Proceedings of the IEEE
International Conference on Mechatronics & Automation
Niagara Falls, Canada • July 2005

Online Optimization of Modular Robot Locomotion

Daniel MARBACH and Auke Jan IJSPEERT
Swiss Federal Institute of Technology at Lausanne,
CH 1015 Lausanne, Switzerland
{daniel.marbach, auke.ijspeert}@epfl.ch
WWW: http://birg.epfl.ch

Abstract – **Adaptive locomotion in unstructured and unpredictable environments is one of the most advertised features of modular robots in the literature. Autonomous modular robots are expected to adapt in the face of a dynamic environment, unexpected tasks and/or module failures. There are two levels of adaptation: Within a static configuration, a chain-type modular robot can adapt its locomotion gait using its many degrees of freedom and the inherent redundancy. In addition, the robot may self-reconfigure to adapt also its morphology. Online optimization of locomotion in a self-organizing manner is mandatory within this context.**

The contribution of this paper is three-fold: i) Inspired by Central Pattern Generators (CPGs) found in vertebrates, we propose a distributed locomotion controller based on coupled nonlinear oscillators; ii) For offline optimization, a genetic algorithm that co-evolves the CPG with the configuration of the modular robot is presented; iii) The ultimate goal of our research being autonomous locomotion, the focus of the paper lies on a novel, fast online adaptation method for coupled nonlinear oscillators. The algorithm allows fast online optimization (adaptation) of locomotion gaits in the face of module failures or new, previously unknown configurations. A realistic simulation of our hardware prototype YaMoR is used for the experiments.

I. Introduction

Modular robots are generally classified as being lattice-type or chain-type. Lattice modular robotic systems use cluster-flow locomotion: In order to move, the robot continuously reconfigures, thereby giving the impression that the module cluster "flows" on the ground and around obstacles. Our research concerns chain-type robots [1-3] that locomote in a static configuration, using powered joints for instance. Reconfiguration occurs 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.

Locomotion is an essential skill of any autonomous robot. The promise of adaptive locomotion in harsh and unpredictable environments has been a major motivation for research in modular robotics and related applications such as urban search and rescue, planetary exploration or undersea mining are mentioned by the vast majority of publications in the field. In these scenarios, modular robots are expected to self-reconfigure in order to autonomously adapt their morphology to a dynamic environment, new tasks and module failures. Adaptation is also possible within a static configuration by taking advantage of the many degrees of freedom (DOF) and the inherent redundancy.

In vertebrates, Central Pattern Generators (CPGs, i.e. neural networks that produce complex oscillatory output from simple tonic input) located in the spinal cord generate the rhythmic signals for locomotion [4]. Various computational CPG models have been used for adaptive biped [5], quadruped [6], swimming [7] and amphibious [8] robot locomotion. These studies have shown that CPGs can be designed as *distributed* systems of coupled neural or nonlinear oscillators, and produce very robust locomotion with speed, direction, and even types of gaits that can quickly be modified depending on the environmental conditions.

Previous research in chain-type modular robot locomotion control [3][9-11] has not taken inspiration from the concept of CPGs; except for Kamimura et al., who use two-neuron Matsuoka oscillators as a CPG model for M-TRAN in their recent work [1,12]. A genetic algorithm (GA) is applied to optimize the free CPG parameters offline for specific configurations. In [1] 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.

The research cited above [1][5-8] incorporates sensory feedback to design adaptive CPGs. Input from the sensors affects the state and shapes the oscillatory output of the system in the near future. However, there is no long-term memory or learning effect and the gait must previously be optimized offline by a GA, for instance. In contrast, we investigate online adaptation, and this on a higher level, acting on the parameters of the CPG (e.g. the coupling weights). The two approaches are complementary and should ideally be combined with each other. We see two major applications for the online optimization (we could also say adaptation or learning) algorithm: First of all, it is possible to quickly optimize the locomotion gait for a new, previously unknown configuration. This may be the case after self-reconfiguration or self-assembly. Secondly, the locomotion gait may be adapted to module failure(s). The algorithm relies on Powell's method [13-14] for multidimensional function optimization. To the best of our knowledge, there has been no previous research in online optimization of chain-type modular robot locomotion.

The paper is organized as follows. The next section reviews our hardware prototype YaMoR and the simulation environment. Subsequently the Central Pattern Generator (CPG) model based on coupled nonlinear oscillators is introduced. Co-evolution of the CPG with the configuration of the modular robot is discussed in Section 4. Finally we explain the online optimization algorithm and discuss the results.

0-7803-9044-X/05/$20.00 © 2005 IEEE

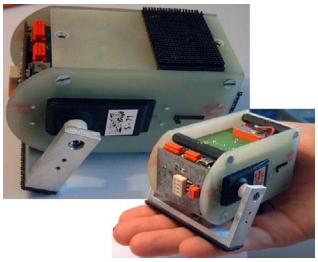

Figure 1: The YaMoR module.

II. YAMOR

YaMoR [15,21] (Figure 1) is the modular robotic system that is currently being developed at the Biologically Inspired Robotics Group (BIRG) of the Swiss Federal Institute of Technology at Lausanne (EPFL). We focus on issues in chain-type locomotion and leave self-reconfiguration aside for now. A special type of strong genderless Velcro® fastener is used to attach the modules together. Velcro proved to be a very simple, flexible and efficient way to manually connect modules.

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 an RC-servo with maximum rotation speed of 60°/0.16s and a maximum torque of 0.73 Nm, which is sufficient for a module to lift three others.

The modules are self-contained, i.e. autonomous with respect to power, actuators and computation. Each module is equipped with a Field Programmable Gate Array (FPGA) with 400'000 gates and 4 Mbit of high speed SRAM. Sensors are not included in this first prototype. The module is powered by two rechargeable Li-Ion batteries. The most distinctive feature of YaMoR is the wireless communication via BlueTooth. Advantages are low power consumption, small size and the absence of an electrical connection between the modules. Furthermore, disjoint modules or groups of modules can communicate [15,21].

YaMoR has successfully demonstrated locomotion in various configurations. Movies and further technical details are available on the BIRG web page [16].

The simulation environment is implemented with the Open Dynamics Engine (ODE) [17]. Powered hinge joints are used to simulate the RC-Servo of YaMoR. A PD controller as described in [18] sets the torques that are necessary to follow the trajectories generated by the CPG.

III. CPG CONTROLLER

A. Coupled Nonlinear Oscillators

We choose the following coupled nonlinear oscillator as canonical sub system of the CPG:

$$\begin{cases} \tau \dot{x}_i = v_i \\ \tau \dot{v}_i = -\alpha \dfrac{x_i^2 + v_i^2 - E_i}{E_i} v_i - x_i + p_i \end{cases} \qquad (2)$$

where α, τ and E_i are positive constants. The variable p_i is the perturbation of the oscillator due to interoscillator coupling (see below). The state variable x_i provides the oscillatory signal for the YaMoR unit. A similar CPG was used in ref. [8] to model salamander locomotion. The limit cycle behavior of an unperturbed oscillator (p_i=0) is a sinusoidal signal:

$$\tilde{x}_i(t) = \sqrt{E_i}\,\sin(t/\tau + \phi_i) \qquad (4)$$

where the phase ϕ_i depends on the initial conditions. In order to drive the system to specific phase differences, the oscillators (Eq. 2) of the CPG are coupled by projecting signals to each other proportional to their x and v states:

$$p_i = \sum_j w_{ij} \left(\frac{\sin(\phi_{ij}) x_j + \cos(\phi_{ij}) v_j}{\sqrt{x_j^2 + v_j^2}} - \frac{v_i}{\sqrt{x_i^2 + v_i^2}} \right) \qquad (3)$$

The positive constant w_{ij} determines the strength of the coupling from oscillator j to oscillator i and the parameter ϕ_{ij} sets the phase difference between the two oscillators. Indeed, x_i converges to a sine (Eq. 4) and the phase difference with oscillator j stabilizes at ϕ_{ij} if the network is acyclic. This holds for arbitrary initial states except for the singularity where all x_i and v_i are zero. See [19] for additional discussion.

B. Connection scheme

We follow the paradigm of strictly local interaction and use nearest neighbor couplings from parent to child oscillators (see Figure 2). In our case, the parent-child relation stems from the tree genome (see next section). Generally, one could use a distributed algorithm to construct a spanning tree over all modules [3]. We use unidirectional and not bidirectional couplings because of faster convergence to the limit cycle in large networks. Parameters are summarized in Table I.

Besides from the modules that are controlled by an oscillator, we also use rigid modules. The hinge joint of YaMoR modules cannot be locked mechanically, thus the trajectory of the module is set to the desired (constant) angle. Having no active oscillator, a rigid module simply relays the couplings from its children to the parent. Refer to Figure 2.

A module with an active oscillator periodically communicates its state (x_i, v_i) to the children (every 0.01s). With the state information received from the parent, the module computes the coupling term (Eq. 3) and integrates the differential equation of its oscillator (Eq. 2) with embedded Runge-Kutta-Fehlberg 4,5 [20] method using adaptive step-size control.

Rigid modules simply forward the state information from their parent to all children. Note that the modules do not need

TABLE I. PARAMETERS OF THE CPG

Parameter	Value/Range	Description
τ	$1/\pi$	Determines the oscillation period (typically 2 sec)
α	1	Controls the ‚attracting force' of the limit cycle.
w_{ij}	1.5	Coupling strength (parent-child couplings only)
E_i	$(0, \pi/4]$	Free parameter, desired amplitude A_i = sqrt(E_i).
ϕ_{ij}	$[0, 2\pi)$	Free parameter, desired phase difference

to be reset to get synchronized; the CPG smoothly converges to constant phase differences through strictly local interaction.

IV. Co-Evolution Of Configuration And Control

In [18] we presented a GA for the co-evolution of configuration and control of chain-type modular robots. Fictive modules, controlled by harmonic oscillators were used. Now we take the same approach to co-evolve YaMoR configurations with CPGs. As a novelty, we apply a genotype-to-phenotype mapping for symmetric structures. The evolved robots are used as a benchmark and testbed for the online optimization (see Section 6). The GA is not discussed in detail; the interested reader may refer to [19] for a comprehensive review.

The genotype is a tree, where nodes represent modules and links physical connections between modules. Nodes encapsulate the free parameters of a module. We distinguish structural parameters and control parameters. The former ones define the structure of the modular robot (e.g. docking positions and initial joint angles); the latter ones are the free parameters of the CPG, namely the energy E_i and the phase difference with the parent oscillator ϕ_{ij} (see Table I). In addition, each node has a Boolean parameter defining if the module is rigid or oscillating. We use the same standard mutation, crossover, selection and replacement operators as in [18].

Inspired by symmetry in nature and the tendency of the GA to evolve quasi-symmetric individuals, we designed a genotype-to-phenotype mapping that mirrors the configuration along the spine (the robots axis of symmetry). Refer to Figure 2. The genotype is the same as discussed above, the only difference being that each node must not only encode its own phase ϕ_{ij}, but also the phase of its mirrored counterpart. As in nature, the amplitudes of corresponding joints in left and right limbs are the same. Thus, only an additional phase parameter must be added to the nodes. Refer to [19].

V. Online Optimization

GAs are by far the most popular offline optimization method for multiple degree of freedom locomotion [1][5-8].

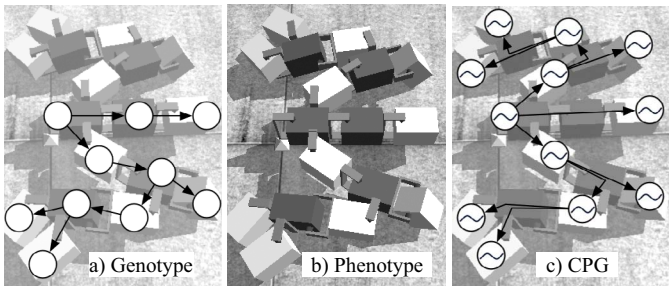

Figure 2: Genotype, phenotype and CPG of an evolved YaMoR robot. a) The genotype is a tree, where nodes represent modules and links physical connections. b) The symmetric genotype-to-phenotype mapping 'mirrors' the limbs along the spine, resulting in a symmetric structure. Rigid modules are in black, oscillating modules in white. c) Unidirectional nearest neighbor couplings are used. Rigid modules relay the coupling from their parent to the children. The root of the tree must always have an active oscillator (even if it is set rigid) to ensure that the network is a spanning tree over all oscillators.

GAs and other stochastic optimization algorithms (e.g. simulated annealing [14]) avoid small local optima to a certain point by exploring large areas of the search space. The payoff is a very slow convergence, which is the main reason why they are not well suited for online optimization. Here we take a radically different approach. By applying a 'classical' function optimization algorithm (Powell's method) to optimize the fitness function *f* we get fast convergence, the payoff being the risk to converge to local optima. The next two sections are a brief introduction to Brent's and Powell's algorithms. Subsequently we discuss how we apply them to robot locomotion.

A. One-Dimensional Function 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 $a < b < c$ such that $f(b)$ is less than both $f(a)$ and $f(c)$. In this case and if f is non-singular, 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 [14].

The challenge is finding the best strategy for choosing the new point x in the bracketing interval at each iteration. We use Brent's method, which is a clever combination of golden section search and parabolic interpolation [13,14,19].

B. Powell's method in multidimensions

Consider a line defined by a starting point **P** and a direction **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 [14] (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 [13,14] is best explained with an example. Consider a function with a 'valley' along $x=y$ that descends to the origin:

$$f(x,y) = \sqrt{x^2+y^2} + (x-y)^2 \qquad (5)$$

Powell's method starts with the unit vectors $\mathbf{e}_1$, $\mathbf{e}_2$, …$\mathbf{e}_N$ 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 3 for the two-dimensional function introduced above (Eq. 5). Starting at the initial point **p0**=(2,5), the first line minimization along the direction given by the unit vector $[1,0]^T$ takes us to the point **p1**. From this point the second line minimization along $[0,1]^T$ takes us to **p2** and completes the first iteration. As

you can see on Figure 3, 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 $\mathbf{v}_i = \mathbf{p}_0 - \mathbf{p}_N$ where $\mathbf{p}_0$ was the starting point at the current iteration and $\mathbf{p}_N$ the new point after the N line minimizations. In the example of Figure 3, $\mathbf{v_2}$ 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 $\mathbf{v}_i$ after each iteration and which direction in the set should be replaced are described in [13,14]. Note that there is no learning rate; the algorithm simply always goes to the optimum in the next direction.

C. Optimization of Locomotion

We shall now explain how Powell's method can be used to optimize the fitness function $f(\mathbf{x})$. The vector $\mathbf{x}$ contains all free parameters of the CPG (remember Table I). At each step of the algorithm, Powell's method gives the next parameter vector $\mathbf{x}_i$ that should be evaluated. The CPG parameters are then reset to $\mathbf{x}_i$ and the gait is evaluated during approximately 10 seconds as described below. Afterward the fitness $f(\mathbf{x}_i)$ is returned to Powell's method, which will give the following parameter vector to be tested.

We define the fitness as the average speed of the modular robot. The aim is to estimate the fitness $f(\mathbf{x})$ in as little time as possible. It is assumed that the robot has the ability to estimate its speed, even though YaMoR is not yet equipped with the necessary sensors. In order to promote locomotion in a straight line, the average speed is not computed from the total distance traveled, but from the distance between the starting and the end point after a certain amount of time. The speed of the robot is defined as the speed of a single, specific module (the root of the configuration tree, which lies in the axis of symmetry). Optimizing this module's speed ultimately leads to optimization of the robot's speed as a whole.

The average speed is always estimated over one period (2 seconds). The fitness evaluation of a gait is only done when the difference between the three last estimated speeds is smaller than 0.02m/s. If this is the case, the fitness of the gait is defined as the average of these three estimates. Thus, the fitness is the average speed over three periods (6 seconds), but only if the speed was stable during that time.

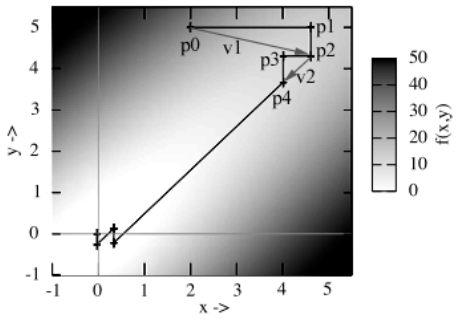

Figure 3: Powell optimization of the two-dimensional function f (Eq. 5). Succeeding line minimizations starting at point p0 are plotted in black. The vectors v1 and v2 correspond to the new directions tested after the first and second iteration respectively. Further explanation in main text.

Depending on the configuration and the gait, one fitness evaluation takes roughly 10 seconds (5 periods). Experience showed that fitness evaluation is quite noisy and that it is not beneficial to use a precision of more than 0.1 radians for Brent's method. Using this precision, a line minimization over an interval of 2π (largest possible bracketing interval for a phase difference) involves less than 10 fitness evaluations (generally about 5). 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 generally between 10 and 30 for the configurations that we tested).

For fast optimization, the number of free parameters must be reduced to a minimum. The CPG model introduced in Section 3 fits well within this context because there are only two parameters per oscillating module: The amplitude and the phase difference with the parent. Symmetric configurations allow further reduction of the search space by using equal parameter values for corresponding modules in symmetric limbs. In other words, symmetric limbs share the same control parameters; only the phase difference to the spine (i.e. the phase of the first module of the limb) must be independent.

As the algorithm converges towards the optimum, line minimizations are often unable to improve the current optimum, i.e. the starting point is already optimal on the considered line. In this case it is important to re-evaluate the current optimum because the environment and consequently also the fitness landscape might be dynamic. By taking the average over the last k re-evaluations of the same optimum (we use k=4) resistance to noise in the fitness estimation is improved.

A master module, for example the root of the configuration tree, 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 upon completion of fitness evaluation, i.e. roughly every 10 seconds).

VI. Results

Typical CPG trajectories are illustrated on Figure 4. The system smoothly synchronizes from *random initial states* (x_0, v_0) to a stable gait. Minor gait changes (e.g. modification of one parameter value), which are common during online optimization, are very fast and smooth. Even major gait transitions where many parameters are brutally reset to new values take less than two periods (4 seconds) and are smooth. This smoothness is a key feature of nonlinear oscillators, and allows us to avoid brutal resets that might damage the motors.

We test online optimization in two scenarios with the body configurations that have previously been evolved with the co-evolutionary algorithm: i) Optimization from scratch. All CPG parameters of the evolved robot are reset to random values. Online optimization starts from this random gait. ii) Adapta-

tion to module failures. The evolved CPG parameters are not reset, but one of the oscillating modules is set rigid to simulate a module failure. In this case the robot may not be considered symmetric anymore, hence control parameters of corresponding limbs are not shared any longer. Consequently, the number of free parameters augments, which improves adaptability of the system to the module failure.

We choose the configuration introduced in Figure 2 for discussion because with eleven DOF and a very sophisticated locomotion gait it was one of the most complex robots evolved by the GA. The results are summarized in Table 2.

Using random initial parameter values, the fitness (speed) of the robot is improved from 0.14m/s to 0.38m/s in only 44 minutes in the average, thus outperforming the gait that had been evolved by the GA (0.37m/s) in dozens of hours. (Note that we always refer to simulated time, which is real time and not computation time). The current optimum after each iteration of Powell's method is plotted in Figure 5 for ten runs. Qualitatively, the eight runs that achieved a speed of more than 0.35m/s converged all to the same type of gait as the GA. The remaining two runs converged to a less efficient gait (i.e. a local optimum).

Figure 6 illustrates how the parameter space is explored and how fitness typically evolves during the optimization process. The 'spikes' correspond to the line minimizations. The first iteration of Powell's method is completed after 12 minutes and the line minimization along the new direction can be observed by the simultaneous change of several parameters at that time.

The second and third line of Table 2 concern adaptation to module failures. As mentioned above, there are more free parameters in this case (22) than in optimization from scratch

TABLE II. PERFORMANCE OF THE ONLINE OPTIMIZATION

	Initial speed (m/s)	Speed after Powell (m/s)	Time in minutes	# fitness evaluations	# powell iterations
Random	0.17 (0.08)	0.40 (0.07)	49.23 (19.13)	273.1 (100.55)	3.40 (1.26)
Evolved gait	0.37 (0.03)	0.44 (0.03)	59.84 (20.68)	344.67 (124.28)	4 (1.12)
Evolved gait w/ failure	0.22 (0.06)	0.39 (0.08)	72.03 (40.39)	368.71 (219.82)	4.71 (2.06)

The 1st line corresponds to online optimization from randomly initialized CPGs. The 2nd and 3rd line start both with the gait from the GA , the latter one with failure of the module that connects the left limb to the spine. Each cell contains the average (top) and the standard deviation (below in parenthesis) from 10 runs.

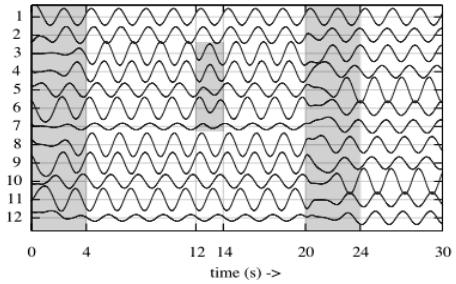

Figure 4: CPG Trajectories of the robot from Figures 2,7. Three kinds of gait transitions are illustrated: i) Initial synchronization in the first 4s. ii) Minor gait change after 12s. iii) Major gait change after 20s.

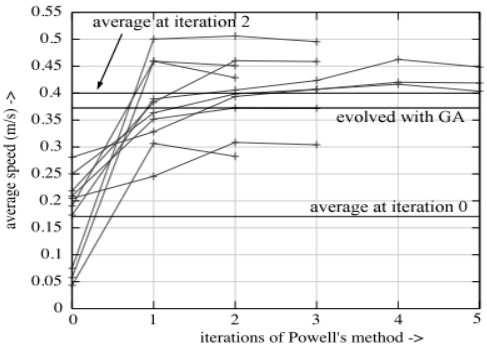

Figure 5: Online optimization from scratch. The current optimum after each iteration is plotted for 10 independent runs. The horizontal lines indicate the average after random initialization, after two iterations and the speed of the gait that had been evolved with the GA. Fitness (average speed) may drop in the last iteration due to re-evaluation and estimation error.

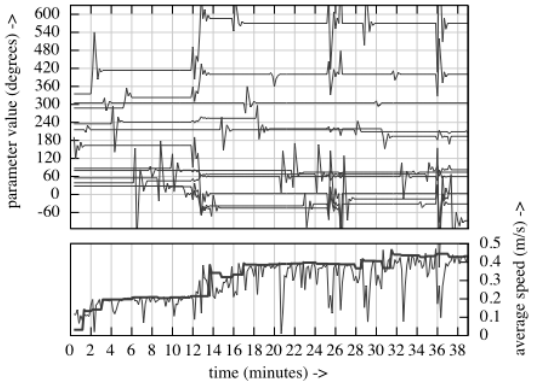

Figure 6: Evolution of CPG parameter values (top) and maximum and current fitness (bottom) during Powell optimization. Maximum fitness (bold) drops occasionally because of re-evaluation and estimation error. Current fitness fluctuates as optimization explores the parameter space.

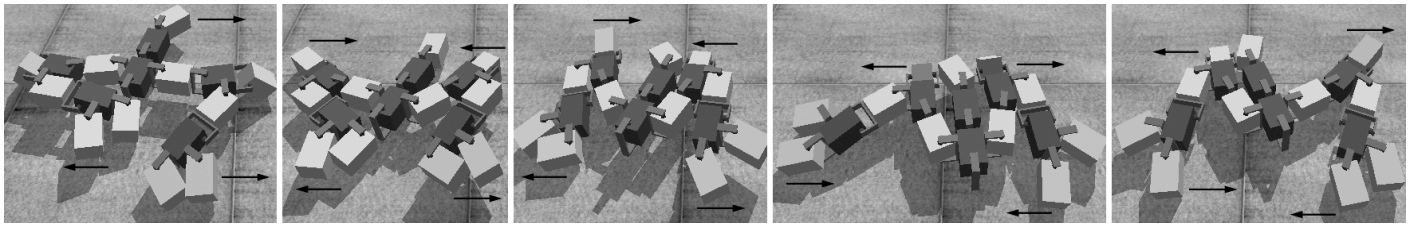

Figure 7: Locomotion gait of the robot from Figure 2 after online optimization from scratch. Rigid modules are in black, oscillating modules in white. The robot is moving from left to right. Arrows pointing to the right indicate that the corresponding limb is in the air and moving in the direction of locomotion. Left arrows mean that the corresponding limb is on the ground, pushing to the left. Movies are available at: *http://birg.epfl.ch/page56514.html*

(13) because the robot is not considered symmetric anymore. As starting point we take the evolved gait, which has an average speed of 0.37m/s. One hour of online optimization improved the performance to 0.44m/s in the average. Deactivating the first module of the left arm caused fitness to drop to 0.22m/s. Subsequently the algorithm successfully adapted the gait to the module failure, achieving still superior fitness (0.38m/s) than the GA without module failure.

We have tested online optimization with various configurations from the GA (e.g. a salamander-like quadruped, a simple two-armed crawler). Outcomes concur with the results presented above: i) Optimization from scratch performs well in the average even though some runs converge to local optima. ii) After a module failure, the gait is successfully adapted to the new constraints. Provided that the configuration is redundant, adaptation almost restores original fitness. iii) Optimization is extremely fast. Locomotion gaits close to the optimum are generally found already after the first iteration of Powell's method, i.e. after about 10 minutes. Movies are available at: *http://birg.epfl.ch/page56514.html*

VII. Conclusions – Future Work

Using coupled nonlinear oscillators as canonical subsystems of the CPG, we have designed a distributed modular robot controller that is characterized by few parameters as well as fast and smooth gait transitions. These features are essential for successful online optimization.

Employing symmetric structures and control architectures improved the performance of both offline and online optimization. This is explained by a limited and better structured search space and the fact that symmetric robots are better suited for locomotion in a straight line than asymmetric ones.

This paper is a proof of concept that it is possible to apply Powell's method (and potentially other direction-set methods for multidimensional function optimization) to optimize multiple DOF robot locomotion online. We have demonstrated that using Powell's method, a modular robot can learn efficient locomotion in a new, previously unknown configuration which might be the result of a self-assembly or self-reconfiguration procedure. Exploiting the same approach, we have successfully tested adaptation to module failures.

As with any other optimization algorithm, there is a certain risk to converge to a local optimum. This risk is higher for Powell's method than for stochastic optimization algorithms (e.g. GAs) that explore large areas of the search space. In practice we found that Powell's method finds the same gait as the GA in the majority of cases. It seems that efficient locomotion gaits correspond to strong attractors in the fitness landscape (figuratively speaking high mountains with a wide base). We suspect that the noise in fitness estimation might actually be beneficial to avoid small local optima because a line minimization may step away from them due to estimation error. A similar concept is used in simulated annealing to avoid local optima [14] (the system can always jump to a higher energy state, i.e. lower fitness).

Our next goals are: including sensory feedback in the nonlinear oscillators, evolving higher-level controllers for navigation and reproducing the experiments that we have conducted in simulation with the YaMoR hardware.

Acknowledgments

We would like to acknowledge support from a Swiss National Science Foundation Young Professorship grant to Auke Ijspeert.

References

[1] A. Kamimura, H. Kurokawa, E. Yoshida, S. Murata, K. Tomita, S. Kokaji. Distributed Adaptive Locomotion by a Modular Robotic System, M-TRAN II (From Local Adaptation to Global Coordinated Motion using CPG Controllers). *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2004)*, 2004, pp. 2370-2377.

[2] M. Yim, K. Roufas, D. Duff, Y. Zhang, C. Eldershaw, S. Homans, Modular Reconfigurable Robots in Space Applications. *Autonomous Robots*, vol. 14, issue 2-3, Mar 2003, pp.225 – 237.

[3] W. Shen, B. Salemi, P. Will. Hormone-inspired adaptive communication and distributed control for CONRO self-reconfigurable robots. *IEEE Transactions on Robotics and Automation*, 18(5):700-712, 2002.

[4] A. Ijspeert. Vertebrate locomotion. In M.A. Arbib, editor, *The handbook of brain theory and neural networks*, pp.649-654. MIT Press, 2003.

[5] G. Taga, Y. Yamaguchi, H. Shimizu. Selforganized control of bipedal locomotion by neural oscillators in unpredictable environment. *Biological Cybernetics*, 65:147–159, 1991.

[6] Y. Fukuoka, H. Kimura, A. Cohen. Adaptive dynamic walking of a quadruped robot on irregular terrain based on biological concepts. *The International Journal of Robotics Research*, 3–4:187–202, 2003.

[7] A. Ijspeert, J. Hallam, D. Willshaw. Evolving swimming controllers for a simulated lamprey with inspiration from neurobiology. *Adaptive Behavior* 7:2, pp 151-172, 1999.

[8] A. Ijspeert. A connectionist central pattern generator for the aquatic and terrestrial gaits of a simulated salamander. *Biological Cybernetics*, 85(5):331–348, 2001.

[9] M. Yim, *Locomotion with a Unit Modular Reconfigurable Robot*, Stanford University Mechanical Engineering Dept. thesis, 1994.

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

[11] Y. Zhang, M. Fromherz, L. Crawford, Y. Shang. A General Constraint-Based Control Framework with Examples in Modular Self-Reconfigurable Robots. *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, Lausanne, Switzerland, Oct. 2002.

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

[13] R. Brent. *Algorithms for Minimization without Derivatives*. Englewood Cliffs, NJ: Prentice-Hall, 1973.

[14] W. Press, S. Teukolsky, W. Vetterling, B. Flannery. *Numerical Recipies in C: the art of scientific computing* (2nd ed.), Cambridge University Press, 1992. Online: http://www.library.cornell.edu/nr/bookcpdf.html

[15] A. Upegui, R. Moeckel, E. Dittrich, A. Ijspeert, E. Sanchez. An FPGA Dynamically Reconfigurable Framework for Modular Robotics. in U. Brinkschulte, editor, *Workshop Proceedings of the 18th International Conference on Architecture of Computing Systems 2005 (ARCS'05).* VDE Verlag, Berlin, 2005.

[16] Biologically Inspired Robotics Group (BIRG): http://birg.epfl.ch/

[17] Russell Smith. Open Dynamics Engine (ODE): http://q12.org/ode/

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

[19] D. Marbach. *Evolution and Online Optimization of Central Pattern Generators for Modular Robot Locomotion*. Unpublished Master Thesis, Swiss Federal Institute of Technology Lausanne, 2004. http://birg.epfl.ch/page56514.html

[20] GNU Scientific Library (GSL): http://www.gnu.org/software/gsl

[21] Moeckel R., Jaquier C., Drapel K., Dittrich E., Upegui A., and Ijspeert A.J. YaMoR and Bluemove – an autonomous modular robot with Bluetooth interface for exploring adaptive locomotion. In *Proc. of the Eighth Intl. Conf. on Climbing and Walking Robots (CLAWAR 2005)*, to appear.