Learning to move in modular robots using central pattern generators and online optimization

Alexander Sproewitz, Rico Moeckel, Jérôme Maye, Auke Jan Ijspeert*
School of Computer and Communication Science
EPFL - Ecole Polytechnique Fédérale de Lausanne
Station 14, CH-1015 Lausanne, Switzerland
{Alexander.Sproewitz, Jerome.Maye, Auke.Ijspeert}@epfl.ch, moeckel@ini.phys.ethz.ch

Abstract

This article addresses the problem of how modular robotics systems, i.e. systems composed of multiple modules that can be configured into different robotic structures, can learn to locomote. In particular, we tackle the problems of *online* learning, that is, learning while moving, and the problem of dealing with unknown arbitrary robotic structures.

We propose a framework for learning locomotion controllers based on two components: a central pattern generator (CPG) and a gradient-free optimization algorithm: Powell's method. The CPG is implemented as a system of coupled nonlinear oscillators in our YaMoR modular robotic system, with one oscillator per module. The nonlinear oscillators are coupled together across modules using Bluetooth communication to obtain specific gaits, i.e. synchronized patterns of oscillations among modules. Online learning is done by running the Powell optimization algorithm in parallel to the CPG model, with the speed of locomotion being the criterion to be optimized. Interesting aspects of the optimization are: it is carried out online, it does not require stopping or resetting the robots, and it is fast.

We present results showing the interesting properties of this framework for a modular robotic system. In particular, our CPG model can readily be implemented in a distributed system, it is cheap computationally, it exhibits limit cycle behavior (temporary perturbations are rapidly forgotten), it produces smooth trajectories even when control parameters are abruptly changed, and it is robust against imperfect communication among modules. We also present results of learning to move with three different robot structures. Interesting locomotion modes are obtained after running the optimization for less than 60 minutes.

1 Introduction

As for any mobile robot, one of the key features that a self-reconfigurable modular robot should exhibit is robust locomotion. Designing efficient locomotion controllers for modular robotic systems is however a difficult and unsolved problem. Their locomotion control suffers from all the traditional difficulties of locomotion control in robots with multiple degrees of freedom, with in addition difficulties related to their specific modular structure. Indeed, the control of locomotion requires multi-dimensional coordinated rhythmic patterns that need to be correctly tuned such as to satisfy multiple constraints: the capacity to generate forward motion, with low energy, without falling over, while adapting to possibly complex terrain (uneven ground, obstacles), and while allowing the modulation of speed and direction. These difficult problems are not yet satisfactorily solved for monolithic robots (e.g. quadruped or biped robots with a fixed structure), in particular for locomotion in complex terrains. Controlling a self-reconfigurable modular robot is even more difficult because of a mechanical structure that might evolve over time due to (self-)reconfiguration, uncertainties in the state of the mechanical structure due to imperfect connection mechanisms, and possibly imperfect communication between different modules, to name a few. Model-based approaches that are successfully used for some monolithic robots (e.g. humanoid robots) are therefore in many cases not suitable for modular robotic systems because of the difficulty to maintain accurate and up-to-date models.

In this article, we propose a framework for learning locomotion controllers based on two components: a central pattern generator and a gradient-free optimization algorithm: Powell's method (Press et al., 1994). Our approach is inspired by a control mechanism that nature has found to deal with the redundancies in animal bodies and the requirement to easily modulate locomotion: central pattern generators. Central pattern generators (CPGs) are neural networks capable of producing coordinated patterns of rhythmic activity without any rhythmic inputs from sensory feedback or from higher control centers (Delcomyn, 1980). A similar approach has been taken by Kamimura et al. (2003, 2004).

Even completely isolated CPGs in a Petri dish can produce patterns of activity, called fictive locomotion, that are very similar to intact locomotion when activated by simple electrical or chemical stimulation (Grillner, 1985). But, while sensory feedback is not needed for generating the rhythms, it plays a very important role for shaping the rhyth-

*To whom correspondence should be addressed.

mic patterns and for keeping CPGs and body movements coordinated. Typically, varying simple stimulation allows modulation of both the speed and direction of locomotion. From a control point of view, CPGs therefore implement some kind of internal model, i.e. a controller that "knows" which torques need to be rhythmically applied to obtain a given speed of locomotion. Interestingly, CPGs combine notions of stereotypy (steady state locomotion tends to show little variability) and of flexibility (speed, direction and types of gait can continuously be adjusted).

In this article we will argue that CPGs are ideal building blocks for constructing locomotion controllers for a self-reconfigurable modular robot and for running online learning algorithms. We present a CPG based on a system of coupled amplitude-controlled phase oscillators. Our CPG is implemented and tested on our YaMoR modular robotic system. A YaMoR module has one actuated degree of freedom, can be mechanically attached up to 5 other modules, and communicates wirelessly with other modules via Bluetooth (Moeckel et al., 2006). Each module is programmed to run one nonlinear oscillator to control the oscillations of its servomotor. The nonlinear oscillators are coupled together across modules using Bluetooth communication to obtain specific gaits, i.e. synchronized patterns of oscillations among modules. Different stable gaits can be obtained by adjusting the parameters of the CPG that determine the frequency, amplitude and phase lag of oscillations.

CPGs as systems of coupled nonlinear oscillators are useful for locomotion control in any type of articulated robot: they exhibit limit cycle behavior (temporary perturbations are rapidly forgotten), they ensure good coordination between different degrees of freedom, they produce smooth trajectories even when control parameters are abruptly changed, and they can readily integrate sensory feedback signals for online modulation. A review of the use of CPG models in robotics can be found in Ijspeert (2008).

In addition, they are particularly useful for modular robotics: they are ideally suited for a distributed implementation (e.g. with one or more oscillator per module), they are cheap computationally, they produce robust synchronization even if CPUs have different clock frequencies, and they are robust against imperfect communication among modules (time delays, packets loss, noise, ...).

Finally CPGs are also a good substrate for online optimization, a property that is used in this article. Indeed an optimization algorithm, in our case Powell's method, can run in parallel to the CPG and regularly update its parameters. Despite abrupt parameter changes, the produced trajectories will smoothly converge towards the new limit cycle after a short transient period. This means that the robot does not need to be stopped or reset between iterations.

This possibility to do *online* optimization, i.e. learning while moving, is one of the main contributions of this article.[1] Being able to learn gaits online, as opposed to *offline* with a simulator or a model, is of great importance for self-reconfigurable modular robotics. First of all, since the optimization does not rely on a simulation or a model, it will avoid the problem of having to transfer the locomotion controller from a simulation/model to the real world. This transfer is often very problematic due to the difficulty of properly modeling complex environments (e.g. friction and contact models with an uneven floor). Second, it allows learning a locomotion gait for a new, previously unknown configuration. This may be the case after self-reconfiguration or self-assembly. Third, the locomotion gait may be continuously adapted to changes in the robotic structure (e.g. because of the addition, removal, or failure of modules). Fourth, locomotion can be adapted to changes in the environment. To the best of our knowledge, there has been no previous research in online optimization of chain-type modular robot locomotion.

This works follows preliminary results in simulation with a slightly different type of CPG that had shown the potential of the approach for modular robotics (Marbach and Ijspeert, 2005). In the next sections, we first make a brief overview of related work (Section 2). We then present the mechanical and electronic design of our robot YaMoR, as well as the Bluetooth protocol that we designed for communication between modules (Section 3). In Section 4, we present the CPG model and the optimization algorithm. Experiments demonstrating the activity of the CPG and the online learning with different robot structures are presented in Section 5. Our approach is discussed in Section 6.

2 Related work

Locomotion control in modular robots Modular robots are generally classified as being lattice-type or chain-type. Lattice modular robotic systems use cluster-flow locomotion and reconfiguration: in order to move, the robot continuously reconfigures (modules attaching and detaching over a lattice of other modules), thereby giving the impression that the module cluster "flows" on the ground and around obstacles. The Crystalline robot (Vona and Rus, 2000), Telecube (Vassilvitskii et al., 2002), and the ATRON (Ostergaard and Lund, 2003) are examples of such robots. Chain-type robots normally locomote in a static configuration (i.e. without using reconfiguration), using powered joints. See, for example, the M-TRAN II (Murata et al., 2002), the CONRO robot (Shen et al., 2004) and Polybot (Duff et al., 2001). Although reconfiguration can also be used for motion, it is normally used only in order to adapt to a new environment or function. For example, a robot could climb over an obstacle in a quadruped configuration and then reconfigure to a snake in order to slide through a small hole.[2]

In this article, we do not address locomotion through self-reconfiguration and only address the problem of locomotion in chain-like robots, that is locomotion that requires the periodic activation of articulated joints. Different locomotion

[1]In this article, we will use learning, optimization, and adaptation as synonyms.

[2]See the URL http://unit.aist.go.jp/is/dsysd/mtran/English/experimentE.htm for videos of the M-TRAN II performing such reconfigurations.

control algorithms have been developed for those types of robots; these include centralized gait tables, role-based control, hormone-based control, constraint-based control, and CPG-based control. The most common and obvious approach to control locomotion of a chain-type robot in a specific configuration is a centralized gait control table (Yim, 1994; Bongard et al., 2006). The gait control tables correspond to simplified finite state machines with a sequence of actions for each module. Actions are generally simple motor commands that set the desired angle. This is the simplest and probably most rigid way of generating a gait. The sequence is usually predefined (Yim, 1994), but can be adjusted during runtime (Bongard et al., 2006).

Role-based control has been proposed by Stoy et al. (2003), for locomotion of chain-type robots. It is an asynchronous distributed approach. Each module plays a role, which 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. Role-based control is an interesting approach to synchronize the system in a distributed manner but the algorithm has two major limitations: (1) it is assumed that modules have one and only one parent connector, and (2) the synchronization procedure can lead to discrete jumps in the generated trajectories A(t).

Shen et al. (2002) from the CONRO project took inspiration from hormones to design reliable, distributed control hormone-based 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. The hormones are used for several tasks such as distributed task negotiation, synchronization, and topology discovery (Salemi and Shen, 2004). For instance, they are used for discovering the current robot configuration and potentially trying to map it to a known configuration in a database (i.e. one for which a controller is available).

Zhang et al. (2002) have developed a constraint-based control framework to program modular robots. The idea is to see locomotion as a constraint-based problem and to use solvers to find satisfactory solutions in a given parameter space (typically parameters for a sine-based controller). Zhang et al. have successfully applied constraint-based control to modular self-reconfigurable robots in simulation. However, the authors mainly focus on scalability rather than on reliability.

The last type of locomotion control is based on the concept of CPGs. The goal is to produce oscillations as the limit cycle behavior in a system of coupled nonlinear oscillators. As discussed above, this approach benefits from many interesting properties such as synchronization between multiple oscillators and robustness against perturbations. In particular CPGs allow much more freedom in modulating gaits than sine-based controllers since changes in the control parameters lead to smooth changes in the produced oscillations. CPG-based control has been used by Kamimura et al. (2003, 2004), who use two-neuron Matsuoka oscillators as a CPG model for M-TRAN in their recent work. A genetic algorithm (GA) is applied to optimize the free CPG parameters *offline* (i.e. in simulation) for specific configurations. In Kamimura et al. (2004) the authors extend the CPG with a drift detection mechanism and demonstrate adaptive locomotion with M-TRAN in the face of external perturbations and varying environmental conditions. Input from the sensors affects the state and shapes the oscillatory output of the system. There is no long-term memory or learning effect. In contrast, we investigate in this article *online* adaptation, i.e. learning while moving without needing offline optimization with a simulator.

Note that CPGs are also increasingly used in monolithic robots, see for instance Kimura et al. (1999); Arena et al. (2004); Conradt and Varshavskaya (2003); Endo et al. (2005); Aoi and Tsuchiya (2006) for a few examples, and Ijspeert (2008) for a review. In our own work, we have used CPG models similar to the one presented here in snake-like robots (Ijspeert and Crespi, 2007), salamander-like robots (Ijspeert et al., 2007), quadruped robots (Buchli et al., 2006a), and humanoid robots (Righetti and Ijspeert, 2006).

Learning and locomotion As discussed above, locomotion control in robots with multiple degrees of freedom is a complex problem. It presents big challenges for learning algorithms for at least three reasons: (1) it is a highly nonlinear problem, (2) it involves large multi-dimensional search spaces, and (3) it is a problem for which the gradient of the functions to optimize is generally not available (the speed of locomotion, for instance, cannot be expressed as an analytical function of the open controller parameters because it intrinsically depends on the robot-environment dynamics). Different learning/optimization techniques have been used to improve locomotion controllers over time including evolutionary algorithms, reinforcement learning, and heuristic optimization.

Evolutionary algorithms are stochastic population-based optimization algorithms that can optimize a large class of cost functions (for instance, the cost functions do not need to be continuous as required for gradient-descent algorithms). That makes them well-suited to optimize performance measurements of a robot, for instance, the speed. They have been extensively used to design locomotion controllers (Beer and Gallagher, 1992; Lewis et al., 1993; Gruau and Quatramaran, 1997; Ijspeert and Kodjabachian, 1999; Paul and Bongard, 2001; Ijspeert, 2001), including for modular systems (Sims, 1994; Kamimura et al., 2003; Marbach and Ijspeert, 2005). But except for a few exceptions (e.g. Lewis et al. (1993)), the very large majority of these projects have optimized the controllers in a simulator. Simulators are used because of the large number of required evaluations (typically in order of thousands or higher, which would be excessive for a real robot) before converging to an optimum. This reliance on the simulator is however a problem in locomotion control. In many cases, the transfer of controllers from simulation to the real robot is poor, because many types of locomotion depend on complex physical interactions between the robot and its environment that are extremely hard to simulate properly. One possible solution to this problem is to combine evaluations in simulation and on the real robot (Bongard et al., 2006).

Reinforcement learning algorithms (Sutton and Barto, 1998) have also been used to train locomotion controllers. They usually have difficulties tackling problems with highly multidimensional search spaces and with continuous (as opposed to discrete) states and actions. But they were recently used with good success for online learning of biped locomotion (Collins et al., 2005; Geng et al., 2006; Matsubara et al., 2006; Nakamura et al., 2007) as well as quadruped locomotion (Kohl and Stone, 2004). It remains to be seen whether these tailored approaches can be adapted for learning locomotion in arbitrary structures as in modular robotic systems.

Finally, another approach is to use heuristic (gradient-free) optimization algorithms like variants of the Simplex method as used on quadruped robots (Weingarten et al., 2004) or Powell's method as used here and in preliminary experiments (Marbach and Ijspeert, 2005). As discussed later in this article, the advantage of these algorithms is that they are fast, but the disadvantage is that they present more risk to converge to a local optimum (as opposed to a global optimum) compared to stochastic methods.

3 The modular robotic system YaMoR

In this section we will briefly describe the robotic system that we use for experimentation in this article, in particular the mechanical and electronic design, as well as the Bluetooth communication protocol that we implemented.

3.1 Mechanical and electronic design

YaMoR consists of mechanically homogeneous modules. A module weights 0.25 kg and has a length of 94 mm (including the lever) with a cross section of 45x50 mm. YaMoR modules have 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 a powerful RC servomotor with maximum rotation speed of 60°/0.16 s and a maximum torque of 0.73 Nm, which is sufficient for one module to lift three others. The casing of a module consists of printed circuit boards (PCB) that also serve as support for the electronics (Figure 1). Modules can be attached to each other at specific locations on each module face. A screw and pin mechanism allows fixations with angles at every 15 degrees. In its current state, YaMoR modules do therefore not support self-reconfiguration and the modules can only be connected together by hand.

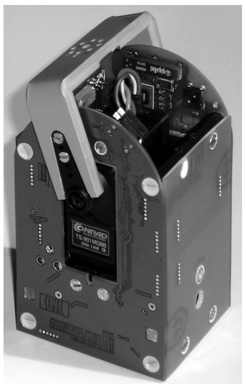

Figure 1: YaMoR module.

A module is powered by an onboard Li-Ion battery and includes the necessary electronics for power management, motor control, communication and execution of algorithms. To achieve more flexibility and modularity in terms of control each YaMoR module contains five separated control boards: (1) one board for handling the Bluetooth communication; (2) one board carrying an ARM microcontroller, (3) one board carrying a Spartan-3 FPGA; (4) one sensor-board with an infrared distance sensor and a 3D accelerometer; and (5) a service board containing power supply and battery management. YaMoR was constructed as a framework for a variety of different projects. For instance, a user may choose between using an ARM7TDMI microcontroller—used in a variety of industrial and research projects—an FPGA or a combination of both for implementing the desired control algorithm. Configuring the FPGA to contain a MicroBlaze softprocessor (Xilinx Corp.), allows exploiting the hardware-software co-design capabilities offered by the platform, taking also advantage of the flexibility provided by the FPGAs partial reconfiguration feature (Upegui et al., 2005). The YaMoR architecture with distributed electronic components gives a flexible solution for connecting the electronic boards: the FPGA board can be left out if it is not needed to save energy; or it can be replaced by a board with specific sensors if useful. The new sensor board can still take advantage of the electronics mounted on the remaining boards. So a designer for an additional sensor board does not have to worry about power supply or battery management. In this article, we only use the power board, the Bluetooth board and the microcontroller board. For more details about the YaMoR modules see Moeckel et al. (2006).

3.2 Bluetooth communication protocol

Designing the communication network for robots can be a great challenge, especially when the mechanical connections are changing during the operation of the robot and when the robot consists of different modules that might provide different services like different sensors, computational units and actuators that should work together. Within the field of self-reconfiguring modular robots it is a strong advantage to have a communication infrastructure that is independent of the mechanical structure of the robot and that after it is set up once is continuously working also during the reconfiguration of the mechanical system. For this reason we decided to use a wireless communications network for our modular robot YaMoR that allows us to provide a stable exchange of information between the modules also during periods of (manual) reconfiguration. The CPG model that will be presented in Section 4 requires modules to communicate their states to each other such as to implement a system of coupled oscillators. Furthermore, a wireless communication is advantageous when commands have to be sent to the robot e.g. from a PC or data should be tracked and analyzed. In this case the robot can operate without being disturbed by a tether.

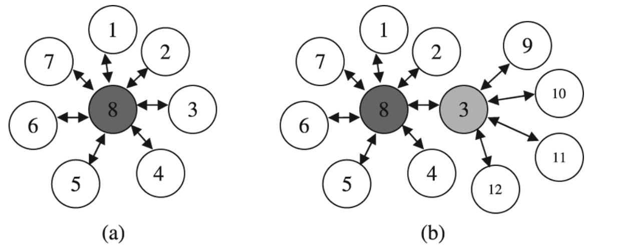

Figure 2: (a) Piconet with master device number 8 and 7 slaves (number 1-7). (b) To form a Bluetooth network with more than 8 actively connected devices, a Scatternet has to be created. There for device number 3 becomes a master/slave device. It acts like a slave for the original Piconet with master number 8 and like a master device for the devices number 9-12.

3.2.1 Advantages of Bluetooth

We decided to use Bluetooth for our modular robot as it is a powerful standard for robust wireless networks that has several benefits: (1) Implementations of the latest Bluetooth standard 2.0 reported power consumptions of more than 10 times less than for WLAN. That is why Bluetooth is very interesting for embedded systems that are battery powered and there for have limited energy resources. (2) Bluetooth is using a very robust communication protocol called Frequency Hopping. (3) In contrast to infrared, Bluetooth devices do not have to be in the visual range of each other to support a wireless link. (4) Bluetooth is a standard that makes sure that every certified Bluetooth device regardless which company was developing it can communicate with every other certified Bluetooth device from any other company. (5) Bluetooth is supported and continuously improved by a group of companies organizing themselves in the so-called Bluetooth interest group. (6) Because Bluetooth devices are manufactured in high numbers and due to the fact that Bluetooth is operating in the license-free frequency band the price for Bluetooth devices is very low. (7) Many Bluetooth software stacks are available; some of them even for free. That is why the time for the development of new Bluetooth software and protocols is dramatically reduced.

3.2.2 Drawbacks of Bluetooth

Although free Bluetooth software stacks are available, the development of software based on Bluetooth can be difficult. This has two main reasons: (1) For interfacing the Bluetooth stack, the user still needs quite a lot of knowledge about the wireless standard. (2) Because Bluetooth was originally designed as a cable replacement with a central host device Bluetooth networks have some major restrictions: As shown in Figure 2a up to 8 Bluetooth devices can be connected to form a so-called Piconet. In this Piconet one Bluetooth device acts as the central host device called master while the other 7 devices act as slaves. The master is controlling the whole communication in the Piconet. Only the master device knows about all slaves and can send data to them but the slaves do not know which other slave devices the Piconet contains neither they are able to communicate directly with each other. If a slave device wants to send data to another slave it has to pass that data to the master device first which has to forward the data to the receiver. The network structure becomes even more difficult if more than 8 devices shall be connected. In this case as shown in Figure 2b a slave device has to act as a master in another Piconet. By acting as a slave device in one Piconet and as a master in another Piconet a so-called Scatternet is formed. In this Scatternet only directly connected devices know about each other and can directly send data to each other.

3.2.3 Scatternet Protocol

To overcome both problem (1) and (2) we developed a new Scatternet protocol (SNP) layer that extends the original Bluetooth communication, e.g. makes it usable for more than 8 Bluetooth devices in a transparent manner. A user who wants to send data to any other device in a Bluetooth network simply sends a packet with the address of the receiver into the network. The Scatternet protocol is responsible for finding the shortest path through the network and for guaranteeing that the packet is received by the target device. When the network is changed the Scatternet protocol is adapting and learning new paths. Only local information that the Scatternet protocol extracts from the data packets that are passing the Bluetooth device it is running on are used to find the shortest path. The Scatternet protocol also supports broadcasts and gives full remote control for all connected Bluetooth devices. This allows a user to control all Bluetooth devices in a Scatternet from a single device. The user of our powerful Bluetooth-on-chip system only needs to know three commands: (1) How to send data. (2) How to connect devices. (3) How to disconnect devices. For a full description of our additional implemented features see Moeckel et al. (2007).

3.2.4 Implementation of the Scatternet protocol

We implemented and tested our Scatternet protocol on the embedded Bluetooth device ZV4002 which was developed and distributed by Zeevo Inc. The ZV4002 contains both the analog Bluetooth components as well as an ARM microcontroller and only needs a small amount of external components including an antenna, SRAM and flash memory. Zeevo provides an embedded Bluetooth stack that is directly running on the ARM. We implemented our Scatternet protocol layer on top of the serial port profile. With this strategy we are able to provide a full Bluetooth Scatternet protocol encapsulated in a single chip. The embedded Bluetooth stack is certified for the Bluetooth standard 1.2 and provides data rates of up to 721 kBit/s.

Data and commands can be sent to the ZV4002 via a serial interface with a rate of up to 921600 Baud. However, since most UARTs of PCs and microcontrollers do not support such a high data rate we had to reduce the baud rate to 115200 for the embedded microcontroller that is running the CPGs and controlling the servomotors. We tried to reduce the overhead of our SNP layer to a minimum. The additional header for SNP packets comprises 5 bytes for the communication between Bluetooth devices and 3 bytes for sending a packet to a Bluetooth device via UART. Additional transmission of internal packets that do not include user defined data but are necessary to provide a transparent Bluetooth communication are reduced to a minimum and normally only come into play when a new communication network has to be set up or if a communication network is changed. Furthermore, we implemented friend tables that store 15 Bytes of data for each Bluetooth device that is known. Up to 255 Bluetooth devices can be stored.

Bluetooth traffic for experiments We tested our SNP with modular robot configurations of up to eight Bluetooth (BT) nodes/YaMoR modules, plus one additional node for the connection to the serial port of the PC. We are using a ZV4002 BT chip providing data rates up to 75 kByte/s (Bluetooth 1.2). It can be assumed that the actual possible data rate is smaller, message delay tests are described in Moeckel et al. (2007).

Constant communication messages between oscillators are synchronization messages based on sparse communication (see Section 5.1.1). Each message has 23 Byte and is sent four times a second to each of the direct neighbors in the CPG network (Figure 18).[3] Our biggest modular robot assembly, the quadruped robot, features outer modules with one neighbor and inner modules with three neighbors. The traffic going through a single inner BT node is therefore $6 \times 4/\text{ s} \times 23$ Byte ≈ 0.5 kByte/s, for the overall network $\approx$ 1.5 kByte/s.[4] Additional traffic is produced when new CPG parameters are sent via the BT node at the PC (e.g. parameter message: 77 Bytes) or commands are sent to switch on/off the RC motors. However in comparison to the communication due to the synchronization messages these are small peaks in the BT traffic. The biggest reduction for BT traffic is clearly provided by sparse communication: only each 25^{th} synchronization message is actually sent. A simple computation for synchronization messages, but without sparse communication, reveals that the serial line between microcontroller and BT node (based on 115200 baud rate, approximately 11.5 kByte/s) would actually overload (13.2 kByte/s load) without this feature.

[3]This corresponds to a communication time step of 25, see Section 5.1.1.

[4]Assuming direct communication. In case the messages are sent indirectly the load is higher than 1.5 kByte/s, but still lower than the max. possible 75 kByte/s.

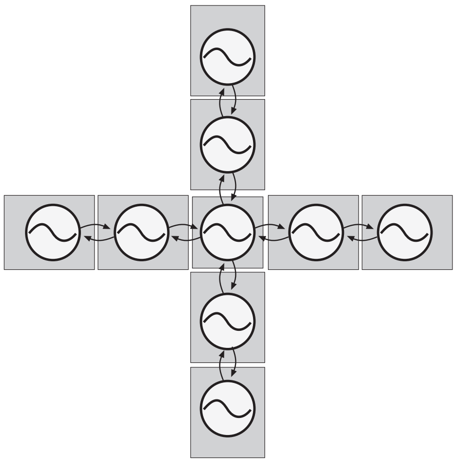
Figure 3: Example of a CPG architecture.

4 Locomotion control

The locomotion controller is composed of a CPG model for producing coordinated oscillations and of an optimization algorithm, Powell's method, for optimizing the speed of locomotion.

4.1 CPG model

The CPG model is implemented as a system of N coupled amplitude-controlled phase oscillators, one per module (Figure 3). An oscillator i is implemented as follows:

$$\begin{aligned}\dot{\phi}_i &= \omega_i && (1)\\ &+ \sum_j w_{ij}\, r_j \sin(\phi_j - \phi_i - \varphi_{ij}) \\ \ddot{r}_i &= a_r\left(\frac{a_r}{4}(R_i - r_i) - \dot{r}_i\right) && (2)\\ \ddot{x}_i &= a_x\left(\frac{a_x}{4}(X_i - x_i) - \dot{x}_i\right) && (3)\\ \theta_i &= x_i + r_i \cos(\phi_i) && (4)\end{aligned}$$

where θ_i is the oscillating set-point (in radians) extracted from the oscillator, and ϕ_i, r_i, and x_i are state variables that encode respectively the phase, the amplitude, and the offset of the oscillations (in radians). The parameters ω_i, R_i, and X_i are control parameters for the desired frequency, amplitude and offset of the oscillations. The parameters w_{ij} and φ_{ij} are respectively coupling weights and phase biases which determine how oscillator j influences oscillator i. An oscillator i receives the value of state variables ϕ_j and r_j of neighbor modules j via the Bluetooth communication protocol (see the next Sections). The parameters a_r and a_x are constant positive gains ($a_r = a_x = 20$ [rad/s]). The reference position (i.e. corresponding to a zero offset) is the position in which the lever of the YaMoR module is aligned with the module (i.e. in the middle of its working range).

These equations were designed such that the output of the oscillator θ_i exhibits limit cycle behavior, i.e. produces a stable periodic output. Equation 1 determines the time evolution of the phases of the oscillators. In this article, we use the same frequency parameter $\omega_i = \omega$ for all oscillators, and bidirectional couplings between oscillators such that $\varphi_{ij} = -\varphi_{ji}$. We also ensure that, in case of a closed loop of interoscillator couplings, all phase biases in the loop are consistent (i.e. sum up to a multiple of 2π). With these parameters, the phases will converge to a regime in which they grow linearly with a common rate ω and with a phase difference between oscillators determined by φ_{ij}. Equations 2 and 3 are critically damped second order linear differential equations which have respectively R_i and X_i as stable fixed points. From any initial conditions, the state variables r_i and x_i will asymptotically and monotonically converge to R_i and X_i. This allows one to smoothly modulate the amplitude and offset of oscillations.

With these settings, two oscillators i and j that are coupled with non-zero weights w_{ij} asymptotically converge to limit cycles $\theta_i^\infty(t)$ and $\theta_j^\infty(t)$ that are defined by the following closed form solutions:

$$\theta_i^\infty(t) = X_i + R_i \cdot \cos(\omega t + \varphi_{ij} + \phi_0) \quad (5)$$
$$\theta_j^\infty(t) = X_j + R_j \cdot \cos(\omega t + \varphi_{ji} + \phi_0) \quad (6)$$

where ϕ_0 depends on the initial conditions of the system.[5] This means that the system stabilizes into oscillations that are phase-locked for all degrees of freedom that are connected together. These oscillations can be modulated by several control parameters, namely ω for setting the common frequency, φ_{ij} for setting the phase lags between two connected oscillators, R_i ($i = 1, 2, ..., N$) for setting the individual amplitudes, and X_i ($i = 1, 2, ..., N$) for setting the individual offsets. Figure 4 illustrates how the system converges to the stable oscillations starting from random initial conditions and after a random perturbation.

Similar CPG models have been developed, for instance to simulate the lamprey swimming network (Cohen et al., 1982; Williams et al., 1990; Kopell et al., 1991; Nishii et al., 1994; Sigvardt and Williams, 1996). The closest model used to control a robot is the one developed by Conradt and Varshavskaya (2003), an important difference being that our model has differential equations controlling the amplitudes of each oscillator (not only the phase). We used (almost)

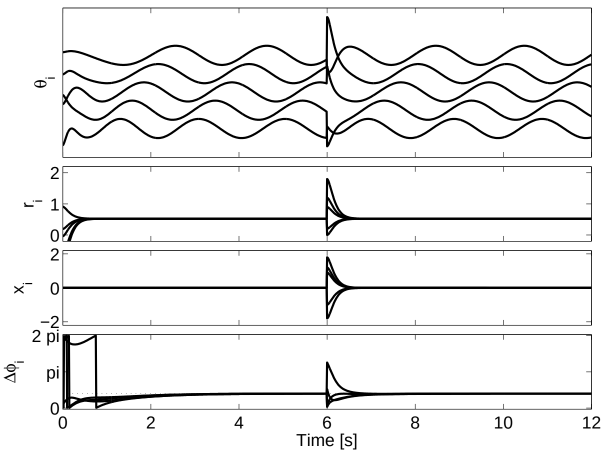

Figure 4: Limit cycle behavior of the CPG. An arbitrary chain structure of 5 oscillators is chosen for demonstration. Output signals θ_i of the 5 oscillators (top). Amplitude state variables r_i (second from top). Offset state variables x_i (third from top). Phase differences $\Delta\phi_i = \phi_{i+1} - \phi_i$ between neighbor oscillators (bottom). Starting from random initial conditions, the system quickly stabilizes in synchronous oscillations with controlled amplitude. At $t = 6s$, random perturbations are applied to the state variables ϕ_i, r_i and x_i, and the system rapidly returns to the steady state oscillations.

the same model to control a snake-like robot (Ijspeert and Crespi, 2007) and a salamander-like robot (Ijspeert et al., 2007).

Each oscillator is implemented locally in the microcontroller of each module. The differential equations are solved using Euler integration with a 10 ms time steps. The setpoints θ_i are sent to the servomotors using pulse-width modulation (PWM).[6] Oscillators that are coupled together (i.e. that have non-zero weights w_{ij}) will regularly exchange their state variables ϕ_i and r_i via Bluetooth for implementing the system of coupled oscillators. As will be analyzed in Section 5, the communication of state variables can take place at a slower time scale than every integration step.

In this article, we will carry out experiments in which the frequency ω and the connectivity between oscillators (i.e. the weights w_{ij}) are fixed in advance. The phase lags φ_{ij}, the amplitudes R_i and the offsets X_i will be set by the optimization algorithm in order to explore different locomotor gaits. These parameters are changed at each iteration (every 23 s) of the optimization algorithm.

Such a CPG model has several nice properties. The first interesting property is that the system exhibits limit cycle behavior, i.e. oscillations rapidly return to the steady-state oscillations after any transient perturbation of the state variables (Figure 4). The second interesting property is that this limit cycle has a closed form solution. Most types of oscillators used do build CPGs (e.g. Matsuoka, Van der Pol,

[5]Note that, because of the common frequency and the consistency of phase lags in a loop, this limit cycle does not depend on the coupling weights w_{ij}. The coupling weights only affect how quickly the system converges to the limit cycle: the larger the weight, the faster the convergence.

[6]The set-points θ_i are being directly interpreted as angular positions by the RC servomotors. We use digital position-controlled RC servomotors (hobby type). The internal PID controller of the RC servomotors is encapsulated and not available from the outside, that is we have no influence on the gain of the control loop.

Stein, FitzHugh-Nagumo, Raleigh oscillators) do not have a closed form solution for their limit cycle, see Buchli et al. (2006b) for a discussion of the pros and cons of different oscillator models. The limit cycle has a harmonic shape and has control parameters (ω, R_i, and X_i) that are explicit and are directly related to relevant features of the oscillations. This facilitates the design and analysis of locomotion controllers. A third interesting property is that these control parameters can be abruptly and/or continuously varied while inducing only smooth modulations of the set-point oscillations (i.e. there are no discontinuities nor jerks). This property is important to avoid damage in the motors and gearboxes, and will extensively be used in the Results section for the online optimization of the locomotor behaviors (Section 5). Finally, a fifth interesting feature is that feedback terms can be added to Equations 1—3 in order to maintain entrainment between control oscillations and mechanical oscillations (however this will not be explored in this article).

4.2 Optimization algorithm

We shall now explain how the CPG is optimized in order to produce fast locomotion. The function to optimize, the cost function $f(\boldsymbol{x})$, is the average speed of the modular robot as estimated from an external camera. The vector $\boldsymbol{x}$ contains all the control parameters of the CPG. We measure the speed of one specific module, by measuring the distance traveled in a given period (see the description of the experimental setup in the next Section).

To do the optimization, we require an algorithm that can compute optima of a function without gradient information (which is not available here). Since time to convergence is critical in online optimization we decided to avoid stochastic optimization methods such as simulated annealing, genetic algorithms and particle swarm optimization and to use Powell's method, a fast heuristic optimization algorithm (Press et al., 1994). Powell's method is easily available and well documented and is therefore simple to use and compare. It is recommended (Press et al., 1994) in comparison to other gradient-free methods. The advantage of the algorithm is that it is fast, the disadvantage is that it presents more risk to converge to a local optimum (as opposed to a global optimum) than stochastic methods.

The principle of the algorithm is to use Brent's method, a bracketing algorithm, for performing one-dimensional optimization and then to use a direction-set method for multi-dimensional optimization. We briefly explain the algorithm here. Our description is inspired from the one found in Press et al. (1994).

One dimensional optimization The goal of function optimization is to find x such that $f(x)$ is the highest or lowest value in a finite neighborhood. From now on we just consider the problem of function minimization. Note that function maximization is trivially related because one can minimize $-f$. The main idea of one-dimensional function optimization is to bracket the minimum with three points

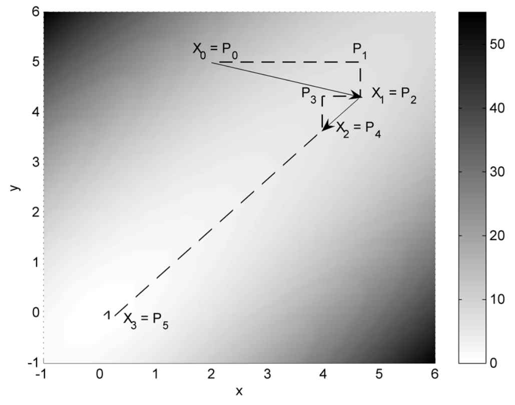

Figure 5: Example of function optimized with Powell's method.

$a < b < c$ such that $f(b)$ is less than both $f(a)$ and $f(c)$. In this case and if f is nonsingular, f must have a minimum between a and c. Now suppose that a new point x is chosen between b and c. If $f(b) < f(x)$, the minimum is bracketed by the triplet (a, b, x). In the other case if $f(x) < f(b)$, the new bracketing points are (b, x, c). In both cases, the bracketing interval decreases and the function value of the middle point is the minimum found so far. Bracketing continues until the distance between the two outer points is tolerably small (Press et al., 1994)[7]. The challenge is finding the best strategy for choosing the new point x in the bracketing interval at each iteration. Powell's algorithm uses Brent's method, which is a clever combination of golden section search and parabolic interpolation (Brent, 1973; Press et al., 1994).

Multi-dimensional optimization Consider a line defined by a starting point $\boldsymbol{P}$ and a direction $\boldsymbol{n}$ in N-dimensional space. It is possible to find the minimum of a multidimensional function f on this line using a one-dimensional optimization algorithm (Press et al., 1994) (e.g. Brent's method, see above). Direction-set methods for multidimensional function minimization consist of sequences of such line minimizations. The methods differ by the strategies in choosing a new direction for the next line minimization at each stage. Powell's method (Brent, 1973; Press et al., 1994) is best explained with an example. Consider a function with a 'valley' along $x = y$ that descends to the origin:

$$f(x, y) \quad = \quad x^2 + y^2 + (x - y)^2 \qquad (7)$$

Powell's method starts with the unit vectors $\boldsymbol{e_1}$, $\boldsymbol{e_2}$, ..., $\boldsymbol{e_N}$ of the N-dimensional search space as a set of directions. One iteration of the algorithm does N line minimizations along the N directions in the set. The algorithm is illustrated in Figure 5 for the two-dimensional function introduced above (Eq. 7). Starting at the initial point $\boldsymbol{P_0} = (2, 5)$, the first line minimization along the direction given by the

[7]Tolerance values used in all our experiments are set to $tol_{\text{Brent}} = 0.05$ and $tol_{\text{Powell}} = 0.02$.

unit vector $[1, 0]^T$ takes us to the point $\boldsymbol{P_1}$. From this point the second line minimization along $[0, 1]^T$ takes us to $\boldsymbol{P_2}$ and completes the first iteration. As you can see on Figure 5, repeated line minimizations along the unit vectors would involve many iterations because the minimum would be approached in small steps. After each iteration, Powell's method checks if it is beneficial to replace one of the directions in the set by $v_i = \boldsymbol{P_0} - \boldsymbol{P_N}$ where $\boldsymbol{P_0}$ was the starting point at the current iteration and $\boldsymbol{P_N}$ the new point after the N line minimizations. In the example of Figure 5, v_2 replaces $[1, 0]^T$ in the second iteration. The algorithm correctly decides not to include new directions in all other iterations as this would actually slow down convergence. The mechanisms for deciding whether or not to include the new direction v_i after each iteration and which direction in the set should be replaced are described in (Brent, 1973; Press et al., 1994). Note that there is no learning rate; the algorithm simply always goes to the optimum in the next direction.

5 Results

In this Section, we will first evaluate the suitability of the CPG model for a distributed implementation with possibly unreliable communication between modules. We will then present results of running the CPG and Powell's method on three different types of robot structures.

5.1 Suitability of CPGs for modular robots

In order to evaluate the suitability of the CPG model, we tested it with Matlab on a PC. We tested one particular configuration: a chain of $N = 5$ oscillators with bidirectional couplings. Unless otherwise specified, the system was implemented with the following parameter values for all oscillators: $\omega = \pi/2$ [rad/s], $a_r = 20$ [1/s], $a_x = 20$ [1/s], and $w_{ij} = 5$. The system of equations was integrated using the Euler method with 10 ms integration steps (like in the robot experiments of the next sections).

Figure 6 illustrates how such a system reacts to abrupt changes of the amplitude R_i, offset X_i, and phase lag φ_{ij} parameters. The system starts with random initial conditions and rapidly stabilizes in a traveling wave ($\varphi_{ij} = 2\pi/5$ for all connections in one direction and $\varphi_{ij} = -2\pi/5$ for all connections in the other direction). At time $t = 10$ s, the signs of the phase lag parameters are changed and the system stabilizes after a short transient period into a wave traveling in the opposite direction. At time $t = 20$ s, the amplitude parameters R_i are modified, and the amplitudes of oscillations change accordingly. At time $t = 30$ s, the offsets X_i are set to non-zero values. After all these abrupt changes, the outputs θ_i of the oscillators smoothly converge to the new limit cycle. This is an important feature since this limits the risk of damaging motors and or gears due to abrupt changes in motor commands. This feature will allow us in the next sections to run an optimization algorithm in parallel to the CPG. The duration of the transient periods before convergence depend on the parameters a_r, a_x, and w_{ij}: the lower these values, the slower the convergence, see Figure 7.

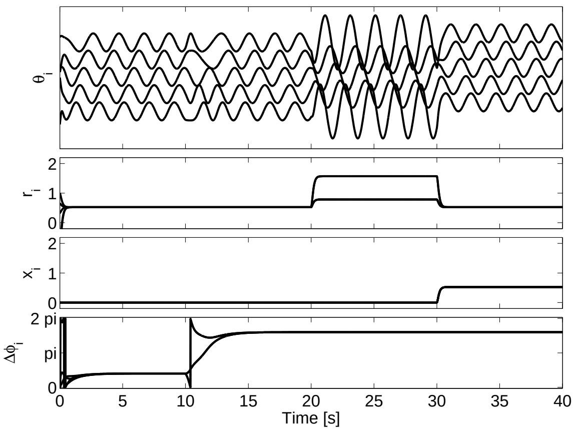

Figure 6: Example of abrupt parameter changes in the CPG model. Output signals θ_i of the 5 oscillators (top). Amplitude state variables r_i (second from top). Offset state variables x_i (third from top). Phase differences $\Delta\phi_i = \phi_{i+1} - \phi_i$ between neighbor oscillators (bottom). See text for explanations.

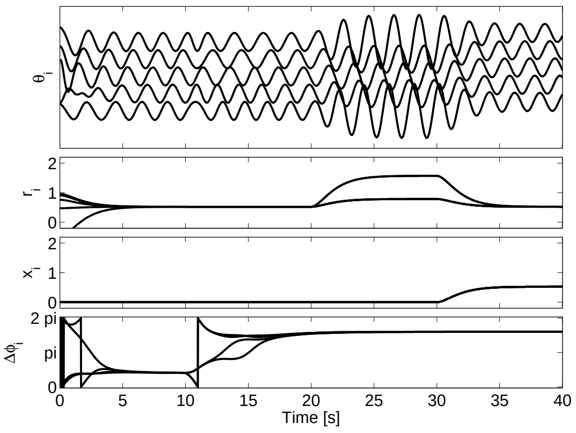

Figure 7: Example of abrupt parameter changes in the CPG model. Same experiment as in Figure 6, but with $a_r = 2$ [1/s], $a_x = 2$ [1/s], and $w_{ij} = 2$.

5.1.1 Sparse communication and temporary loss of connection

The distributed implementation of the CPG in separate YaMoR modules that communicate via Bluetooth implies several potential problems in the communication between oscillators. These potential problems include sparse communication, that is, the fact that states of neighboring oscillators can not be communicated as fast as the numerical integration, temporary loss of connection, and time delays due to hops in the Scatternet.

We tested these different potential problems in Matlab to evaluate how much they could affect the activity of the CPG.

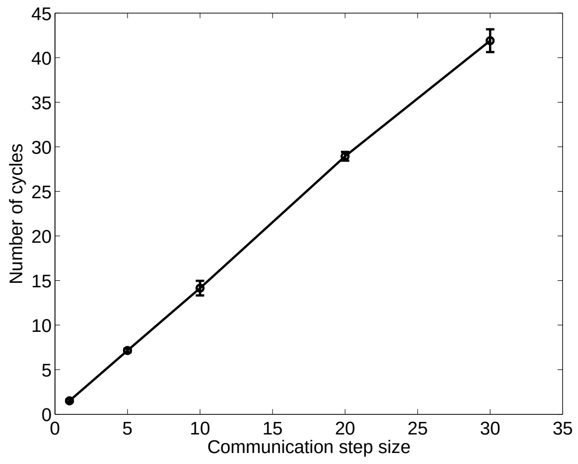

Figure 8: Sparse communication in a system with an oscillation period of 2 s. Time to reach the limit cycle depending on the communication time step size. The communication time step size is given as multiples of the integration time step (10 ms). The time to convergence is measured as the number of cycles needed for all oscillators to come within 0.1 rad of the desired phase lags between each other. Data points and error bars correspond respectively to the average and standard deviation of 5 trials with random initial conditions.

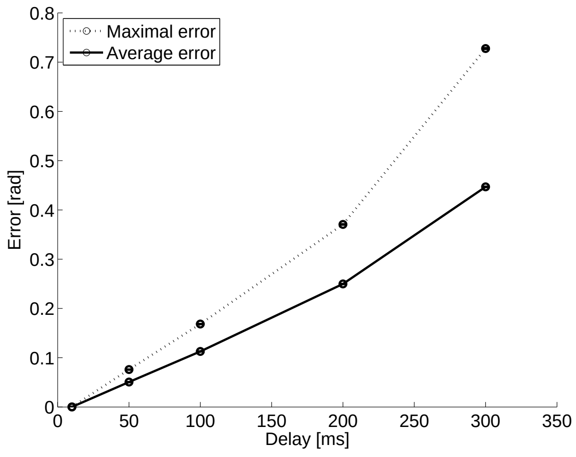

Figure 9: Delayed communication. Average and maximal difference between the desired phase lags between oscillators and the actual ones after running the system for 50 s as function of the time delay of the communication. Data points and error bars correspond respectively to the average and standard deviation of 5 trials with random initial conditions.

The sparse communication is tested by communicating the state variables of oscillators with a *communication time step* that is larger than the integration step. At each communication time step, oscillator i receives the state variables r_j and ϕ_j from oscillator j. There are several options in how to deal with the coupling terms during the intervals between each communication time step: (1) one can set r_j to zero when there is no communication, (2) one can keep the old r_j and ϕ_j values as constants during the interval, and (3) one could try to interpolate these values based on past values. We tested options 1 and 2.

Figure 8 illustrates the time needed to reach convergence with option 1 as a function of the communication time step size. The communication time step size is given as multiples of the integration time step. The time to convergence is measured as the number of cycles needed for all oscillators to come within 0.1 radians of the desired phase lags between each other. The test shows that time to convergence increases more or less linearly with the communication time step size. This means that the system will converge to the desired oscillatory pattern even if the communication between modules takes place only every so often. Note that the cycle duration of the illustrated system is 2 s, which means that the maximum communication step of 30 times the integration step corresponds to a communication only every 300 ms (i.e. less than 7 times per cycle).

With option 2, the system will converge to a steady state regime much faster than with option 1 (data not shown), but with the problem that the steady regime does not exactly correspond to the desired oscillatory pattern. Indeed the phases in the steady state regime do not increase linearly, which means that the phase differences between oscillators vary over time and that the oscillation signals are not pure sinusoids. In practice this becomes noticeable only for large communication steps (30 integration steps or above, we are using option 2 with 25 integrations steps in our hardware experiments).

5.1.2 Communication delays

The Scatternet underlying the communication between different modules introduces time delays in the transmission of the state variables between coupled oscillators. We tested the same network as above (a chain of 5 oscillators with bidirectional couplings) with a 10 ms integration step and a 50 ms communication step. In addition to the sparse communication, we added delays in the transmission by sending state variables that are old by some time duration.

Figure 9 shows that time delays deform the patterns of oscillation but in a gradual fashion (using option 1 from the previous Subsection). To measure the deformation, we measure the average and maximal difference between the desired phase lags between oscillators and the actual ones after running the system for 50 s.

The results show that the error slowly increases with time delays up to the very large delays tested (300 ms). Time delays in the Scatternet with multiple hops are typically smaller. The time delays due to the Bluetooth communication should therefore lead to negligible effects on the CPG activity.

5.2 Experimental setup with the YaMoR robots

We tested our approach on the actual YaMoR modules with three different robot structures. The experimental setup is

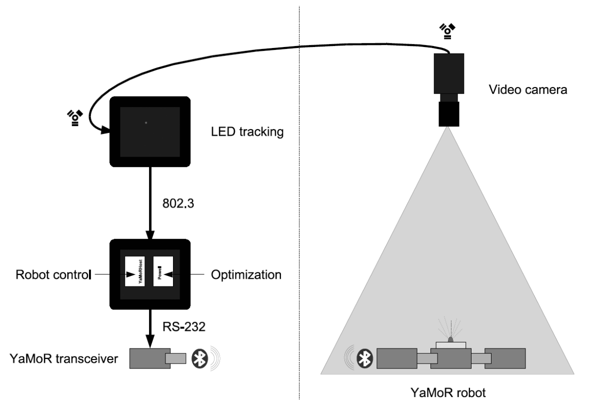

Figure 10: Experimental setup. The YaMoR robot configuration is moving in a 2m by 2m area, an attached LED is being tracked by a camera on the ceiling. X and y positions are extracted both for the starting and finishing point of the evaluation. The optimization is running on a PC, sending a new set of CPG parameters to the robot configuration after each re-evaluation.

shown in Figure 10. The robots are tested in a square arena of 2m by 2m. The speed of the robot is measured by tracking a red LED fixed to one of the robot modules by a camera fixed to the ceiling. The x and y positions of the LED are measured in real-time, and provided to a PC that runs the Powell's method. The evaluation of the speed of the robot is done as follows. Each time the parameters of the CPG are updated and sent from the PC to the modules (this takes 8 seconds e.g. in case of the quadruped robot), we wait for 7 seconds to give the oscillators and the robot time to go into steady-state locomotion, and then let the system move for 8 seconds (evaluation window). The estimated speed is the distance between the end and beginning positions during the evaluation window divided by 8 seconds. In other words, we optimize the capability to move as far as possible from the start of the measure but without specifying a preferred direction. If needed, the robot is manually moved towards the center of the arena at the end of the evaluation in order to maintain it in the field of view of the camera.

The different relevant time step sizes are the following: the integration time step is 10 ms, the communication time step is 250 ms, and the optimization time step is 23 s. Experience showed that the evaluation is quite noisy and that it is not beneficial to use a precision of more than 0.05 radians for Brent's method (Powell: 0.02). Using this precision, a line minimization over an interval of 2π (largest possible bracketing interval for a phase difference) involves less than 15 speed evaluations (generally between 5 and 10). An iteration of Powell's method consists of N line minimizations, where N is the number of parameters. Therefore, it takes in the order of 10 N fitness evaluations for one iteration of Powell's method (N is between 2 and 6 for the configurations that we tested, see next sections).[8]

In this article, a PC runs the optimization algorithm. In contrast to locomotion control, a distributed approach is not essential because the optimization algorithm has negligible computational cost (compared to integration of the nonlinear oscillators) and involves little communication (sending new parameter values to all modules every 23 seconds).

[8]We tested structures with up to N=30 in simulation, see Marbach and Ijspeert (2005)

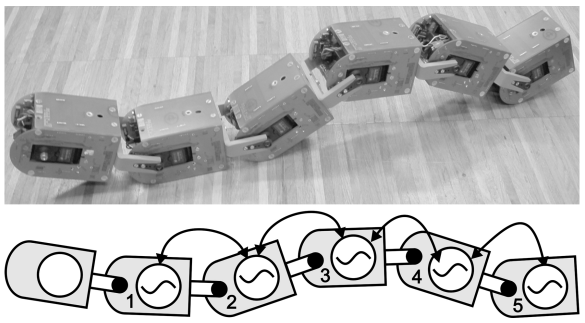

Figure 11: Structure of the snake robot (top) and its CPG (bottom). Oscillators are coupled bidirectionally between each other (thick solid line).

5.3 Snake robot

The first robot structure in which we tested our approach is a snake robot made of five active modules, plus one inactive module, with all axes of rotations in parallel (Figure 11). The CPG network is a chain of 5 coupled oscillators with bidirectional couplings. The couplings between oscillators correspond to the mechanical connections. The choice of this robot structure was motivated by the fact that it has several symmetries, which helps to reduce the number of parameters to optimize and allows us to do a comparison with a systematic search over two parameters. In our case, we set all offset parameters X_i to zero and all frequency parameters ω_i to 0.6π. We assume all modules to have the same amplitude and phase lag parameters $R_i = R$ and $\varphi_{ij} = \varphi$. In other words, the CPG is constrained to produce traveling waves whose amplitude and phase lag are determined by R and φ respectively. The following parameters ranges are explored $R \in [0, 0.6]$ and $\varphi \in [0, \pi]$. Higher values for the amplitude parameter can lead to self-collision between the modules.

Figure 12 shows the exploration of the two-dimensional parameter space by the Powell algorithm. Within less than 18 minutes, interesting gaits are found allowing the robot to progress at 2.1 cm/second on average. Note that due to the bracketing method of the optimization algorithms, parameters sometimes jump to sub-optimal solutions and hence the evolution of the speed measurements shows frequent drops. Because this particular problem is relatively simple, only three Powell iterations (first iteration from $t = 0$ min to $t = 8$ min, second iteration from $t = 8$ min to $t = 16$ min, third until end: $t = 23$ min) were needed before convergence.

Snapshots of a learned gait are shown in Figure 13 (see also Extension 2; Extension 1 shows an initial parameter configuration for the snake robot). The gait is a caterpillar-

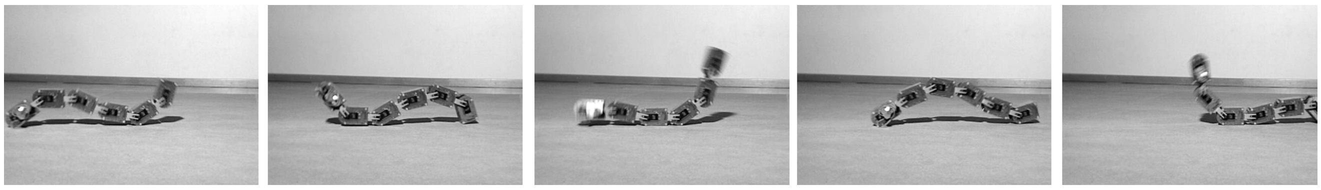

Figure 13: Snapshots of a learned gait in the snake robot. The robot is propelled by a traveling wave going from tail (left end with LED marker) to head.

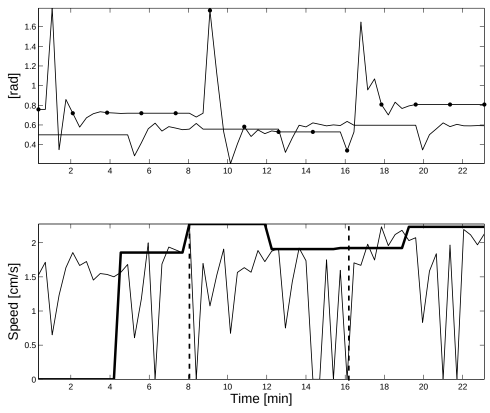

Figure 12: Optimization with the snake robot. Top: variations of the R and φ parameters, line with markers represents the phase lag values (φ), the single solid line the shared amplitude R. Bottom: evolution of the measured speed. Iterations are divided by vertical dashed lines, the thick solid line presents the values for the line minimization using the last available speed value of a parameter optimization. Thin lines represent actual speed measurement points.

like gait in which an undulation of body deformation travels from the robots tail to head. The amplitude of the fastest gait described in (Figure 12) of the oscillations is approximately 34 degrees (0.59 rad), and the phase lag between modules is 46 degrees (0.81 rad). The total phase lag between head and tail is $(N-1)0.81 = 4.05$ rad. This corresponds to a wavelength of $2\pi/4.05 = 1.55$ body lengths. By increasing the upper limit for R further increases of the speed are possible, however for certain R-φ parameter combinations the snake structure can self-collide.[9]

Since the number of parameters is only two, we can compare the results of the optimization with a systematic exploration of the parameter space. We carried out 81 speed measurements in the same ranges as for Powell's optimization, with 9 different equally spaced values for both parameters. As can be observed in Figure 14, the speed function is relatively smooth for this robot, with an optimum around $R = 34$ degrees and $\varphi = 50$ degrees. Six repetitive runs of Powell's method on the two-parameter snake-robot show parameter combinations and speed results in the same region as the maximum of the systematic search (see optimization results in Table 1 and Figure 14).

We regard therefore Powell's method as being equivalent or better than a fine-grid systematic search in the case of a two-parameter search.[10] However unlike Powell's method a systematic search is not scalable when using up to 6 parameters ($9^6 = 531441$ evaluations would be necessary in this case).

As can be seen in Figure 12 and in Table 1 the algorithm may find intermediary speed measurements that are higher than the final value after convergence of the Powell algorithm. This is because of the intrinsic noise in the measure of speed and due to the small time window we use to estimate speed (see also the variance in speed measurement for two gaits: Figure 21). We found out that converged solutions tend to be more robust than intermediary fastest-speed (v_{maxval}) parameter results.

Table 1: Results for 6 experiments, snake robot with 2 parameters. t_{maxval} is the time until first maximal speed values show up, t_{final} the time of convergence of the algorithm.

exp.	v_{maxval}	t_{maxval}	v_{final}	t_{final}
1	2.3 [cm/s]	19 [min]	1.9 [cm/s]	36 [min]
2	2.1 [cm/s]	11 [min]	1.9 [cm/s]	23 [min]
3	2.1 [cm/s]	27 [min]	2.1 [cm/s]	27 [min]
4	2.1 [cm/s]	19 [min]	2.1 [cm/s]	19 [min]
5	2.1 [cm/s]	11 [min]	2.1 [cm/s]	31 [min]
6	2.3 [cm/s]	8 [min]	2.2 [cm/s]	19 [min]

5.4 Tripod robot

We tested the approach on a robot structure with three limbs (Figure 15). The tripod robot is made of 7 modules, one of them being a non-actuated extension of the leg. Similarly to the snake robot, the couplings in the CPG network match the mechanical connections between modules. We assumed a left-right asymmetry, as well as symmetries between limbs. The number of parameters to be optimized is six (see Table 2). The value range for R_i was chosen to prevent the robot from self-collision, X_o covers the range from 0 to 90 degrees. Higher values would make the motors run continuously into their mechanical blocks (RC servomotors are restricted to a 180 degrees range), or the outer modules

[9]Self-collision happens if the snake robot is bent by half a period of its own length, describing a circle.

[10]We are using relatively fine and general tolerance values for Powell and Brent. If needed they can be adapted to a specific structure and optimization, what would reduce the amount of necessary evaluations.

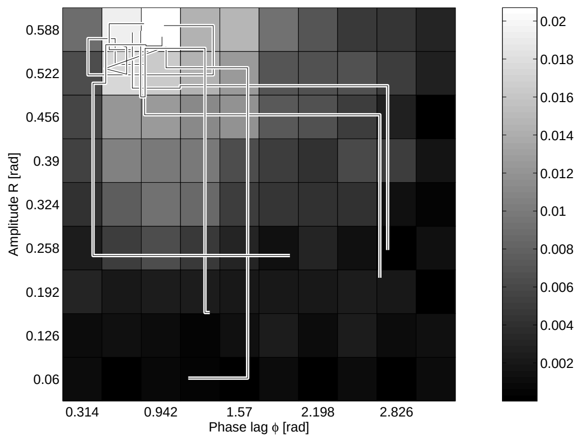

Figure 14: Systematic search with the snake robot. Overlaid are the speed evaluations from experiments using Powell's method in the same parameter range. Final speed values using Powell's method (average over six experiments) is $v = 2.05$ cm/s, the systematic search (one experiment) finds $v = 2.06$ cm/s.

would wiggle continuously in the air. $\varphi_{12} = \varphi_{34} = \varphi_{56}$ describes the phase difference from the inner modules towards the outer modules (e.g. in Figure 15 from module 1 to 2, vice versa $-\varphi_{21}$), φ_{13} and φ_{35} phase lags in between the inner modules. The tripod robot shows for its inner modules a circular mechanical structure. By forming a closed loop the interoscillator coupling rule must be applied:

$$\varphi_{sum} = \varphi_{13} + \varphi_{35} + \varphi_{modulo} \tag{8}$$

Phase lag $\varphi_{modulo} = \varphi_{51} = -\varphi_{15}$ is therefore filled up automatically by the controlling software to close the circle (φ_{sum} being a multiple of 2π).

Table 2: Open parameter description and range for the tripod robot.

Parameter	Description	Range
R_i	Amplitude inner modules	$[0, \pi/4]$
R_o	Amplitude outer modules	$[0, \pi/2]$
X_o	Offset outer modules	$[0, \pi/2]$
$\varphi_{12=34=56}$	Phase lag inner-outer modules	$[0, 2\pi]$
φ_{13}	Phase lag 1 inner modules	$[0, 2\pi]$
φ_{35}	Phase lag 2 inner modules	$[0, 2\pi]$

Figure 16 shows a typical evolution of the six parameters and the corresponding speed measurements. The optimization took three Powell iterations to converge (first iteration from $t = 0$ min to $t = 17$ min, second iteration from $t = 17$ min to $t = 36$ min, dashed vertical lines). Already within 25 minutes interesting gaits are found. The fastest gaits move around 5.5 cm/s, final gaits around 5.1 cm/s. We repeated the experiment six times, maximum speed values are given in Table 3. As for the snake robot, intermediate speed measurements are sometimes higher than the final ones because of the noise in speed measurement. The final

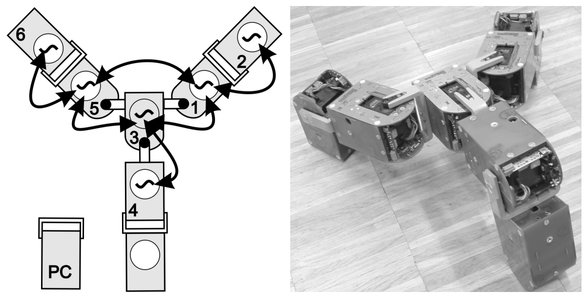

Figure 15: Structure of the tripod CPG (left) and the tripod robot (right). Solid lines show the bidirectional couplings between oscillators. The inner nodes (1, 3 and 5) form a closed loop of interoscillator couplings.

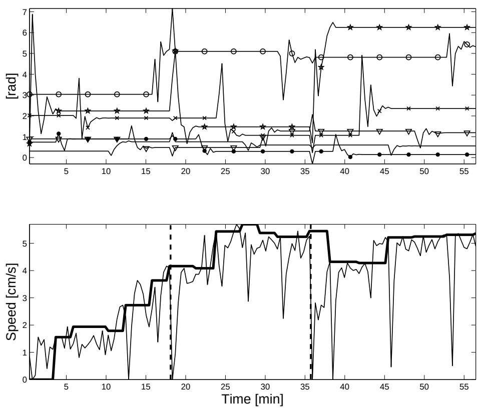

Figure 16: Optimization with the tripod robot. Top: variations of the six network parameters. Bottom: evolution of the measured speed.

parameter combination shows usually a more robust behavior than the first found maximum.

Table 3: Best and final parameter combinations for 6 optimization experiments, using the tripod robot.

exp.	v_{maxval}	t_{maxval}	v_{final}	t_{final}
1	5.5[cm/s]	27[min]	5.1[cm/s]	56[min]
2	6.0[cm/s]	28[min]	4.0[cm/s]	43[min]
3	5.1[cm/s]	41[min]	5.1[cm/s]	41[min]
4	4.5[cm/s]	33[min]	2.8[cm/s]	55[min]
5	4.1[cm/s]	27[min]	5.6[cm/s]	64[min]
6	5.3[cm/s]	28[min]	4.0[cm/s]	93[min]

Snapshots of the best gaits, a crawling gait using the whole body, is illustrated in Figure 17 (see also Extension 4). It uses two legs to pull forward, while the third leg slides behind. At this position the outer modules of the pulling legs points downwards, the third leg has a phase lag such that its outer module points upwards, providing a low friction value. As the two-leg-pulling phase is finished, the third leg points its outer module downwards, fixing the tripod robot against further movement. At the same time the synchronous work-

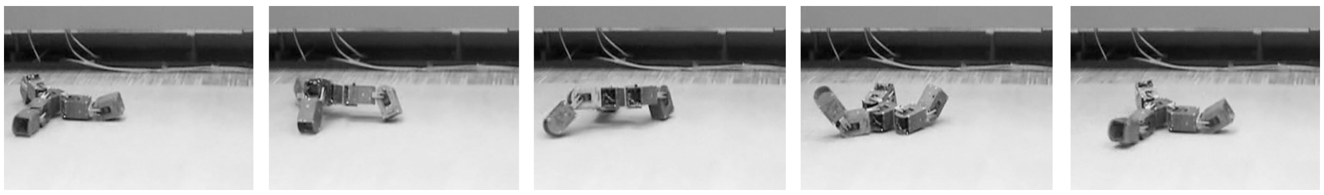
Figure 17: Snapshots of the learned gait in the tripod robot (see also Extension 4; Extension 3 shows an initial gait). The tripod uses two legs in front, and pulls one leg behind.

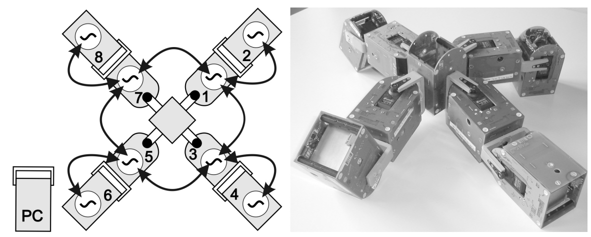

Figure 18: Quadruped robot (right) and its CPG structure (left). Solid lines show the bidirectional couplings between oscillators. The inner nodes (1, 3, 5 and 7) form a closed loop of interoscillator couplings.

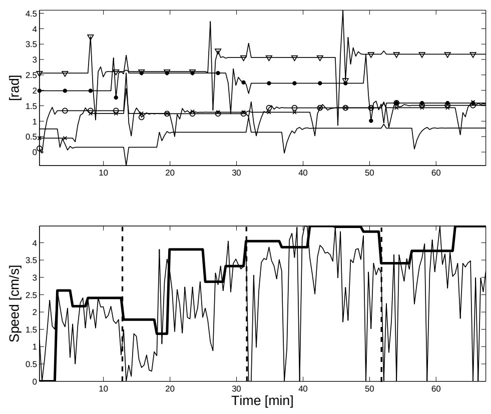

Figure 19: Powell's optimization for the quadruped robot, with five open parameters.

ing two legs point their outer modules in the air. The final parameter distribution is possible in several variations because of several possible symmetry lines for the tripod robot.

During the experiments the tripod robot often developed a gait that was not going straight but slightly in circles, only some gait patterns featured straight forward locomotion. For both cases maximum speed values up to 6 cm/s are reached. Because this gaits can derive from noisy measurements, we normally use more robust parameter combinations provided by the final values of an optimization. For Figure 16 this would refer to gaits in the time range from 55 min until the end of the experiment, rather than the very maximum point at 26 min.

5.5 Quadruped robot

By using an empty YaMoR shell in the center we assembled a quadruped robot (Figure 18) with the general orientation of the degrees of freedom staying the same as in the tripod robot configuration. The CPG structure for the quadruped robot is very similar to the CPG-tripod structure, only the inner circle of oscillators has four instead of three nodes. Each inner node is connected to one outer oscillator via the same phase lag ($\varphi_{12} = \varphi_{34} = \varphi_{56} = \varphi_{78}$).

The quadruped robot configuration introduces one new variable φ_{57} (compared to the tripod configuration). However the symmetry of the quadruped structure gives us the opportunity to actually decrease the amount of open parameters for the optimization as follows. We induce a symmetry for the inner phase relations by fixing the phase lag φ_{13} between two neighboring modules (1 and 3, Figure 18). The same for node 5 and node 7 (φ_{57}). This fixed value now shows up twice, plus one open phase parameter (φ_{35}), and the computed fourth phase value ($\varphi_{71} = \varphi_{modulo}$) for the closed loop ($\varphi_{sum}$ must be a multiple of 2π): $\varphi_{sum} = \varphi_{13} + \varphi_{35} + \varphi_{57} + \varphi_{modulo}$. We set $\varphi_{13} = \varphi_{57} = 0\,\text{rad}$. This "seeded" value for both phase lags makes the corresponding legs move with a right angle in respect to each other at all time.[11] Because only φ_{35} is left open in this configuration, overall 5 open parameters remain to be optimized. The choice for the above introduction of symmetries and the seed value will likely restrict the amount of possible solutions. However the implicit introduction of symmetry for the inner cycle derived from experience with the tripod robot, looking for a fast convergence towards straight gaits. As we can show with the following results, it leads to fast online learning of good gaits for a complex robot structure such as the quadruped robot.

Figure 20 shows an evolved gait (see also Extension 6; Extension 5 shows an initial configuration), where the amplitudes R_i ($R_{1,3,5,7}$) of the inner modules are relatively small. This gait patterns propel the robot in an almost straight line (looking at it over several oscillation cycles). A simplified description of the particular gait looks at two opposing legs: they work as a bridge, with the center of the robot in the middle. By oscillating around their foot contact points two opposing legs bend the rest of the body forward, while the other two legs are mostly in the air. As

[11] An abstraction of the quadruped robot can be two triangles, being connected by one of their tips and moving around this connection point.

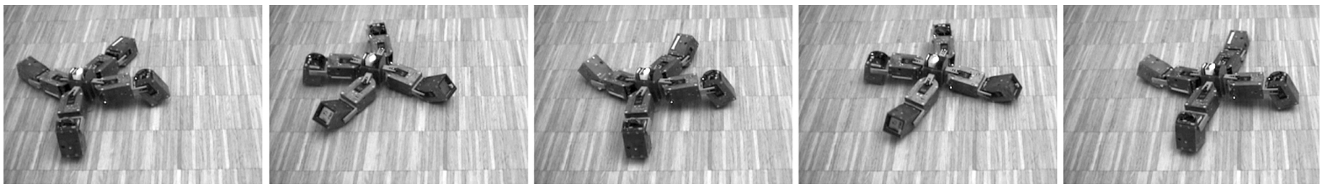
Figure 20: Snapshots of the learned gait in the quadruped robot. The center of the robot describes a zig-zag trajectory in the horizontal layer (with two steps per cycle).

soon as this movement is finished (after half a cycle time), the remaining two legs repeat exactly the same movement, but in a perpendicular direction. What results is a zig-zag gait (two steps over one cycle time) when looking at the center of the robot. All of the evolved gaits during six experiments showed derivations of the above simplified patterns. However several nuances of gaits are found at convergence time. Mainly gaits with higher values for R_i tend to slightly self-turn by small increments (with a global straight motion), whereas gaits with small R_i-values move as in the snapshots.

Similar to the tripod robot the quadruped robot is using not only the tip points of the outer modules for locomotion, but the surface of single or several legs (this depends largely on the values for the outer modules amplitudes, R_o and their offsets X_o). Extreme maximum speed measurement points show patterns that slide single or several legs rather then precisely lifting a leg up, bringing it forward and putting it down again. Complex movement patterns are not predefined in this framework, but can evolve if several degrees of freedom are connected in series (as in case of a leg with two joints) and the optimization method evaluates the derived gait as successful.

In the case of the quadruped robot the mechanical degrees of freedom are never all ideally positioned for straight locomotion because of the star structure. That is why a zig-zag gait is possibly an optimal solution for a quasi-straight gait. During half a cycle period two opposing legs co-work and propel the robot forward, either along the axis colinear to these legs or by an axis perpendicular through the center of the body.

Table 4: Experimental results from six experiments using Powell's method with the quadruped robot, optimization with five open parameters. The first and the last experiment show at the convergence of the algorithm relatively poor results. Good gaits have average speed values of 3.5 cm/s.

exp.	v_{maxval}	t_{maxval}	v_{final}	t_{final}
1	2.7[cm/s]	17[min]	2.6[cm/s]	30[min]
2	3.3[cm/s]	30[min]	3.2[cm/s]	40[min]
3	3.6[cm/s]	15[min]	3.5[cm/s]	32[min]
4	4.5[cm/s]	41[min]	3.5[cm/s]	65[min]
5	3.7[cm/s]	60[min]	3.7[cm/s]	60[min]
6	4.0[cm/s]	25[min]	2.0[cm/s]	95[min]

Table 4 shows a list of speed values resulting from the fastest gait (usually an intermediate point) and the gait at conversion time. The optimization process of optimal gaits seemed more difficult e.g. comparing it to experiments with the tripod robot. We observed that speed values from evolved gaits are smaller than the ones of the tripod robot (about 1.5 cm/s).

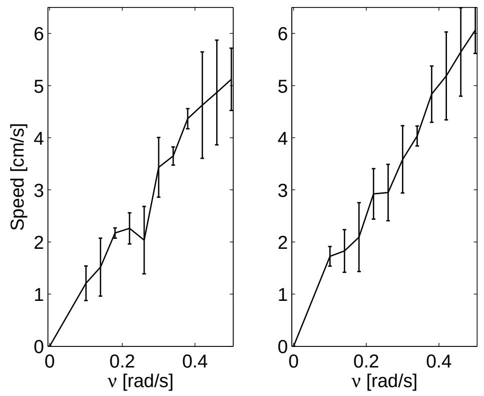

Figure 21: Test of the speed control with the quadruped robot: two of the fastest gaits are used. The frequency of the oscillators is gradually increased from 0.1 [rad/s] to 0.5 [rad/s], the experiment was repeated six times. Data points and error bars correspond respectively to the average and standard deviation of six trials per gait. (left) gait A (right) gait B.

Although chosen with care, our restriction to five open optimization parameters is likely one reason. By opening further degrees of freedom to the optimization algorithm we expect better results. The necessary precise coordination of the four legs and a constantly shifting center of mass (our quadruped robot has no "backbone", hence no implemented mechanical preferred locomotion direction) makes the optimization of such a structure especially challenging.

5.6 Changing frequency

We were interested in exploring how the speed of locomotion can be varied by changing the frequency ν ($\nu_i = \frac{\omega_i}{2\pi}$). Figure 21 shows the speed results obtained with two good gaits selected out of the experiments with the quadruped robot from the previous chapter when ν is changed in the range of [0.1, 0.5] rad/s. Both gaits have approximately the same speed (3.5 cm/s) at $\nu = 0.3$ rad/s. For both gaits, the speed can be adjusted monotonically with the frequency, and interestingly the relation is almost linear in the given frequency range. This means that the frequency is a useful open parameter to control speed. Note that we could not properly explore frequencies higher than 0.5 rad/s because these frequencies draw a lot of power from the YaMoR modules, which regularly result in resetting the microcontrollers of some modules. For higher frequencies, it is expected that speed will at some point saturate and even decrease because of slippage.

6 Discussion

We have developed a framework for learning to move with modular robots using central pattern generators and online optimization. The main contributions of this work are the following: a distributed implementation of a CPG that offers an ideal substrate for producing locomotion patterns and for online learning, a Bluetooth communication protocol for coordinating multiple modules, and an optimization framework for fast learning.

As discussed in the introduction, the CPG model has several interesting features that make it well suited for modular robotics. First, it can readily be implemented in a distributed fashion. As shown in the robotic experiments, each module runs its own nonlinear oscillator. In the YaMoR module, a single oscillator is sufficient because the module has only one degree of freedom, but more could easily be added if there were additional actuators. Second, the distributed network of oscillators is robust against sparse communication and time delays in the communication. Indeed, the transfer from our tests in Matlab to the hardware implementation did not present any difficulty. Furthermore, the synchronization properties of system of coupled oscillators offer interesting features such as robustness against differences in clock frequencies of the multiple microcontrollers involved, as well as the possibility to dynamically add or remove oscillators in the network (e.g. through the addition or removal of modules, see below). Finally, the limit cycle behavior of the CPG allows one to abruptly change parameters in the CPG while the CPG is running. Despite the abrupt changes, the oscillating output signals smoothly converge towards the new oscillation pattern (the new limit cycle) after a short transient period. This feature allowed us to run the optimization algorithm in parallel to the CPG without needing to stop and reset the robot. It is also a feature that is useful for modulating the speed and direction of locomotion by changing frequency and/or amplitude parameters (see Lachat et al. (2006) for instance). Moreover, the smooth modulation of set-point trajectories helped preventing damage in the motor and gearbox of the servomotors. Indeed we did not have any damaged servomotor during all our tests and experiments (more than 40 hours of extensive use in total for some of the modules).

The Bluetooth communication protocol used with the YaMoR modules has greatly simplified the rapid construction and control of different robotic structures, since there is no need to electrically connect modules together. With the help of our Scatternet protocol, Bluetooth provides a transparent communication system that is easy to use. The protocol automatically learns the shortest path through the network and provides broadcast functionality. Via remote control, every other Bluetooth device that belongs to a Scatternet can be controlled from a single device. Wireless communication networks have however the drawback that they are normally less efficient than networks based on wires. The latest Bluetooth 2.0 standard supports data rates up to 2.1 MBit/s which is much smaller than data rates that can be achieved with a communication based on wires. However, with our modular robot we can show that even data rates of 721 kBit/s are sufficient for autonomous control strategies based on CPGs. If higher data rates are needed we propose to use a hybrid communication system. Wired communication can be used while the modules of the robot remain in a fixed mechanical structure, while wireless communication could be used when the robot is in the process of a reconfiguration, between the modules of different modular robots, and/or between a control center like a PC and the modular robots.

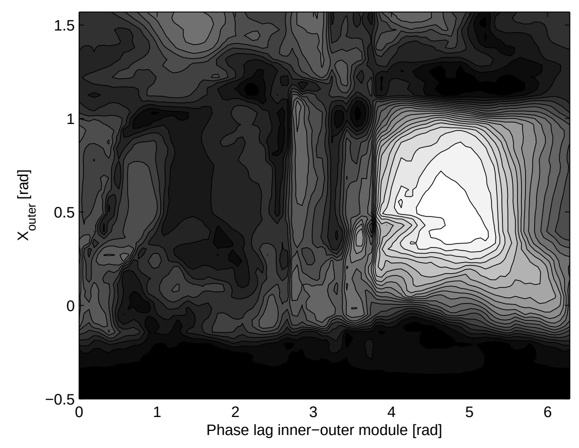

Figure 22: Example of a 2D slice of the search space for our quadruped robot with the above CPG network. The fine-grid systematic search was done in simulation (64x64 data points, cubic interpolation for the plot), work by Yerly (2007). White areas correspond to higher speed, black areas to low speed.

The optimization based on Powell's method has led to interesting gaits in a very short time. The approach is faster than a genetic algorithm by at least one order of magnitude (see comparative tests in simulation by Marbach and Ijspeert (2005)). For all trials with the different robot structures, it managed to significantly improve the speed of locomotion compared to initial random values. The strength of the algorithm is its rapid convergence to a local optimum. Its weakness is that it can relatively easily miss global optima (compared to stochastic methods such as genetic algorithms, particle swarm optimization, or simulated annealing).

In our tests, we however had the good surprise that the algorithm tended to evolve to gaits of the same high quality for different trials with the same robot, and only rarely converged to bad solutions. The likely reason for this is that although the search space presents multiple local optima, the regions close to the global optima are rather large and "flat" (rather than peaked). This is what we observed when we made a very fine-grid systematic search in a 2D slice of the search space of the quadruped robot using a simulator (Figure 22). This means that finding near optimal solutions was not too difficult for the optimization algorithm for these three types of robots and their given search spaces. Of course, it remains to be seen if this is valid for more complex robots and/or larger search spaces, as discussed later.

We are currently extending this work in several directions. First of all we are repeating the experiments with a larger variety of robot structures and more trials per structure. The goal is to better characterize the optimization and to test whether the approach can deal with structures that have more open parameters. While the tests presented here are promising, they were obtained by restricting the size of the search spaces using symmetries in the robot structures. In order to be more generic, it is important to run the optimization algorithm in a larger search space (and to monitor how much time is needed to find interesting solutions) and/or to have the robot modules discovering their symmetries on their own by exploring the topology of the mechanical couplings.

Another experiment that we are carrying out is learning with dynamically changing robot structures, e.g. with the addition or removal of limbs. It is possible to add a mechanism that monitors whether modules are added or removed and/or whether the speed of locomotion drops, and that restarts the optimization algorithm accordingly. We obtained promising results in preliminary experiments. This will provide the robot with life-long learning and hence the capacity to deal with changing structures and/or environments. This is a very valuable feature for self-reconfigurable modular robots. In this framework, it would be interesting to extend the optimization algorithm with the possibility to search a database of previously learned gaits in order to avoid relearning gaits for previously encountered structures or environments. The self-modeling approach proposed by Bongard and colleagues (2006) could here be very useful for monitoring changes and damages in robot structures, and would nicely complete our online learning mechanism.

Note that one problem with online learning is the risk of damaging the robots when testing bad locomotor patterns (e.g. patterns that make the robot fall heavily or that produce and internal collisions). We did not address this problem here because the robot structures that we tested are not capable of lifting themselves up far from the ground. But this is certainly a problem that can occur in the more general case. One solution that we could envision to limit this risk is to run a simulator in parallel to the real robots, and to test locomotor patterns in simulation before deciding to send them to the real robot. The simulator would not be used to quantitatively evaluate the locomotion (as discussed earlier, we believe this should be done on the real robot in its real environment), but only to filter out potentially dangerous ones.

We are also exploring how to carry out optimization autonomously on board of the robots. The optimization algorithm could easily be implemented on a master module (instead of the PC). The master module would estimate its speed based on its own sensors (inertial sensors for instance), and transmit CPG parameters to the other modules at each evaluation step. More interesting, but also more difficult, would be to design an optimization algorithm that is itself distributed among modules, and does not depend on a master module.

Finally, locomotion is not useful if it cannot be modulated by sensory information. We therefore intend to explore how sensory information can be integrated in the CPGs for modulation of speed and direction of locomotion. See Kamimura et al. (2004) for a nice example of entrainment between the M-TRAN modules and a CPG model and of compensation of drift phenomena. In related work, we demonstrated how CPGs can continuously be modulated by sensory information (Righetti and Ijspeert, 2008) for agile locomotion with rapid changes of speed, direction and types of gait (Lachat et al., 2006; Ijspeert, 2001), and also to adapt to the resonant frequency of a compliant robot (Buchli et al., 2006a). The challenge here is to do the same but with a distributed network of sensors, and to learn how to best use and transmit among modules the sensory information available in each module.

Appendix A: Index to Multimedia Extensions

The multimedia extensions to this article can be found online by following the hyperlinks from www.ijrr.org.

Table 5: Index to Multimedia Extensions

Ext.	Media type	Description
1	Video	Snake configuration, random gait
2	Video	Snake configuration, efficient gait
3	Video	Tripod configuration, random gait
4	Video	Tripod configuration, efficient gait
5	Video	Quadruped configuration, random gait
6	Video	Quadruped configuration, efficient gait

Acknowledgment

We gratefully acknowledge the technical support of André Guignard, Andres Upegui, André Badertscher and Philippe Vosseler in the design and the construction of the robot modules. We also acknowledge Elmar Dittrich, Kevin Drapel, Cyril Jaquier, Adamo Maddalena, Daniel Marbach, and Michel Yerly for helping testing the modules and the algorithms. This work was made possible thanks to the financial support from the EPFL and the Swiss National Science Foundation.

References

Aoi, S. and Tsuchiya, K. (2006). Stability analysis of a simple walking model driven by an oscillator with a phase reset using sensory feedback. *IEEE Transactions on Robotics*, 22(2):391–397.

Arena, P., Fortuna, L., Frasca, M., and Sicurella, G. (2004). An adaptive, self-organizing dynamical system for hierarchical control of bio-inspired locomotion. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 34(4):1823–1837.

Beer, R. D. and Gallagher, J. C. (1992). Evolving dynamical neural networks for adaptive behavior. *Adaptive Behavior*, 1(1):91–122.

Bongard, J., Zykov, V., and Lipson, H. (2006). Resilient machines through continuous self-modeling. *Science*, 314(5802):1118–1121.

Brent, R. (1973). *Algorithms for Minimization without Derivatives*. NJ: Prentice-Hall.

Buchli, J., Iida, F., and Ijspeert, A. (2006a). Finding resonance: Adaptive frequency oscillators for dynamic legged locomotion. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2006)*, pages 3903–3909. IEEE.

Buchli, J., Righetti, L., and Ijspeert, A. (2006b). Engineering entrainment and adaptation in limit cycle systems – from biological inspiration to applications in robotics. *Biological Cybernetics*, 95(6):645–664.

Cohen, A. H., Holmes, P. J., and Rand, R. (1982). The nature of coupling between segmented oscillations and the lamprey spinal generator for locomotion: a mathematical model. *J. Math. Biol.*, 13:345–369.

Collins, S., Ruina, A., Tedrake, R., and Wisse, M. (2005). Efficient Bipedal Robots Based on Passive-Dynamic Walkers. *Science*, 307(5712):1082–1085.

Conradt, J. and Varshavskaya, P. (2003). Distributed central pattern generator control for a serpentine robot. In *International Conference on Artificial Neural Networks (ICANN 2003).*

Delcomyn, F. (1980). Neural basis for rhythmic behaviour in animals. *Science*, 210:492–498.

Duff, D., Yim, M., and Roufas, K. (2001). Evolution of polybot: A modular reconfigurable robot. In *In Proc. of the Harmonic Drive Intl. Symposium, Nagano, Japan, Nov. 2001, and Proc. of COE/Super-Mechano-Systems Workshop, Tokyo, Japan, Nov. 2001.*

Endo, G., Nakanishi, J., Morimoto, J., and Cheng, G. (2005). Experimental studies of a neural oscillator for biped locomotion with Q RIO. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation (ICRA2005)*, pages 598–604, Barcelona, Spain.

Geng, T., Porr, B., and Wörgötter, F. (2006). Fast biped walking with a reflexive neuronal controller and real-time online learning. *International Journal of Robotics Research*, 3:243–261.

Grillner, S. (1985). Neural control of vertebrate locomotion – central mechanisms and reflex interaction with special reference to the cat. In Barnes, W. J. P. and Gladden, M. H., editors, *Feedback and motor control in invertebrates and vertebrates*, pages 35–56. Croom Helm.

Gruau, F. and Quatramaran, K. (1997). Cellular encoding for interactive evolutionary robotics. In Husbands, P. and Harvey, I., editors, *Proceedings of the Fourth European Conference on Artificial Life, ECAL97*, pages 368–377. MIT Press.

Ijspeert, A. (2001). A connectionist central pattern generator for the aquatic and terrestrial gaits of a simulated salamander. *Biological Cybernetics*, 84(5):331–348.

Ijspeert, A. and Kodjabachian, J. (1999). Evolution and development of a central pattern generator for the swimming of a lamprey. *Artificial Life*, 5(3):247–269.

Ijspeert, A. J. (2008). Central pattern generators for locomotion control in animals and robots: a review. *Neural Networks*, page In press.

Ijspeert, A. J. and Crespi, A. (2007). Online trajectory generation in an amphibious snake robot using a lamprey-like central pattern generator model. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2007).*

Ijspeert, A. J., Crespi, A., Ryczko, D., and Cabelguen, J. M. (2007). From swimming to walking with a salamander robot driven by a spinal cord model. *Science*, 315(5817):1416 – 1420.

Kamimura, A., Kurokawa, H., Toshida, E., Tomita, K., Murata, S., and Kokaji, S. (2003). Automatic locomotion pattern generation for modular robots. In *IEEE International Conference on Robotics and Automation (ICRA2003).*

Kamimura, A., Kurokawa, H., Yoshida, E., Murata, S., Tomita, K., and Kokaji, S. (2004). Distributed adaptive locomotion by a modular robotic system, M-TRAN II. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2004)*, pages 2370–2377.

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

Kohl, N. and Stone, P. (2004). Policy gradient reinforcement learning for fast quadrupedal locomotion. In *Proceedings of the 2004 IEEE International Conference on Robotics & Automation*, pages 2619–2624.

Kopell, N., Ermentrout, G. B., and Williams, T. L. (1991). On chains of oscillators forced at one end. *SIAM, Journal of Applied Mathematics*, 51(5):1397–1417.

Lachat, D., Crespi, A., and Ijspeert, A. J. (2006). Boxybot: a swimming and crawling fish robot controlled by a central pattern generator. In *Proceedings of The first IEEE/RAS-EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob 2006)*.

Lewis, M. A., Fagg, A. H., and Bekey, G. A. (1993). Genetic algorithms for gait synthesis in a hexapod robot. In Zheng, Y. F., editor, *Recent trends in mobile robots*. World Scientific.

Marbach, D. and Ijspeert, A. J. (2005). Online optimization of modular robot locomotion. In *Proceedings of the IEEE Int. Conference on Mechatronics and Automation (ICMA 2005)*, pages 248–253.

Matsubara, T., Morimoto, J., Nakanishi, J., Sato, M., and Doya, K. (2006). Learning CPG-based biped locomotion with a policy gradient method. *Robotics and Autonomous Systems*, 54:911–920.

Moeckel, R., Jaquier, C., Drapel, K., Dittrich, E., Upegui, A., and Ijspeert, A. J. (2006). Exploring adaptive locomotion with YaMoR, a novel autonomous modular robot with Bluetooth interface. *Industrial Robot*, 33(4):285–290.

Moeckel, R., Sproewitz, A., Maye, J., and Ijspeert, A. J. (2007). An easy to use bluetooth scatternet protocol for fast data exchange in wireless sensor networks and autonomous robots. In *Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2801–2806.

Murata, S., Yoshida, E., Kamimura, A., Kurokawa, H., Tomita, K., and Kokaji, S. (2002). M-tran: Self- reconfigurable modular robotic system. *IEEE/ASME Transactions on Mechatronics*, 7(4):431–441.

Nakamura, Y., Mori, T., Sato, M., and Ishii, S. (2007). Reinforcement learning for a biped robot based on a cpg-actor-critic method. *Neural Networks*, 20(6):723–735.

Nishii, J., Uno, Y., and Suzuki, R. (1994). Mathematical models for the swimming pattern of a lamprey, i. analysis of collective oscillators with time-delayed interaction and multiple coupling. *Biological Cybernetics*, 72:1–9.

Ostergaard, E. H. and Lund, H. H. (2003). Evolving control for modular robotic unit. In *In Proceedings of CIRA'03, IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pages 886–892.

Paul, C. and Bongard, J. C. (2001). The road less travelled: Morphology in the optimization of biped robot locomotion. In *Proceedings of The IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2001)*.

Press, W., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1994). *Numerical recipes in C : the art of scientific computing, 2nd edition*. Cambridge University Press.

Righetti, L. and Ijspeert, A. (2008). Pattern generators with sensory feedback for the control of quadruped locomotion. In *Proceedings of the 2008 IEEE International Conference on Robotics and Automation*. Accepted for publication.

Righetti, L. and Ijspeert, A. J. (2006). Programmable central pattern generators: an application to biped locomotion control. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation (ICRA2006)*, pages 1585–1590.

Salemi, B. and Shen, W. (2004). Distributed behavior collaboration for self-reconfigurable robots. In *Proceedings of the IEEE Conference on Robotics and Automation (ICRA 2004)*.

Shen, W., Salemi, B., and Will, P. (2002). Hormone-inspired adaptive communication and distributed control for self-reconfigurable robots. *IEEE Transactions on Robotics and Automation*, 18(5):1–12.

Shen, W., Will, P., Galstyan, A., and Chuong, C. (2004). Hormone-inspired self-organization and distributed control of robotic swarms. *Autonomous Robots*, 17(4):93–105.

Sigvardt, K. A. and Williams, T. L. (1996). Effects of local oscillator frequency on intersegmental coordination in the lamprey locomotor CPG: theory and experiment. *J. of Neurophysiology*, 76(6):4094–4103.

Sims, K. (1994). Evolving 3d morphology and behavior by competition. In *Proceedings, Artificial Life IV*, pages 28–39. MIT Press.

Stoy, K., Shen, W., and Will, P. (2003). Implementing configuration dependent gaits in a self-reconfigurable robot. In *Proceedings of the IEEE International conference on Robotics and Automation (ICRA2003)*.

Sutton, R. and Barto, A. G. (1998). *Reinforcement learning: an introduction*. MIT Press.

Upegui, A., Moeckel, R., Dittrich, E., Ijspeert, A. J., and Sanchez, E. (2005). An fpga dynamically reconfigurable framework for modular robotics. In Brinkschulte, U., editor, *Workshop Procedings of the 18th International Conference on Architecture of Computing Systems 2005 (ARCS'05)*. VDE Verlag, Berlin.

Vassilvitskii, S., Kubica, J., Rieffel, E., Suh, J., and Yim., M. (2002). On the general reconfiguration problem for expanding cube style modular robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA2002)*.

Vona, M. and Rus, D. (2000). Ta physical implementation of the self-reconfigurable crystalline robot. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA2000)*, pages 1726–1733.

Weingarten, J. D., Lopes, G., Buehler, M., Groff, R. E., and Koditschek, D. E. (2004). Automated gait adaptation for legged robots. In *Proceedings of the 2004 IEEE International Conference on Robotics & Automation*, pages 2153–2158.

Williams, T. L., Sigvardt, K. A., Kopell, N., Ermentrout, G. B., and Rempler, M. P. (1990). Forcing of coupled nonlinear oscillators: studies of intersegmental coordination in the lamprey locomotor central pattern generator. *J. of Neurophysiology*, 64:862–871.

Yerly, M. (2007). Yamor lifelong learning. Master's thesis, EPFL.

Yim, M. (1994). *Locomotion with a Unit Modular Reconfigurable Robot*. PhD thesis, Stanford University Mechanical Engineering Dept.

Zhang, Y., Fromherz, M., Crawford, L., and Shang, Y. (2002). A general constraint-based control framework with examples in modular self- reconfigurable robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2002)*.