces alone would take dozens of pages. I decided that it makes more sense to include the sources and the documentation of the implementation electronically.

On the CD that is included with this report, you find the complete HTML documentation of the project. The documentation has been generated automatically from the sources with Doxygen. It is also available on the project web page at:

<http://birg.epfl.ch/page32031.html>



Simplified design class diagram. For details, refer to the online documentation.

References

[Butler et al. 2002]

Z. Butler, K. Kotay, D. Rus, and K. Tomita. *Generic decentralized control for a class of self-reconfigurable robots*. In Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA'02), pages 809--815, Washington, D.C., USA, 2002.

[Bojinov et al. 2000]

H. Bojinov, A. Casal, and T. Hogg. *Emergent structures in modular self-reconfigurable robots*. In Proceedings of the IEEE int. conf. on Robotics & Automation, volume 2, pages 1734 -1741, San Francisco, California, USA, 2000.

[Bongard & Pfeifer 2001]

J.C. Bongard, R. Pfeifer. Repeated structure and dissociation of genotypic and phenotypic complexity in artificial ontogeny. In Genetic and Evolutionary Computation Conference, p. 829-836, 2001.

[Bourquin 2004]

Y. Bourquin. *Self-Organization of Locomotion in Modular Robots*. Unpublished Master Thesis, Swiss Federal Institute of Technology Lausanne, 2004. <http://birg.epfl.ch/>

[Buchli & Ijspeert 2004]

J. Buchli and A.J. Ijspeert. *Distributed central pattern generator model for robotics application based on phase sensitivity analysis*. In Proceedings of the First International Workshop on Biologically Inspired Approaches to Advanced Information Technology - Bio-ADIT2004, Lecture Notes in Computer Science. Springer, January 2004.

[Castano & Will 2001]

A. Castano and P. Will. *Representing and Discovering the Configuration of CONRO Robots*. In Proceedings of the IEEE International Conference on Robotics and Automation, Vol. 4, pp. 3503-3509, Seoul, Korea, May 2001.

[Chirikjian et al. 1996]

G. Chirikjian, A. Pamecha, and I. Ebert-Uphoff. *Evaluating efficiency of self-reconfiguration in a class of modular robots*. J. Robot. Syst., vol. 13, no. 5, pp. 317-338, 1996.

[Chirikjian et al. 2002]

G. Chirikjian, Y. Zhou, and J. Suthakorn. *Self-Replicating Robots for Lunar Development*. ASME & IEEE Transactions on Mechatronics in the Special Issue of Self-Reconfiguration Robots, Vol. 7, Issue: 4, Dec 2002.

[Dittrich 2004]

E. Dittrich. *Modular Robot Unit - Characterisation, Design and Realisation*. Unpublished Master Thesis, Swiss Federal Institute of Technology Lausanne, 2004. <http://birg.epfl.ch/>

[Duff et al. 2001]

D. Duff, M. Yim, and K. Roufas. *Evolution of PolyBot: A Modular Reconfigurable Robot*. In Proc. of the Harmonic Drive Intl. Symposium, Nagano, Japan, Nov. 2001, and Proc. of COE/Super-Mechano-Systems Workshop, Tokyo, Japan, Nov. 2001.

[Esterin et al. 1999]

D. Estrin, R. Govindan, J. S. Heidemann, and S. Kumar. *Next Century Challenges: Scalable Coordination in Sensor Networks*. In Mobile Computing and Networking, 1999, p. 263-270.

[Fromherz et al. 2001]

M. Fromherz, T. Hogg, Y. Shang, and W. Jackson. *Modular robot control and continuous constraint satisfaction*. In Proceedings of the IJCAI-01 Workshop on Modelling and Solving Problems with Constraints, Seattle, WA, 2001.

[Hinman et al. 2003]

V. F. Hinman, A. T. Nguyen, R. A. Cameron, and E. H. Davidson. *Developmental gene regulatory network architecture across 500 million years of echinoderm evolution*. Proc Natl Acad Sci U S A. 2003 November 11; 100(23): 13356–13361. Published online 2003 October 31.

[Hornby & Pollack 2001]

G. S. Hornby, J. B. Pollack. Body-brain coevolution using l-systems as a generative encoding. In Genetic and Evolutionary Computation Conference, 2001.

[Ijspeert 1998]

A. J. Ijspeert. *Design of artificial neural oscillatory circuits for the control of lamprey- and salamander-like locomotion using evolutionary algorithms*. 1998. PhD Thesis.

[Ijspeert & Cabelguen 2003]

A. J. Ijspeert, J.-M. Cabelguen. *Gait transition from swimming to walking: investigation of salamander locomotion control using non-linear oscillators*. In Proceedings of Adaptive Motion in Animals and Machines, 2003.

[Hosokawa et al. 1998]

K. Hosokawa, T. Tsujimori, T. Fujii, H. Kaetsu, H. Asama, Y. Kuroda, and I. Endo. *Self-organizing collective robots with morphogenesis in a vertical plane*. In Proceedings of the IEEE International Conference on Robotics & Automation, pages 2858--2863, Leuven, Belgium, 1998.

[Kamimura et al. 2003]

A. Kamimura, H. Kurokawa, E. Toshida, K. Tomita, S. Murata, and S. Kokaji. *Automatic locomotion pattern generation for modular robots*. In Proceedings of the IEEE Int. Conf. on Robotics and Automation (ICRA), 2003.

[Kimura et al. 1999]

H. Kimura, S. Akiyama, and K. Sakurama. *Realization of dynamic walking and running of the quadruped using neural oscillators*. Autonomous Robots, 7 (3), 247-258, 1999.

[Kirkpatrick 1983]

S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. *Optimization by simulated annealing*. Science, 220(4598):671-680, 1983.

[Komosinsky & Rotaru-Varga 2001]

M. Komosinsky, A. Rotaru-Varga. Comparison of Different Genotype Encodings for Simulated 3D Agents. Artificial Life Journal, 7:395-418, 2001.

[Kurokawa et al. 2003]

H. Kurokawa, A. Kamimura, E. Yoshida, K. Tomita, S. Kokaji, and S. Murata. *M-TRAN II: Metamorphosis from a Four-Legged Walker to a Caterpillar*. In Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2454-2459, 2003.

[Kurokawa et al. 2004]

H. Kurokawa, E. Yoshida, K. Tomita, A. Kamimura, S. Murata, and S. Kokaji. *Deformable Multi M-TRAN Structure Works as Walker Generator*. In Proceedings of the 8th Conference on Intelligent Autonomous Systems (IAS-8), p. 746-753, 2004.

[Lipson & Pollack 2000]

H. Lipson, J.B. Pollack. Automatic design and manufacture of robotic lifeforms. Nature, 406:974-978, 2000.

[Lund 2001]

H. H. Lund. Co-evolving Control and Morphology with LEGO Robots. In Hara and Pfeifer (eds.) Morpho-functional Machines, Springer-Verlag, Heidelberg, 2001.

[Marbach & Ijspeert 2004]

D. Marbach, and A.J. Ijspeert. *Co-evolution of Configuration and Control for Homogenous Modular Robots*. In Proceedings of the Eighth Conference on Intelligent Autonomous Systems (IAS8), F. Groen et al. (Eds.), IOS Press, pp 712-719, 2004.

[Mesot 2004]

B. Mesot. *Self-Organisation of Locomotion in Modular Robots: A Case Study*. Unpublished Master Thesis, Swiss Federal Institute of Technology Lausanne, 2004. <http://birg.epfl.ch/>

[Michel 2004]

O. Michel. *Webots: Professional Mobile Robot Simulation*. International Journal of Advanced Robotic Systems, Volume 1 Number 1, pp 39-42, 2004.

[Mondada et al. 2002]

F. Mondada, A. Guignard, A. Colot, D. Floreano, J.-L. Deneubourg, L.M.Gambardella, S. Nolfi, and M. Dorigo. *SWARM-BOT: A New Concept of Robust All-Terrain Mobile Robotic System*. Technical report, LSA2 - I2S - STI, Swiss Federal Institute of Technology, Lausanne, Switzerland, 2002.

[Murata et al. 1994]

S. Murata, H. Kurokawa, and S. Kokaji. *Self-assembling machine*. In Proceedings of the 1994 IEEE International Conference on Robotics and Automation (ICRA), p. 441-448, San Diego, California, 1994.

[Murata et al. 2000]

S. Murata, E. Yoshida, K. Tomita, H. Kurokawa, A. Kamimura, and S. Kokaji. *Hardware design of modular robotic system*. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, p. 2210-2217, Takamatsu, Japan, 2000.

[Murata et al. 2002]

S. Murata, E. Yoshida, A. Kamimura, H. Kurokawa, K. Tomita, and S. Kokaji. *M-TRAN: Self-Reconfigurable Modular Robotic System*. IEEE/ASME Transactions on Mechatronics, Vol. 7, No. 4, pp. 431-441, 2002.

[Murata et al. 2004]

S. Murata, A. Kamimura, H. Kurokawa, E. Yoshida, K. Tomita, and S. Kokaji. *Self-Reconfigurable Robots: Platforms for Emerging Functionality*. Embodied Artificial Intelligence, Lecture Notes in Artificial Intelligence 3139, pp. 312-330, Springer-Verlag, 2004.

[Mytilinaios et al. 2004]

E. Mytilinaios, D. Marcus, M. Desnoyer, and H. Lipson. *Designed and Evolved Blueprints For Physical Self-Replicating Machines*. In Proceedings of the Ninth Int. Conference on Artificial Life (ALIFE IX), p.15-20, 2004.

[Østergaard & Lund 2003]

E.H. Østergaard and H.H. Lund. *Evolving Control for Modular Robotic Unit*. In Proceedings of CIRA'03, IEEE International Symposium on Computational Intelligence in Robotics and Automation, Kobe, Japan, p. 886-892, 2003.

[Salemi et al. 2001]

B. Salemi, W. Shen, and P. Will. *Hormone-Controlled Metamorphic Robots*. International Conference on Robotics and Automation, Seoul, Korea, 2001

[Salemi et al. 2003]

B. Salemi, P. Will, and W. Shen. *Distributed Task Negotiation in Modular Robots*. Robotics Society of Japan, Special Issue on "Modular Robots", 2003.

[Salemi & Shen 2004]

B. Salemi, and W. Shen. *Distributed Behavior Collaboration for Self-Reconfigurable Robots*. International Conference on Robotics and Automation, New Orleans, LA, USA, 2004.

[Sanchez et al. 1997]

E. Sanchez, D. Mange, M. Sipper, M. Tomassini, A. Perez-Urbe, A. Stauffer. *Phylogeny, Ontogeny, and Epigenesis: Three Sources of Biological Inspiration for Softening Hardware*. In Proc. 1st Int. Conf. on Evolvable Systems (ICES'96), LNCS, v. 1259, Springer-Verlag, Berlin, 1997.

[Shen et al. 2002]

W. Shen, B. Salemi, and P. Will. *Hormone-Inspired Adaptive Communication and Distributed Control for Self-Reconfigurable Robots*. IEEE Trans. on Robotics and Automation 18(5):1-12, 2002.

[Shen et al. 2004]

W. Shen, P. Will, A. Galstyan, and C. Chuong. *Hormone-inspired self-organization and distributed control of robotic swarms*. Autonomous Robots, 17:93-105, 2004.

[Shik et al. 1966]

Shik, M., Severin, F., & Orlovsky, G. (1966). *Control of walking by means of electrical stimulation of the mid-brain*. Biophysics, 11, 756-765.

[Sims 1994]

K. Sims. *Evolving 3D Morphology and Behavior by Competition*. Artificial Life IV Proceedings, ed. by R. Brooks & P. Maes, MIT Press, p. 28-39, 1994.

[Smith]

Russell Smith. Open Dynamics Engine (ODE), web page: <http://q12.org/ode/>

[Støy et al. 2002]

K. Støy, W.-M. Shen, and P. Will. *On the Use of Sensors in Self-Reconfigurable Robots*. In proceedings of the 7th international conference on simulation of adaptive behavior (SAB'02), pages 48-57, Edinburgh, UK, August 4-9, 2002.

[Støy et al. 2003]

K. Støy K, W. Shen, P. Will. *Implementing configuration dependent gaits in a self-reconfigurable robot*. In Proceedings of the IEEE International conference on Robotics and Automation (ICRA), Taipei, Taiwan, 2003.

[Støy 2004]

K. Støy. *Controlling Self-Reconfiguration using Cellular Automata and Gradients*. In Proceedings of the The 8th Conference on Intelligent Autonomous Systems (IAS-8), Amsterdam, 2004.

[Taga 1998]

G. Taga. *A model of the neuro-musculo-skeletal system for anticipatory adjustment of human locomotion during obstacle avoidance*. Biological Cybernetics, 78 (1), 9-17, 1998.

[Tempesti et al. 2002]

G. Tempesti, D. Roggen, E. Sanchez, Y. Thoma. *A POEtic Architecture for Bio-Inspired Hardware*. In Proc. 8th Intl. Conf. on the Simulation and Synthesis of Living Systems (Artificial Life VIII), Sydney, Australia, 9-13 Dec. 2002. MIT Press, Cambridge, MA, 2002, pp. 111-115.

[Tomita et al. 1999]

K. Tomita, S. Murata, H. Kurokawa, E. Yoshida, S. Kokaji. *A Self-Assembly and Self-Repair Method for a Distributed Mechanical System*. IEEE Transactions on Robotics and Automation, Vol.15, No.6, pp.1035-1045, 1999.

[Unsal et al. 2000]

C. Unsal, H. Kiliccote, M. Patton, and P. Khosla. *Motion Planning for a Modular Self-Reconfiguring Robotic System*. Distributed Autonomous Robotic Systems 4, Springer, November, 2000.

[Ventrella 1994]

J. Ventrella. *Explorations in the emergence of morphology and locomotion behavior in animated characters*. In R. Brooks and P. Maes, editors, *Proceedings of the Fourth Workshop on Artificial Life*, Boston, MA, MIT Press, 1994.

[Vassilvitskii et al. 2002]

S. Vassilvitskii, J. Kubica, E. Rieffel, J. Suh, and M. Yim. *On the General Reconfiguration Problem for Expanding Cube Style Modular Robots*. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2002.

[Vona & Rus 2000]

M. Vona and D. Rus. *A Physical Implementation of the Self-reconfigurable Crystalline Robot*. In *Proc. of the IEEE Int'l Conf. on Robotics and Automation*, San Francisco, CA, p. 1726-1733, 2000.