Czech Technical University in Prague
Faculty of Electrical Engineering

DIPLOMA THESIS

Milan Prouza

Locomotion Generation for Modular Robots

Department of Cybernetics

Supervisor: **Ing. Vojtěch Vonásek**

Prague, 2015

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

DIPLOMA THESIS ASSIGNMENT

Student: Bc. Milan P r o u z a

Study programme: Cybernetics and Robotics

Specialisation: Robotics

Title of Diploma Thesis: Locomotion Generation for Modular Robots

Guidelines:

1. Study HW platforms of modular robots (e.g. Scout [1] or CosMO [6]) and create a simplified simulation model of one of these (e.g. using ODE physical engine).
2. Study Central Pattern Generation (CPG) approach for locomotion generation [2,3], and implement at least two different CPGs suitable for modular robots.
3. Implement at least two evolutionary-based optimization methods for finding parameters of selected CPGs to achieve various gaits (e.g. 'move-forward', 'rotate-left', etc.). Suitable methods can be found in [4,5]. Compare the efficiency of the optimization methods to find gaits for robots of various shapes (e.g. snake, lizard-like, quadropod, etc.)
4. Investigate how to decrease the number of fitness function evaluations to speed-up the optimization process (e.g. [7]).
5. Investigate the influence of failures of actuators to the efficiency of locomotion. Try to design such gaits, that are robust enough and that can provide a locomotion even with failed modules. Perform this study on all CPGs from the task 2.
6. Verify the optimization of locomotion generation with real robots (optional, depends on availability of HW).

Bibliography/Sources:

[1] P. Levi, E. Meister AC. van Rossum, T. Krajnik, V. Vonasek, P. Stepan, W. Liu, F. Caparrelli, "A cognitive architecture for modular and self-reconfigurable robots," Systems Conference (SysCon), 2014 8th Annual IEEE , vol., no., pp.465,472, March 31 2014-April 3, 2014.
[2] Q. Wu et al. "Survey of locomotion control of legged robots inspired by biological concept." Science in China Series F: Information Sciences 52.10 (2009): 1715-1729.
[3] A. J. Ijspeert, "Central pattern generators for locomotion control in animals and robots: a review." Neural Networks 21.4 (2008): 642-653.
[4] R. Eberhart, J. Kennedy. "A new optimizer using particle swarm theory." Proceedings of the Sixth International Symposium on Micro Machine and Human Science', pp.39,43, 4-6 Oct 1995, doi: 10.1109/MHS.1995.494215
[5] R. Eberhart, Y. Shi, J. Kennedy, „Swarm Intelligence“, Morgan Kaufmann, 2001.
[6] Liedke, J., „The Collective Self-reconfigurable Modular Organism (CoSMO)“, IEEE/ASME International Conference on Advanced Intelligent Mechatronics, 2013
[7] Z. Cui , J. Zeng, G. Sun, „A fast particle swarm optimization, International Journal of Innovative Computing, Information and Control“, 1365-1380, vol. 2, number 6, 2006.

Diploma Thesis Supervisor: Ing. Vojtěch Vonásek

Valid until: the end of the summer semester of academic year 2015/2016

L.S.

doc. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 20, 2015

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Bc. Milan P r o u z a

Studijní program: Kybernetika a robotika (magisterský)

Obor: Robotika

Název tématu: Generování pohybu pro modulární roboty

Pokyny pro vypracování:

1. Seznamte se s platformou modulárních robotu Scout [1] a Cosmo [6]. Vytvořte simulační model jedné z těchto platforem s použitím knihovny ODE.
2. Seznamte se generátory pohybu založenými na principu Central Pattern Generation (CPG) [2,3], implementujte nejméně dva různé CPG pro modulární roboty.
3. Naimplementujte alespoň dvě evoluční optimalizační metody pro nalezení parametrů CPG tak, aby bylo dosaženo požadovaného pohybu (např. dopředná chůze, otáčení). Lze použít metody uvedené např. v [4,5]. Porovnejte tyto optimalizační metody pro hledání chůze na robotech různých tvarů (např. had nebo ještěrka).
4. Navrhněte metodu pro snížení počtu volání vyhodnocovací funkce tak, aby byla zrychlena optimalizace [7].
5. Studujte vliv poruch jednotlivých modulů na chůzi celého robotu. Navrhněte takové typy chůze, které umožňují realizovat požadovaný pohyb i s uvažováním rozbitých modulů. Tuto studii proveďte na všech CPG naimplementovaných v bodě 2.
6. Ověřte schopnost navržených metod generovat chůzi pro reálné roboty (volitelně, záleží na dostupnosti HW modulů).

Seznam odborné literatury:

[1] P. Levi, E. Meister AC. van Rossum, T. Krajnik, V. Vonasek, P. Stepan, W. Liu, F. Caparrelli, "A cognitive architecture for modular and self-reconfigurable robots," Systems Conference (SysCon), 2014 8th Annual IEEE , vol., no., pp.465,472, March 31 2014-April 3, 2014.
[2] Q. Wu et al. "Survey of locomotion control of legged robots inspired by biological concept." Science in China Series F: Information Sciences 52.10 (2009): 1715-1729.
[3] A. J. Ijspeert, "Central pattern generators for locomotion control in animals and robots: a review." Neural Networks 21.4 (2008): 642-653.
[4] R. Eberhart, J. Kennedy. "A new optimizer using particle swarm theory." Proceedings of the Sixth International Symposium on Micro Machine and Human Science', pp.39,43, 4-6 Oct 1995, doi: 10.1109/MHS.1995.494215
[5] R. Eberhart, Y. Shi, J. Kennedy, „Swarm Intelligence“, Morgan Kaufmann, 2001.
[6] Liedke, J., „The Collective Self-reconfigurable Modular Organism (CoSMO)“, IEEE/ASME International Conference on Advanced Intelligent Mechatronics, 2013
[7] Z. Cui , J. Zeng, G. Sun, „A fast particle swarm optimization, International Journal of Innovative Computing, Information and Control“, 1365-1380, vol. 2, number 6, 2006.

Vedoucí diplomové práce: Ing. Vojtěch Vonásek

Platnost zadání: do konce letního semestru 2015/2016

L.S.

doc. Dr. Ing. Jan Kybic
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 20. 1. 2015

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne ....

Podpis autora práce

Abstract

In this diploma thesis, methods for automatic locomotion generation for modular robots are investigated. Methods based on Central Pattern Generation (CPG) are used to generate control signals for actuators of modules. Optimization of CPGs parameters is processed by two bio-inspired methods, Particle Swarm Optimization (PSO) and Genetic Algorithm (GA). To decrease the number of expensive fitness evaluations, the Fitness Estimation method was applied to both PSO and GA. The performance of optimization methods under different circumstances was statistically compared. An important task of modular robotics is coping with module failures. A novel preventive method for decreasing the impact of failures is proposed in this work and verified together with standard locomotion generators in a scenario with failures. The experiments were conducted using a dedicated simulation environment and selected motion patterns were executed on real robots.

Keywords: robot, modular, locomotion, failure

Abstrakt

V diplomové práci jsou popsány metody pro automatické generování pohybů pro modulární roboty. Řídicí signály pro aktuátory modulů jsou získávány metodami založenými na principu Central Pattern Generation (CPG). Optimalizaci parametrů CPG zajišťují biologicky inspirované metody Particle Swarm Optimization (PSO) a Genetický Algoritmus (GA). Za účelem snížení počtu časově nákladných vyhodnocení fitness funkce byly uvedené algoritmy rozšířeny o metodu Fitness Estimation. Výkony obou optimalizačních algoritmů za různých okolností byly statisticky porovnány. Důležitou úlohou modulární robotiky je vypořádání se se závadami modulů. V této práci je navržena nová preventivní metoda zmírňující dopad poruch na pohyb robotu a je porovnána se standardními generátory pohybu v situacích s poruchami. Experimenty byly prováděny v simulátoru vytvořeném pro potřeby této práce a vybrané pohybové vzory byly spuštěny na reálných robotech.

Klíčová slova: robot, modulární, pohyb, porucha

Contents

Chapter 1

Introduction

In modular robotics the robots are composed of robotic devices called modules. This architecture allows construction of different robotic organisms of various shapes from a finite set of either homogenous or heterogeneous modules. This variability makes modular robotics a very flexible and versatile technology. Furthermore, the modules can be capable of autonomic self-reconfiguration, thereby the organism can adapt its structure in reaction to actual circumstances, or replace a broken module with a spare one, which is extremely useful in case a human operator can't intervene, as in space or in dangerous areas.

A fundamental property of autonomous robot is a capability of a locomotion. In this diploma thesis, a locomotion through joint-control approach is investigated. This type of locomotion uses a collaboration of joints of all modules to generate a movement of the whole organism. To maintain control signals for module actuators, we used the Central Pattern Generation (CPG) method. The control signals are generated by a set of oscillators and the resulting movement is determined by a setting of parameters of these oscillators (e.g. a frequency and phase shift of a harmonic oscillator). Two types of CPG were investigated in this diploma thesis. To find a feasible setting of parameters that emerges into a desired locomotion, Particle Swarm Optimization (PSO) and Genetic Algorithm (GA) were used.

The most time-consuming part of the parameter optimization is computation of the fitness value. If the optimization runs on a real robot, it has to apply the tested parameters, realize the locomotion and measure its quality, thereby every fitness evaluation lasts several seconds. For the purposes of this diploma thesis, a simple simulator of CoSMO modular robots was created to generate and test locomotion patterns. Although the simulator provides much faster fitness evaluation, it still remains the slowest step of the optimization, therefore decreasing the number of

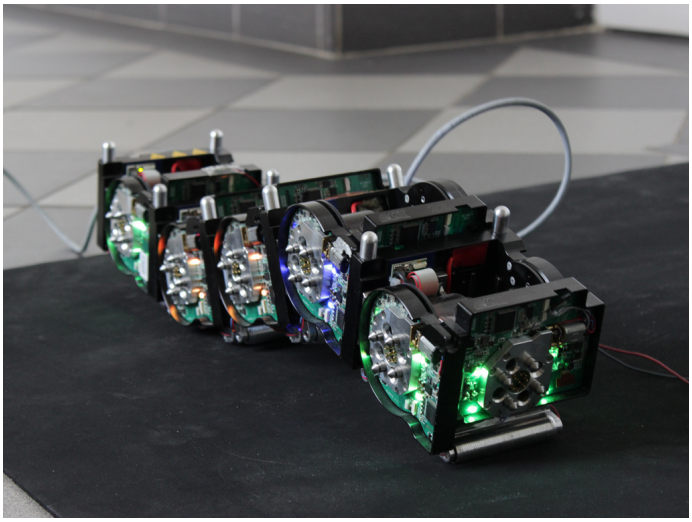


Figure 1.1: Validation of results with a real robot.

needed fitness evaluations is desired. A Fitness Estimation method, that uses an approximate estimated value of fitness instead of calling the fitness function, was applied to both PSO and GA.

In the real world, any module can break down at any time. In this diploma thesis, a preventive method was investigated, where the failures are taken into account during the optimization of the CPG parameters. We also tried adding a simple feedback to one of the CPGs to make it capable of active reaction to failures.

A comparison of automatic locomotion generation using different optimization algorithms and different CPG types was made during the experimental part. Also, the performance of Fitness Estimation method and generation of failure-resistant locomotion patterns were examined. All experiments were run for six different shapes of a robot. To verify the simulator, some of the found locomotion patterns were run on real robots and resulting movements were compared to simulations.

Chapter 2

State of the art

2.1 Modular robotics

A modular robot is a mechanical organism composed of either heterogeneous or homogenous modules. Advantages of using this architecture is a wide variability of organisms, achievable by reassembling modules, and also a capability of replacing disabled modules by a spare device of the same type. As there may be one or relatively few types of the module, the mass production could be possible, which is effective from an economic viewpoint. If the hardware architecture provides self-reconfiguration [1], a robot can autonomously modify its structure in a response to environmental changes or a module defect. Therefore the robot is highly adaptive as it have advantages of the every shape it can be reconfigured to. For example a legged robot is able to walk on a rough terrain and to pass through a narrow space it can reconfigure to a snake-like robot. An autonomous self-reconfiguration is very useful especially in dangerous, distant or inaccessible environments where it is impossible for a human operator to intervene. A possible issue of self-reconfiguration is that it may require cooperation of all modules which might not be possible in case of failures. Moreover, self-reconfiguration requires precise movements which cannot be easily achieved in rough terrains. In this diploma thesis, only fixed-structure organisms are examined, which preserve a single configuration and can be operated through controlling joint angles of their modules.

The task of self-reconfiguration and its usage for changing the structure and for a locomotion is described for various robot types in [2, 3, 4]. An distributed approach capable of finding a time-efficient solution of the self-reconfiguration task for organisms with very large number of module is proposed in [5]. The parallel path planning is defined as a Markov decision problem and solved using dynamic program-

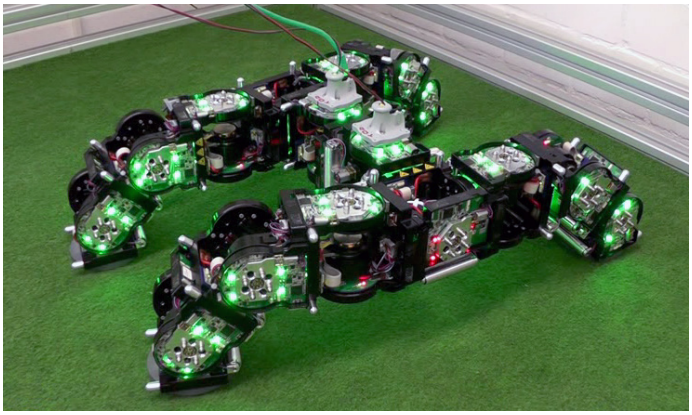


Figure 2.1: A four-legged robot composed of 16 modules.

ming. The organism must remain connected whole time. The proposed method uses depth-first search implemented with message passing between modules to find the mobile modules, which can move without the structure being disconnected. If a path connecting all neighbors of the module is found, the module is identified as mobile. This method provided plans with running time proportional to $\sqrt[3]{m}$ for cubic robots with m modules.

The article [6] discusses the problem of autonomic repairing an organism by replacing broken modules. They use homogenous organisms with numbers of modules which are capable of self-assembling according to defined constraints using local inter module communication. The self-assembly process is made layer by layer. If a missing module is detected, the organism is degenerated to a stage when this module was added and then the assembling procedure runs again.

A basic building block of the modular robot called module has usually a small number of DOFs (usually one or two). Connecting several modules together results into a robot with many DOFs which requires a precise collaboration of all modules to perform any action, to maintain stability or to prevent self-collisions between the modules. Recent robots includes communication system between modules that allows synchronization of actions whether the control is distributed or control inputs are provided centrally e.g. by one of the modules. A low level controller for adjusting joint angles is implemented in each module processor. In this diploma thesis, the CoSMO robotic cubes were used for experiments.

Figure 2.2: Different types of modules (from left to right): Scout robot, Active wheel and Backbone robot (CoSMO).

2.1.1 CoSMO modules

The CoSMO modular robotic platform was developed within EU projects Symbrion and Replicator [7]. The modules are cubes of size 10x10x10 cm and weight≈1 kg (Fig. 4.1a). The modules are equipped with a pair of screw-drives that provides 2D locomotion. The screw-drives allow the modules to operate fully autonomously, so the modules can be used also as stand-alone mobile robots. 3D locomotion is achieved using powerful hinge that is able to lift up to five other modules. Besides the actuators, the modules are equipped with four docking mechanisms, camera, IR receivers/transmitters, RGB sensors and accelerometers. The data can be processed by the on-board Blackfin CPU. The modules communicate with each other using Ethernet, Bluetooth or Zigbee. The modules can be combined with each other using an active docking mechanism. Each module can operate using an internal battery, and the modules can share energy when connected to an organism.

The CoSMO modules are part of large modular robotic system, that consists of three types of modules: "active wheels", "scouts" and "backbones" (Fig. 2.2). The backbones are realized using the CoSMO modules. The three modules are specialized to different tasks. The scouts robots are used to explore 2D environments, backbones/CoSMOs are designed mainly for 3D locomotion and active-wheel modules serve as a backup modules.

The 3D locomotion of the CoSMO robots is realized using the main hinge, which can move in range $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$. The hinge is controlled by setting a desired angle and its motion is controlled using an internal PD regulator. Although the joint can move

very fast (the whole motion from $-\frac{\pi}{2}$ to $\frac{\pi}{2}$ can be realized in less than a second), the fast movements are not used and the modules move very slow.

2.1.2 Failures in modular robotics

As mentioned before, an important advantage of the modular robot architecture is the possibility to cope with the failures. The basic solution of a module malfunction is to disconnect the broken module and to replace it with a spare robot using autonomic self-reconfiguration. The issues of this method may be an inability to undock the broken module due to mechanical or electrical failures, an impossible self-reconfiguration due to a rough terrain or the unavailability of a spare functional module. In case the self-repair by self-reconfiguration is unfeasible for any reason, the organism must be able to finish the task or transport itself to a location where the repair can be done, even with the presence of failures. Recovering from failures by adapting the robot locomotions instead of exchanging the broken parts is investigated in [8]. The presented method, including the adaptation of the locomotive primitives after failure, was verified to significantly increase the ratio of successful plan fulfilling in most cases, in comparison to only updating the high-level path planner to cope with a lame locomotion.

For a modular robot consisting of m modules the number of possible failures N_m is equal to:

$$N_m = \sum_{i=0}^{m} \binom{m}{i} \cdot k^i = (1+k)^m, \tag{2.1}$$

where k is a constant number of considered module defects relevant to the locomotion, for example a discrete set of joint angles where the module might get stuck or free-wheel run if an actuator is broken. If we presume that at most n modules can break, the formula changes to:

$$N_m = \sum_{i=0}^{n} \binom{m}{i} \cdot k^i. \tag{2.2}$$

For a robot assembled from 6 modules, considering three defect types and at most 2 broken modules, there are yet 154 possible functional states of the robot. With the growing number of modules or defect types this count increases rapidly, thereby it is computationally intractable to prepare a set of locomotion patterns for all failure states in advance. For example for the four-legged robot depicted in Fig. 2.1 consisting of 16 modules and with other conditions left as in the previous case, the number

of combinations is 1129. Also, in the real world, the number of potential module defects k is very high.

The method presented in [8] tries to adapt the locomotion under failures on-line. It uses the original locomotion as an initial solution and optimizes it on the broken robot. This method requires a mechanism for failure detection and a robot capable of optimizing the locomotion in the field, that may be difficult to guarantee. In this diploma thesis, another approach was investigated, which could be used when it is impossible to detect module defects or alter the locomotion of the robot on-line. We try to overcome failures of the modules by providing such a robust locomotion pattern, that it would work sufficiently for a healthy robot as well as for a robot under different failures.

2.2 Automatic locomotion pattern generation for modular robots

A key property of each autonomous robot is the ability to move through an environment, which can include various obstacles. Movements resulting in progression of an individual from one place to another is called locomotion. As we almost never know robot's workspace precisely, the locomotion should be robust enough to proceed in as wide range of conditions as possible. For example, the locomotion pattern should be applicable for different types of surfaces (slippery, rough) and should be able to handle with some amount of bumpiness of the terrain. One robust locomotion is easier to achieve than several ad hoc locomotion patterns being switched due to actual circumstances.

The locomotion of a modular robot can be achieved by a repetitive self-reconfiguration or joint-control approaches. In the case of a motion through self-reconfiguration, modules from back part of the robot are being transported to front and that results in organism movement. This approach is not very fast for long locomotions over large distances, because modules have to be disassembled, sent through the whole organism and reassembled at a new position in every step, but on the other hand it can solve difficult tasks like climbing over a high obstacle. The second way is to preserve the only configuration of the modular robot, use module joint motors to act like organism muscles and perform a whole-body motion. The latter locomotion, called joint-locomotion, is the one investigated in this thesis.

Both approaches for locomotion generation (motion through self-reconfiguration and joint-control locomotion) can be solved using planning approaches. In the former case, the self-reconfiguration can be formulated as a discrete planning problem, where

the task is to find a sequence of actions to change the structure of the robot [9, 10, 2, 6, 3]. In this case, it is assumed that each module can perform several actions (e.g. disconnect, move to left, right, up, down across its neighbors and reconnect back to the organism).

The joint-control locomotion can be formulated as a motion planning problem, where the task is to derive control signals for each actuator that would move the robot. This would require a motion model of the modular robot in order to determine how the pose of the robot changes after a control input is applied. Generally, the motion of modular robots depends on their kinematics (that can vary in the case of self-reconfigurable robots), as well as on contacts with the underneath terrain. Therefore, it is not easy to derive an analytical motion model [11]. A motion of the robot can be modeled using physical simulation instead [12, 13].

The motion planning problem for joint-control robots can be solved using sampling-based approaches like Rapidly Exploring Random Tree [14, 15]. The sampling-based approaches can cope with many-DOF systems and they allow us to consider kinematics and dynamics constraints. The methods can be easily extended to utilize a black-box motion model, which is necessary in the case of simulated motion model. The disadvantage of the RRT method is the necessity to derive control signals in order to expand the configuration space of the robot. In the case of modular robots, this would require to discretize the possible control inputs (desired angles of the joints) and to generate combinations of these discretized values.

Such an approach can be applied only to modular robots with few actuators (e.g. snakes with three modules), but it cannot be used for robots with many modules. The number of combinations grows exponentially with the number of actuators [16], which increases the complexity of the planning task. Although a random subset of the combinations can be used [14], a better approach is to utilize locomotion generators in the planning task [16, 17]. A possible method to reduce the dimensionality of this problem to a reasonable level is usage of Central Pattern Generation (CPG), where the control inputs are generated by a set of oscillators. In CPG, a feasible setting of parameters of the oscillators is being found, which is an easier task than finding the control inputs directly and is accomplishable even for robots with many modules.

2.2.1 Central Pattern Generators

In order to transport themselves between different places, animals perform less or more complicated rhythmic moves. Formerly it wasn't clear if this movement is caused by chains of reflexes or by central oscillators [18]. By isolating animal's nervous system and comparing its output to an output observed during motion, it was

proved that there is no need for feedback from environment to generate rhythmic motor pattern [19]. Although the sensory feedback is unnecessary for generating rhythms, together with the higher-level neural center it allows adaptation of the motor pattern to the current situation or even learning new patterns. Neural networks responsible for creating rhythmic patterns are called Central Pattern Generators. Besides locomotion CPGs generate every stereotyped patterns such as breathing, chewing or peristalsis.

As robot structures are often inspired by living creatures it is natural to try to imitate also the principles of their locomotion. Central Pattern Generators can be modeled as a set of coupled oscillators, which is appropriate to use in a distributed system - like in a modular robot [20], where we can assign a single oscillator to each module and use its output to control it. The coupling of oscillators means that an output of a oscillator can be influenced by a state of another oscillator. The main motivation for using CPGs is that it can significantly reduce the dimensionality of the problem of a locomotion generation. Instead of generating control signals directly, we only need to find feasible parameters of the oscillators that emerge into a desired high-dimensional locomotion.

An example of using CPG in modular robotics may be [21], where a serpentine bot is driven by CPGs with near-neighbor coupling. In [22], the switching between walking and swimming gaits of a salamander robot is achieved by adjusting the CPG parameters on-line. CPG with neural oscillators was used to generate locomotion for six different shapes of robots in [23]. The parameters of neural oscillators were obtained from a genetic algorithm.

For trivial situations the parameter values can be assigned manually. For example for a snake-like robot driven by CPG with simple harmonic oscillators (described in 3.1.1) a locomotion can be achieved by setting frequencies and amplitudes equally for all modules and setting the phase shift value of the i-th module to $\phi_i = \frac{\pi}{c} \cdot i$. This setting emerges into a waving movement of the organism, where c is a constant determining the wave shape. Otherwise, in nontrivial cases, the optimization methods have to be used to set CPGs up. Because the influence of individual CPG parameters or even direct influence of a single module on the resulting behavior of the whole organism is generally unknown, the optimized fitness function is black-box type. The appropriate optimization algorithm to be used without any initial knowledge of the problem could be an evolutionary algorithm or the particle swarm optimization algorithm.

An example of locomotion generated by CPG is shown in Figure 2.3, that consists of a graph with control signals obtained from harmonic oscillators and snapshots of the driven robot in the simulation environment.

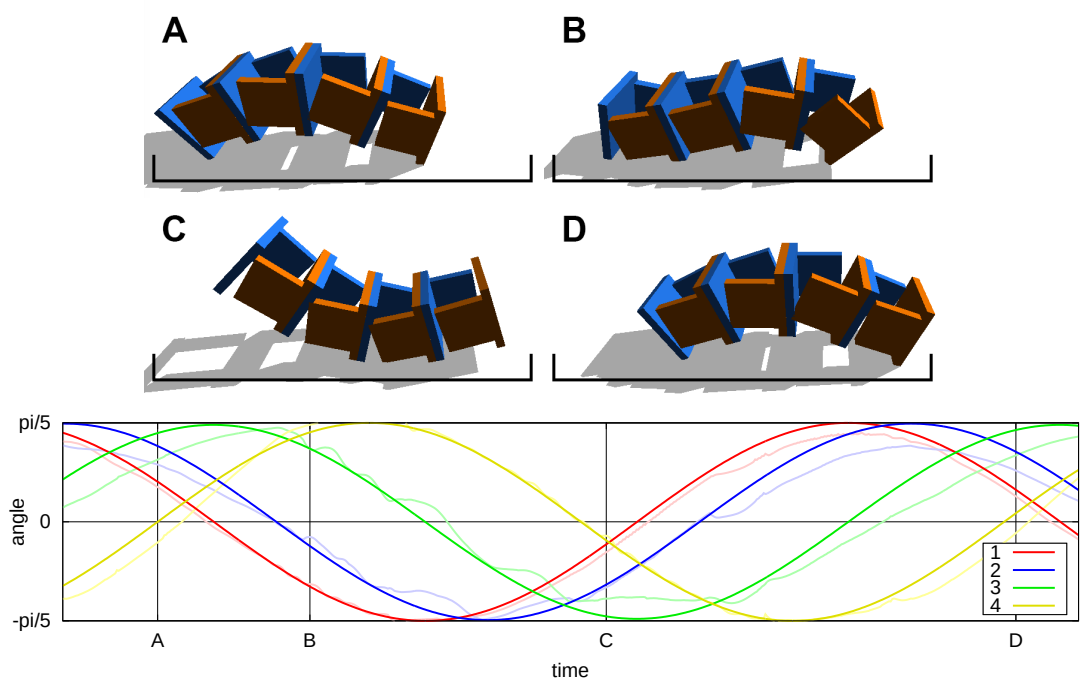


Figure 2.3: A snake-like robot driven by harmonic oscillators. The modules are numbered from left to right. The lighter lines stands for real positions of actuators. The snapshots of the robot corresponds to time points marked in the graph.

2.3 Parameter optimization

Central Pattern Generators can be used to obtain a whole-body locomotion. A CPG model consists of a set of oscillators with some parameters that have to be set properly. As these parameters affect only one particular oscillator, it is not apparent how it influences the locomotion of the whole organism. For nontrivial cases the resulting fitness function is a black-box, so we have to use an optimization algorithm based on the trial and error method. In this diploma thesis, two bio-inspired methods were investigated - a Genetic Algorithm (GA) and a Particle Swarm Optimization algorithms (PSO). The former belongs to evolutionary algorithms that try to imitate natural evolution of living organisms and use its principles to evolute a population of feasible solutions. The latter reproduces movements of individuals in insect swarms, fish schools or bird flocks but instead of the three-dimensional world the individuals move in a n-dimensional problem space while examining it.

The advantage of the methods is the ability to cope with high-dimensional prob-

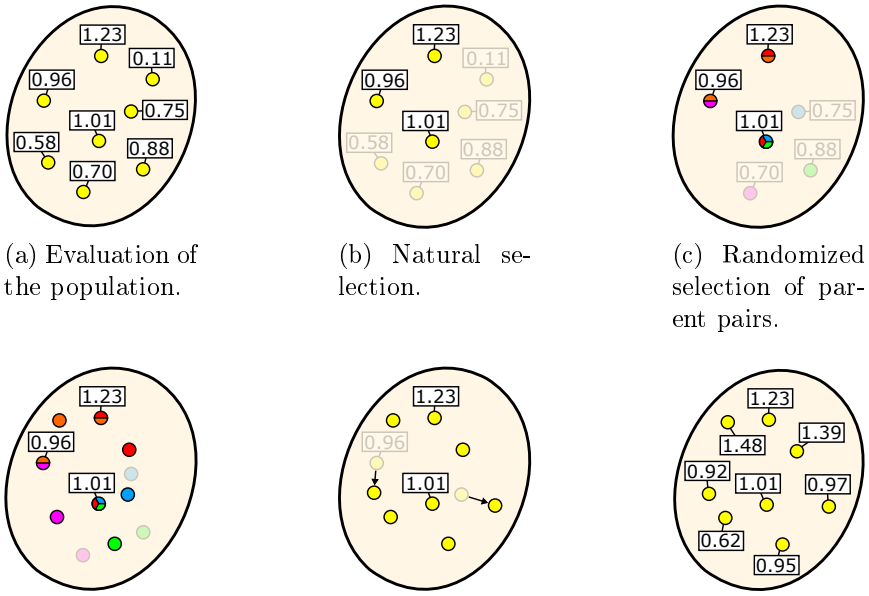
lems. Another important advantage of the bio-inspired methods is the fast convergence to an acceptable solution, which is crucial in the case of the locomotion generation, where the evaluation of the fitness function is very time consuming.

2.3.1 Genetic algorithms

The natural evolution is evidently a very robust process. Diverse lifeforms have evolved and adapted themselves to survive in almost all thinkable places on the earth, from cold poles to the heat equator, in water, on land or in air. The evolution can be considered as an optimization process with the fitness function defined implicitly by the ability to survive, reproduce and pass own genetic information to next generations. Evolutionary algorithms intend to model perfectly time-proven principles of the natural evolution and use their strength to search a space of solutions to different problems. There are more variations of evolutionary computing but the main structure often remains [24]. Initially a population of random solutions is generated and then it is iteratively improved by repeating stochastic operations of selection, recombination and mutation until some condition is satisfied. Probably the best known evolutionary algorithms are genetic algorithms.

Classical genetic algorithms usually don't compute directly with the vectors from the solution space but use a discrete binary alphabet to encrypt them into vectors of zeros and ones. The genetic algorithms terminology is derived from molecular genetics so these binary vectors, which are analogical to DNA chains, are called genotypes or chromosomes. By decrypting chromosome we get a corresponding solution or, in terms of genetics, a phenotype.

Genetic algorithm (Alg. 1) starts with a set of arbitrarily selected chromosomes called population. Then every chromosome is evaluated by fitness function, which quantify the quality of the solution stored by the chromosome. A natural selection ensures that the fittest chromosomes are kept in the population and those with insufficient fitness values are forgotten. The ratio between kept and rejected individuals may be set arbitrary. The population gap developed after the selection is then filled by a crossover, which is the most important operation in GAs, differing them from other evolution algorithms. Chromosomes are paired randomly and the fitness value of the individual may be taken into account when defining the probability of its mating. The offspring chromosome is generated as a combination of parents, while there exist several methods how to combine the chromosomes. The final step is a mutation. In classical binary representation it means that any bit among the chromosomes in the population can swap its value from zero to one and vice versa with a given small probability. The mutation can prevent a premature convergence and find solutions



(a) Evaluation of the population.

(b) Natural selection.

(c) Randomized selection of parent pairs.

(d) Crossover.

(e) Mutation.

(f) Evaluation of the next generation.

Figure 2.4: The schema of a basic Genetic Algorithm. The numbers in boxes are the fitness values. During the selection of parents, the individuals which are about to mate have the same color, and after the crossover their offspring also has this color. The semitransparent individuals are the members of the original population that were not chosen to proceed to the next generation.

which were not present in the original population. The algorithm continues using the new population of chromosomes for a next iteration of the selection, the crossover and the mutation until the population converges or some another condition is met.

The modifications of the genetic algorithm for real-valued chromosomes also exists [25]. They use the same steps as the classical GA and differ only in the implementation of the mutation and crossover operators. The chromosomes in a continuous genetic algorithm can be vectors straight from the space of solutions so coding between the chromosomes and the solutions is not necessary any more. The real-coded genetic algorithm is superior for optimizing multimodal functions or variables with many local minimums [23] and therefore it was used also in this diploma thesis for the CPG parameter optimization.

```
// initialize population
pop[1..N] := random chromosome;
// decode chromosomes and evaluate their fitness
evaluate(pop);
repeat
    // select fit chromosomes to be copied into next generation
    pop_new[1..N_keep] := selection(pop);
    // pair chromosomes and recombine them to fill the rest
    pop_new[N_keep + 1..N] := crossover(pop);
    // modify some chromosomes randomly with small probability
    pop_new := mutate(pop_new);
    pop := pop_new;
    // decode chromosomes and evaluate their fitness
    evaluate(pop);
until condition;
```

Algorithm 1: Basic genetic algorithm

2.3.2 Particle Swarm Optimization

An ant colony is capable of the complex social behavior including building ant-hills, defending enemies or food seeking although a single ant is credited with minimal individual intelligence and does not plan his actions in manner of achieving long term outcome. The individuals behave according to simple rules like a pheromone following and working together they are capable to solve a global problem, like selecting the shortest path of many [26]. The pheromone, which is laid down by the ants, attracts other ants. If the path is efficient, the ants shuttle faster and leave more pheromone there which provides a positive feedback. The phenomenon of simple individual behavior emerging into complex results is known as swarm intelligence [27]. Beside the ant colony behavior, an another example of the swarm intelligence is the bird flocking. A simulation of the complicated movement of the whole flock can be obtained easily if the problem is decentralized. The flock members are influenced by local forces that are easy to determine. These may be for example a collision avoidance, velocity matching with their neighbors and moving toward the center of the flock.

The particle swarm optimization algorithm [28, 29, 30, 31] prospers from the swarm intelligence phenomenon as well and is inspired by the social behavior of individuals in swarms or flocks. A population of particles swarms through the solution

space and each particle is attracted to the best position it has visited so far and to the global best position, which is the best position found by the particle's neighborhood. As an initialization, a set of particles is generated arbitrary. Every particle consists of a position, which is a vector from the solution space, and of a velocity vector determining the position change in the succeeding step. Each particle also keeps track of its best evaluated position and of the best evaluated position of its neighborhood, also called the global best. In every step the particle moves according to its velocity and then the velocity is updated as stated in equations:

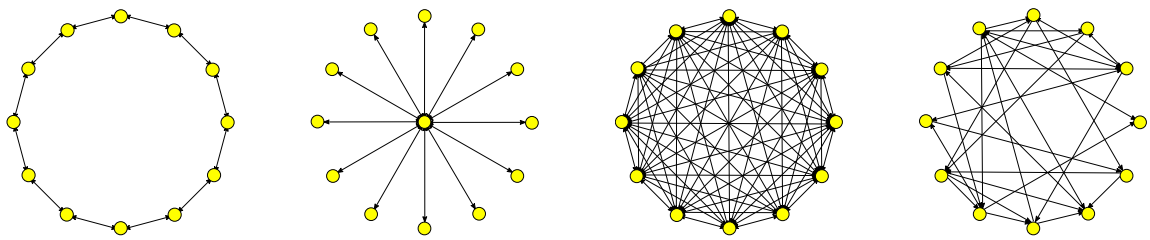
$$\begin{aligned} \mathbf{x}_i^{k+1} &= \mathbf{x}_i^k + \mathbf{v}_i^k \\ \mathbf{v}_i^{k+1} &= \mathbf{v}_i^k + c_1 \cdot \mathrm{rnd}(0,1) \cdot (\mathbf{pbest}_i^k - \mathbf{x}_i^k) + c_2 \cdot \mathrm{rnd}(0,1) \cdot (\mathbf{gbest}_i^k - \mathbf{x}_i^k) \end{aligned} \quad (2.3)$$

where $\mathbf{x}_i^k$ is the position of i-th particle in k-th step, $\mathbf{v}_i^k$ is the velocity vector of this particle, $\mathbf{pbest}_i^k$ is the particle's current best position achieved so far, $\mathbf{gbest}_i^k$ is the current best position of the i-th particle's neighborhood, $\mathrm{rnd}\,(0,1)$ is a random number between 0 and 1 with the uniform distribution, $c_1$and c_2 are weight constants determining the importance of the personal and global best values. Usually these constants are set equally $c_1 = c_2 = 2$ to make the mean of stochastic values $c_1 \cdot \mathrm{rnd}(0,1)$ and $c_2 \cdot \mathrm{rnd}(0,1)$ equal to 1 [28]. After the particle's position change, the new fitness value is computed and if it is better than the fitness of the particle's best position, the actual position is stored as **pbest**. To prevent a gain of oscillations of particle positions, velocity should be damped. A simple solution is to define a vector $\mathbf{v}_{max}$ and make sure that each dimension of the actual particle velocity fits in the interval $[-v_{max,d}, v_{max,d}]$, where $v_{max,d}$ is the maximal speed in the tested dimension. In this diploma thesis, the maximal velocity is given by a scalar parameter v_{max}, that determines the maximal speed as a percentage part of each dimension size of the solution space. Thus, the vector $\mathbf{v}_{max}$ is computed as $\mathbf{v}_{max} = v_{max}\,(\mathbf{ubound} - \mathbf{lbound})$, where **ubound** and **lbound** are vectors of upper and lower boundaries of each dimension. The PSO algorithm in a pseudo-code is shown in Alg. 2, where $G()$ stands for a fitness evaluating function. Other symbols are the same as that in Eq. 2.3.

The stated algorithm intentionally lacks the way how the particles are clustered into neighborhoods, as there are more approaches for implementation of this. Instead of geometrical neighborhood constructed in terms of particle distances in search space, the topological structures are more common. In [27] there are two topologies compared. The *circle* topology, also called the *ring* or *lbest*, where each particle is connected to its k immediate neighbors (Fig. 2.5a), and the *wheel*, where one particle is connected to all others and they are connected only to that one (the origin of the name "wheel" is obvious from Fig. 2.5b). In the circle topology distant particles are

```
// compute the maximal speed vector
v_max = v_max (ubound − lbound);
// initialize a swarm of particles
for i := 1 to particle count do
    x_i := random position;
    v_i := random velocity;
    pbest_i := x_i;
    gbest_i := x_i;
end
repeat
    // run following loop for every particle
    for i := 1 to particle count do
        // update particle best
        if G(x_i) > G(pbest_i) then
            pbest_i := x_i;
        end
        // update the neighborhood best
        for j := indexes of neighbors do
            if G(pbest_j) > G(gbest_i) then
                gbest_i := pbest_j;
            end
        end
        // update the position and the speed
        x_i := x_i + v_i;
        v_i := v_i + 2 · rnd(0,1) · (pbest_i − x_i) + 2 · rnd(0,1) · (gbest_i − x_i);
        // ensure the maximal velocity in all dimensions
        for d := number of dimensions do
            if v_id < −v_max,d then v_id := −v_max,d;
            if v_id > v_max,d then v_id := v_max,d;
        end
    end
until condition;
```

Algorithm 2: Basic PSO algorithm



(a) Circle, $k = 2$ (b) Wheel (c) gbest (d) random, $p_{conn} = 0.25$

Figure 2.5: Examples of swarm topologies.

independent of each other, but neighbors are connected. In the wheel topology all information is communicated through the focal individual, thus other individuals are isolated. From the experimental comparison a hypothesis it was proposed that the wheel topologies may perform better by maintaining diverse population on functions with many local optima. Another basic topology is *gbest*, where all particles communicate with each other (Fig. 2.5c), thus there is only one common best position at a time. The gbest topology often suffers by the premature convergence in a local optimum. As the character of the fitness function used for optimizing CPG parameters in this thesis is not known, a randomized topology of the particle interaction was used (Fig. 2.5d). The selected topology is described below in 3.2.1. This approach is reported to perform equally or better than selected deterministic topologies in [32].

The basic improvement of PSO algorithm is an introduction of inertia weight ω, described in [30]. The parameter defines the willingness of a particle to change its speed according to the personal and neighborhood best positions. The modified equation for velocity changes is:

$$\mathbf{v}_i^{k+1} = \omega \mathbf{v}_i^k + c_1 \cdot \mathrm{rnd}\,(0,1) \cdot \left(\mathbf{pbest}_i^k - \mathbf{x}_i^k\right) + c_2 \cdot \mathrm{rnd}\,(0,1) \cdot \left(\mathbf{gbest}_i^k - \mathbf{x}_i^k\right), \quad (2.4)$$

where ω stands for the inertia weight. In the classic PSO, this value equals one. Increasing ω results into searching the wider part of the search space, but in the final phase of the optimization the particles can oscillate around better solutions. Decreasing results in a more precise local search but the particles can easily get stuck in a local optimum. If the value of ω is initialized at higher value and linearly decreased during optimization process, the algorithm searches the bigger space in the beginning and in the final stadium it performs a partial search. The fine setting of ω can improve the speed of algorithm convergence while searching sufficient part

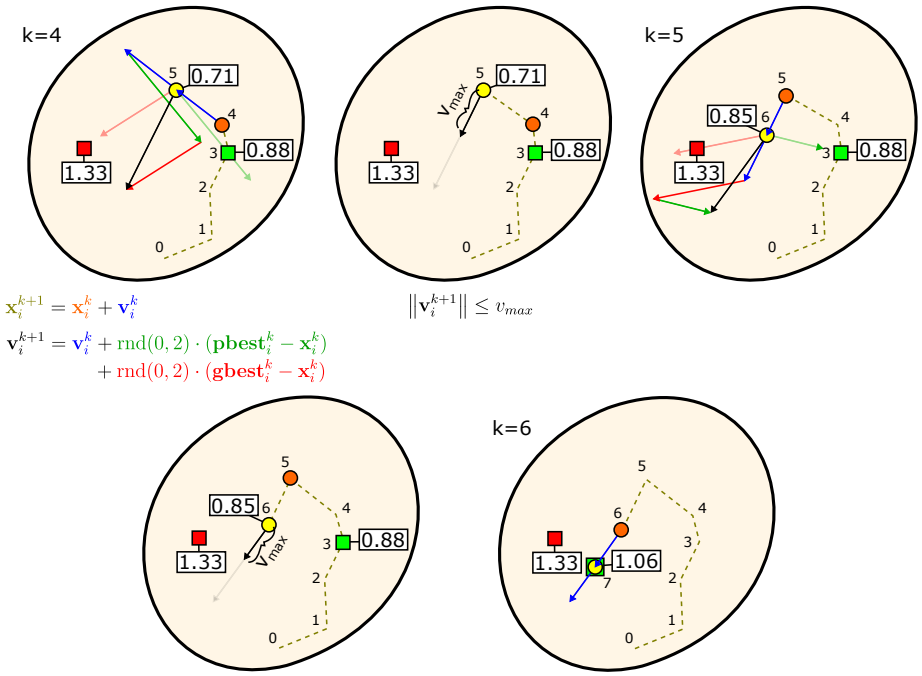


Figure 2.6: Explanation of the particle movement in the search space. An orange circle represents the previous position of the particle $\mathbf{x}^k$, a yellow circle represents the updated position $\mathbf{x}^{k+1}$, a green square stands for **pbest**, the best position the particle has visited and a red square is **gbest**, the best particle found by the neighborhood of the particle. The numbers in boxes are the fitness values. In the last picture the **pbest** position is updated, as the particle passes a point with a fitness value 1.06, which is higher than the actual **pbest** fitness 0.88.

of the search space.

2.4 Conclusion

Modular robotics is a versatile technology applicable especially in fields with high demands on a device autonomy. An organism assembled from a set of modules may be capable of self-reconfiguration and change its shape according to actual needs, like for climbing over a high obstacle or passing through a narrow space. The locomotion of a modular robot can be achieved by either repetitive self-reconfiguration, where modules from the back of the robot are being transported over other modules to the front, or by joint-control approaches, where the robot structure is fixed and the

organism is moved by cooperation of actuators of the modules. The latter is studied in this thesis.

As computing the proper joint angles for all modules directly would be complicated, the CPG method can be used to simplify the problem. In this method a set of coupled oscillators is used to generate the control inputs. The CPG sets constraints of the generated signals so the dimension of the solution space is decreased to the number of parameters of the oscillators. An adequate setting of the parameters ensures the desired locomotion of the organism.

To find feasible parameters automatically, optimization methods must be used, with the fitness value determined by a measured value from the locomotion generated by the currently evaluated parameter, e.g. a change of the coordinates of the organism after a specified running time. The number of the parameters usually grows linearly with the number of the modules and a curse of the fitness function is unknown, therefore the appropriate optimization methods can be a genetic algorithm or the particle swarm optimization, that are proved to be efficient in such types of problems. The evaluation of the fitness function is very time consuming in the locomotion learning and it dominates the overall optimization time, therefore minimizing the number of evaluations is desired.

In the next chapter two particular variants of the CPG controller and concrete implementations of GA and PSO are described. A method of decreasing the number of fitness evaluations for both optimization algorithms is also proposed.

Chapter 3

Investigated solutions

3.1 Locomotion Generation

The central pattern generation uses a set of oscillators that generate control inputs. In modular robotics, we can assign the oscillators to modules and use their outputs as a reference signal for the module position controller to maintain an organism locomotion. In this thesis, two kinds of CPG were tested. The former uses the simplest oscillating function — the harmonic oscillator, whose output is described by a sinus function and is characterized by an amplitude, a frequency and a phase shift. The latter is implemented as a chain of coupled nonlinear oscillators. Next to the intrinsic amplitude and frequency that have the same meaning as in the harmonic oscillator, this approach contains also parameters that define connections between oscillators and their relative influences. Performance of both of them was examined experimentally with different robot shapes.

As there are physical and mechanical limitations in controlling a real module, some boundaries must be applied also for the parameters of the central pattern generators to get a reasonable behavior. For the sinusoidal CPG the setting of the boundaries is done intuitively. The boundaries of the parameters of the coupled nonlinear CPG, in contrast, are hard to set to maintain a desired signal shape. We found a feasible settings for our experiments experimentally using the Fast Fourier Transform to verify that the generated signal doesn't contain too high frequencies.

3.1.1 Harmonic oscillators

The simplest implementation of CPG composes of a set of harmonic oscillators. The control input of i-th module is given by function:

$$u_i(t) = A_i \cdot \sin(\omega_i t + \phi_i) \tag{3.1}$$

There are three parameters per module: A_i is an amplitude, $\omega_i = 2\pi f_i$ is an angular frequency and ϕ_i is an phase shift. The overall number of parameters is $3m$ for a modular robot consisting of m modules. The amplitude and frequency boundaries are defined by mechanical attributes of a module. The amplitude matches the range of module joint angles a the frequency is limited by the maximal speed of joint movement.

3.1.2 Coupled nonlinear oscillators

The second implemented variation of CPG was based on [22] where it is used to generate two different gaits of a salamander inspired robot, walking and swimming, by modifying parameters of the oscillators. Compared with sinusoidal CPG, the coupled nonlinear CPG allows coupling between oscillators. The system of coupled nonlinear oscillators is described by following set of differential equations:

$$\begin{aligned} \dot{\theta}_i &= 2\pi f_i + \sum_{j \neq i} w_{ij} r_j \sin(\theta_j - \theta_i - \phi_{ij}) \\ \ddot{r}_i &= a_i \left(\frac{a_i}{4}(R_i - r_i) - \dot{r}_i \right) \\ u_i &= r_i(1 + \cos(\theta_i)), \end{aligned} \tag{3.2}$$

where θ_i and r_i are state variables representing the phase and the amplitude of the i-th oscillator. The parameters f_i and R_i defines the intrinsic frequency and amplitude of the oscillator and w_{ij} and ϕ_{ij} determines the weight and the phase bias of the j-th oscillator state with respect to the i-th oscillator. The coupling parameters are not necessarily symmetric, thus w_{ij} may differ from w_{ji}. The parameter a_i is positive constant affecting the response of output on R_i changes. The phase shift is given by the initial condition $\theta_i(0)$. All locomotions in this thesis start with steady modules set to zero angles, therefore initial conditions $r_i(0)$ and $\dot{r}_i(0)$ are set to zero.

If $r_i(0)$, $\dot{r}_i(0)$ and a_i are set manually and fixed, there are $m\,(3 + 2(m-1)) = 2m^2 + m$ parameters to specify for a modular robot consisting of m modules. f_i, R_i and $\theta_i(0)$, with a meaning similar to the parameters f_i, A_i and ϕ_i of sinusoidal CPG, and w_{ij} and ϕ_{ij}, defining the strength of the coupling between two oscillators and their phase bias. To generate signals that are feasible for used modules, the upper limit of R_i is set to the value of the maximal angle of the joint. The instantaneous frequency of the coupled nonlinear oscillators depends on an intrinsic frequency f_i but is also influenced by the states of the linked oscillators weighted by the parameter

w_{ij}, so the setting of the parameters boundaries is not clear. We used the Fast Fourier Transform in MATLAB to determine which frequencies are present in the outputs of the oscillators and lowered the range of the w_{ij} boundaries until the frequencies were appropriate.

3.2 Parameter optimization

The resulting whole body locomotion depends on the collaboration of all modules. In case of using central pattern generators the locomotion is given by a properly set combination of parameters for all oscillators. As denoted in the previous subsection, the sinusoidal CPG and the coupled nonlinear CPG have $3m$ and $2m^2 + m$ real valued parameters to be set, respectively, where m is the number of modules in the organism. Acquiring these parameters is the optimization problem with the fitness function whose value is determined by some measured output of a locomotion emerged from the evaluated parameters. This output may be for example the change of some coordinate after the specified amount of time for which the locomotion was executed. The fitness function is defined implicitly by the used CPG method, the physical and mechanical parameters of modules, the structure of the organism, the environment etc., therefore the only way how to get the fitness function value for a set of parameters is to actually drive the robot using CPG with these parameters and observe its behavior.

As the number of dimensions of the search space is a multiple of the module count and the course of the fitness function is unknown, gradient optimization methods are not efficient for this problem. In contrast, genetic algorithms and the particle swarm optimization perform good results on high-dimensional and black-box problems. In this diploma thesis, both of them were implemented and their performance in finding the parameters for different robot shapes was investigated. It usually takes many fitness evaluations to find a feasible combination of CPG parameters. As measuring the fitness value on a real hardware is very time-consuming, the locomotion finding methods were tested in a simulator, but still the fitness evaluation consumes the most significant part of the computation time. Methods decreasing the number of evaluations were investigated for both optimization algorithms.

3.2.1 Particle swarm optimization

The particle swarm optimization is an optimization algorithm based on swarm intelligence. The swarm of particles representing possible solutions imitates the behavior of a insect swarm or a bird flock to examine the search space and to find a solution

with the highest fitness value. For the purposes of this diploma thesis the basic PSO algorithm was implemented. The algorithm structure is the same as in Alg. 2 listed above. To minimize the number of the expensive fitness evaluations, the fitness values of the particle actual position and its best position are stored in memory, so only the new positions after the particle movement are evaluated. The algorithm terminates after a specified number of iterations.

An important part of the PSO algorithm is the velocity update function, which in this case is:

$$\mathbf{v}_i^{k+1} = \mathbf{v}_i^k + \mathrm{rnd}(0,2) \cdot (\mathbf{pbest}_i^k - \mathbf{x}_i^k) + \mathrm{rnd}(0,2) \cdot (\mathbf{gbest}_i^k - \mathbf{x}_i^k) \tag{3.3}$$

This equation corresponds to Eq. 2.3 and Eq. 2.4 with substituted parameters c_1, c_2 and ω. The inertia weight ω is constant for the whole duration of the algorithm run and equals one, the parameters c_1 and c_2 are both set to 2.

```
// generate neighborhood for every particle in the swarm
for i := 1 to particle count do
    for j := 1 to particle count do
        // add particle_j to neighborhood with probability p_c
        if i ≠ j ∧ rnd(0,1) < p_c then
            neigborhood_i := neigborhood_i ∪ particle_j;
        end
    end
end
```

Algorithm 3: Neighborhood generation in PSO

Another key property of the algorithm is the way the neighborhood of a particle is generated, which imply the social behavior of the swarm. There are many approaches for determining the connections between particles that perform differently for different optimization problems. The one used in our implementation of PSO is based on the randomized *neighborhood re-structuring method* described in [32]. The topology created by this method is asymmetric, thus the neighbor of the particle don't have to contain this particle in its own neighborhood (Fig. 2.5d). Neighborhoods of different particles are completely independent. In this method, the connections between particles are set randomly, kept for a specified number of PSO iterations and then re-structured entirely. In our implementation there are parameters p_c and t_{keep} standing for the probability of adding a particle into generated neighborhood and

the number of iterations for the one specific structure remains, respectively. Pseudocode for neighborhoods generation is shown in Alg. 3. After t_{keep} iterations all the neighborhoods are regenerated. The parameters were set $p_c = 0.25$ and $t_{keep} = 10$ for the experiments presented in this diploma thesis.

3.2.2 Genetic algorithm

Genetic algorithms belongs to evolutionary algorithms, thereby they consists of typical evolutionary processes of a natural selection, a crossover and a mutation. They are distinguished from another EAs mainly by a large importance of the crossover operator. An outline of a basic genetic algorithm was shown above in Alg. 1. Classical GAs cope with binary genomes that represent encoded possible solutions, however for optimizing multimodal functions with many local minimums real-coded GAs are more efficient. As an application of genetic algorithm for optimizing CPG parameters was already presented in [23], we adopted the published settings of the algorithm from this paper.

The population is initialized randomly and each individual's fitness value is evaluated. The individuals are sorted by their fitness value and n_{keep} of them are copied into the next generation. The rest of the population of the next generation is filled by crossover. The Unimodal Normal Distribution Crossover (UNDX) method, described below, is used. This method requires three parents for generating an offspring which are selected by a roulette selection method (Alg. 4), where the probability of selecting an individual is proportional to its fitness value. With a probability given by m_rate, every dimension of the chromosome can mutate by being set to a random value from its range. The used mutation operator is shown in 5.

3.2.2.1 Unimodal Normal Distribution Crossover

The Unimodal Normal Distribution Crossover (UNDX) method [33] is a componentwise crossover method which provides a sufficient diversity of children while preserving the original distribution of the parent population. For the n-dimensional real-valued search space $\mathcal{R}^n$ the UNDX method works as follows. The steps of parent selection and offspring generation are shown in Fig. 3.1. At first, two parents $\mathbf{x}^1 \in \mathcal{R}^n$ and $\mathbf{x}^2 \in \mathcal{R}^n$ and are selected from the population and their midpoint x^p and difference vector $\mathbf{d}$ are computed:

$$\mathbf{x}^p = \frac{1}{2}(\mathbf{x}^1 + \mathbf{x}^2) \tag{3.4}$$

$$\mathbf{d} = \mathbf{x}^2 - \mathbf{x}^1 \tag{3.5}$$

```
// roulette initialization
fitness_sum := the sum of all fitness values;
fitness_min := the minimal fitness value;
roulette[1] := (pop[1].fitness − fitness_min)/fitness_sum;
for i := 2 to population size − 1 do
    roulette[i] := roulette[i − 1] + (pop[i].fitness − fitness_min)/fitness_sum;
end
roulette[population size] := 1;
// spinning the roulette
i := 1;
r := rnd(0, 1);
while roulette[i] < r do
    i := i + 1;
end
// the selected individual is pop[i]
```

Algorithm 4: Roulette wheel method. The variable pop[i] is the i-th individual and pop[i].fitness is its fitness value.

```
// mutation
for i := 1 to population size do
    for j := 1 to number of dimensions do
        if rnd(0, 1) < m_rate then
            pop[i].chromosome[j] := rnd(lbound[j], ubound[j]);
        end
    end
end
```

Algorithm 5: The used mutation operator. The variable pop[i] is the i-th individual and pop[i].chromosome[j] is the j-th value of its chromosome. The variables lbound[j] and ubound[j] are the lower and upper boundaries of the j-th dimension.

The vector $\mathbf{d}$ determines the primary search direction. As the next step, the third parent $\mathbf{x}^3 \in \mathcal{R}^n$ is randomly selected from the population. The distance D between $\mathbf{x}^3$ and the line connecting $\mathbf{x}^1$ and $\mathbf{x}^2$ is given by the equation:

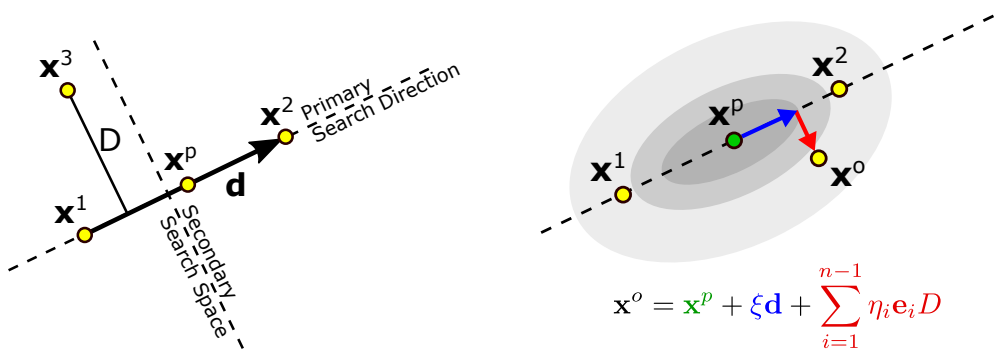
$$D = \left|\mathbf{x}^3 - \mathbf{x}^1\right| \cdot \left(1 - \left(\frac{(\mathbf{x}^3 - \mathbf{x}^1)^T(\mathbf{x}^2 - \mathbf{x}^1)}{|\mathbf{x}^3 - \mathbf{x}^1|\,|\mathbf{x}^2 - \mathbf{x}^1|}\right)^2\right)^{\frac{1}{2}} \tag{3.6}$$

The offspring vector $\mathbf{x}^o \in \mathcal{R}^n$ is given by the equation:

$$\mathbf{x}^o = \mathbf{x}^p + \xi\mathbf{d} + \sum_{i=1}^{n-1} \eta_i \mathbf{e}_i D \tag{3.7}$$

where ξ is a random number following a normal distribution $N(0, \sigma_\xi^2)$ and η_i are $n-1$ random numbers independently following a normal distribution $N(0, \sigma_\eta^2)$. The vectors $\mathbf{e}_1, \ldots, \mathbf{e}_{n-1}$ are normalized orthogonal basis that span the secondary search space, or in other words these are vectors perpendicular to the vector $\mathbf{d}$. These vectors can be obtained using the Gram–Schmidt orthonormalizing process, as described in Appendix B. The recommended setting of the variance parameters based on numerical experiments, as denoted in [33], is:

$$\begin{aligned} \sigma_\xi^2 &= \frac{1}{4} \\ \sigma_\eta^2 &= \frac{0.35^2}{n}. \end{aligned} \tag{3.8}$$



(a) Selection of parents.

(b) Generation of offspring. The meaning of the terms in Eq. 3.7 is denoted.

Figure 3.1: Unimodal Normal Distribution Crossover.

3.2.3 Fitness estimation

As the evaluation of the fitness function is very time-consuming and dominates the overall time of the locomotion pattern generation, minimizing the number of fitness evaluations is desired. A possible method of achieving this is estimating the fitness of an individual instead of computing it. An estimation method for evolutionary algorithms is described in [34]. The fitness of an offspring can be estimated from the known fitness values of its parents. As the accuracy of the estimated values gets lower with a number of estimations, a reliability parameter is associated with each fitness value to specify, how much reliable this values is. If the reliability falls under a defined threshold, the estimated fitness value is discarded and a true fitness value is computed. The reliability varies between 1 and 0, where only true evaluated values have the reliability equal to 1. The reliability of offspring is given by the reliability of the parents and also by the distance of the offspring and the parents in the solution space. The similarity of two individuals is given by the equation:

$$S = 1 - \frac{\sqrt{\frac{\sum_{i=1}^{n}(x_i^1 - x_i^2)^2}{n}}}{2A}, \tag{3.9}$$

where n is the number of dimensions, x_i^1 and x_i^2 are the values of i-th dimensions of the vectors of the two compared individuals. The values x_i^1 and x_i^2 lie between $\pm A$. In this diploma thesis, different boundaries were used for separate dimensions, so the previous equation is modified into:

$$S = 1 - \sqrt{\frac{\sum_{i=1}^{n}\left(\frac{x_i^1 - x_i^2}{ubound_i - lbound_i}\right)^2}{n}}, \tag{3.10}$$

where $lbound_i$ and $ubound_i$ are the boundaries of the i-th dimension. Having the fitness values and reliabilities of the parents, we can compute the fitness and the reliability of their child as:

$$f = \frac{S_1 r_1 f_1 + S_2 r_2 f_2}{S_1 r_1 + S_2 r_2}, \tag{3.11}$$

$$r = \frac{(S_1 r_1)^2 + (S_2 r_2)^2}{S_1 r_1 + S_2 r_2}, \tag{3.12}$$

where S_1 is the similarity between the first parent and the offspring, S_2 is the similarity between the second parent and the offspring and r_1,f_1,r_2,f_2 are the reliabilities and fitness values of the parents.

In [35], this method is extended for usage with the PSO algorithm. In PSO, as stated in Eq. 2.3, the next position of a particle, $\mathbf{x}^{k+1}$ is determined by: 1) its current speed ($\mathbf{v}^{k-1} = \mathbf{x}^k - \mathbf{x}^{k-1}$), 2) its best visited position so far ($\mathbf{pbest}^k$), and 3) the best position found by its neighborhood ($\mathbf{gbest}^k$). Thereby the positions $\mathbf{x}^k$, $\mathbf{x}^{k-1}$, $\mathbf{pbest}^k$ and $\mathbf{gbest}^k$ can be regarded as the parent vectors while the position $\mathbf{x}^{k+1}$ is their offspring. Modifying the above equations for four parents yields:

$$f = \frac{S_1 r_1 f_1 + S_2 r_2 f_2 + S_3 r_3 f_3 + S_4 r_4 f_4}{S_1 r_1 + S_2 r_2 + S_3 r_3 + S_4 r_4}, \tag{3.13}$$

$$r = \frac{(S_1 r_1)^2 + (S_2 r_2)^2 + (S_3 r_3)^2 + (S_4 r_4)^2}{S_1 r_1 + S_2 r_2 + S_3 r_3 + S_4 r_4}, \tag{3.14}$$

where S_i, $i = 1 \ldots 4$, are the similarities of $\mathbf{x}^k$, $\mathbf{x}^{k-1}$, $\mathbf{pbest}^k$ and $\mathbf{gbest}^k$ in respect to $\mathbf{x}^{k+1}$, r_i and f_i are the reliability and fitness values of $\mathbf{x}^k$, $\mathbf{x}^{k-1}$, $\mathbf{pbest}^k$ and $\mathbf{gbest}^k$.

As the change of the particle position is limited by the given maximal velocity, the reliability doesn't decrease so rapidly. In GA, the parents are selected randomly from the population regardless of their relative distance in the search space, so their offspring may be generated far away from its parents and its reliability may be very low. To provide information from the close neighborhood of the offspring and to increase the reliability of the estimated fitness value, we added another two parents, which are selected according to their reliability and similarity to the offspring. The resulting fitness and reliability values are given by the same equations as in the case of PSO, but the four parents are $\mathbf{x}_{p1}$, $\mathbf{x}_{p2}$ - the genetic parents of the offspring, and $\mathbf{x}_{p3}$, $\mathbf{x}_{p4}$ which are two individuals having the highest values of the product $S \cdot r$.

The important parameter of the estimation method is the reliability threshold r_{min}. If the reliability of the estimated fitness drops below r_{min}, the real fitness value is computed and the reliability is set to 1. The lower the r_{min} parameter is, the more fitness values are estimated. For $r_{min} = 1$ there is no estimation at all and the true fitness value is computed every time. The goal is to find such value of r_{min} that decreases the number of real fitness evaluations but keeps enough to sufficiently evaluate the search space. In our experiments we discovered, that there was a thin border, where the number of real evaluations dropped quickly. A method to prevent too many fitness values from being estimated, called Random Fitness Evaluation, is also presented in [34]. If the reliability of an estimated fitness value is higher than r_{min}, with a small probability (e.g. $P_E = 0.05$) the real fitness value is computed. In this diploma thesis, this method was modified in order to be able to explicitly set the minimal rate of the true fitness evaluations. The desired minimal amount of true fitness evaluations is given by a probability P'_E, so that e.g. $P'_E = 0.6$ (used in our experiments) ensures at least 60 % of true evaluations. For every individual

```
// fitness estimation
x.fitness = estimate_fitness(x, p1, p2, p3, p4) (Eq. 3.13);
x.reliability = compute_reliability(x, p1, p2, p3, p4) (Eq. 3.14);
// check if a true fitness value should be computed
if rnd(0,1) < P'_E ∨ x.reliability < r_min then
    // compute a real fitness value
    x.fitness = true_fitness_function(x);
    x.reliability = 1;
end
// continue the optimization algorithm
```

Algorithm 6: Fitness estimation process. The variable x may be a PSO particle or a GA individual. Variables p_1, p_2, p_3 and p_4 are the parents selected as stated in 3.2.3.

a random number is obtained and if it is lower than P'_E, the real fitness value is computed regardless of the reliability of the current fitness.

Another improvements of the fitness evaluation method may be the Error Compensation method [34], where an average difference between recent true fitness evaluations and their estimated value is computed and used to add a simple feedback to following fitness estimations, or the usage of visual parents vectors [35] which is applied in case the offspring does not lie in the area determined by its parents, where the classic convex combination may be inaccurate. From the stated enhancements, only the modified Random Fitness Evaluation technique was implemented for the purposes of this diploma thesis. The process of the used fitness evaluation method is shown in Alg. 6.

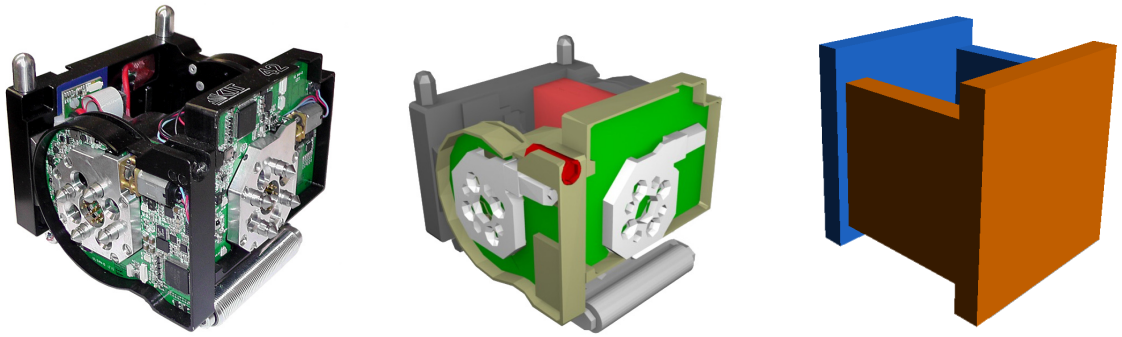
Chapter 4

Experiments

The algorithms for the automatic locomotion generation described in the previous chapter were implemented in C++ and their performance was quantified by several benchmark experiments, using a physical simulator to evaluate the fitness of the locomotion. The locomotion generation was achieved via CPG with two types of oscillators, the simple harmonic oscillators (3.1.1) and the coupled nonlinear oscillators (3.1.2). The implemented methods for parameter optimization of the CPGs were Particle Swarm Algorithm (3.2.1) and Genetic Algorithm (3.2.2). For both of them, the Fitness Estimation method (3.2.3) was also implemented. The methods enhanced by Fitness Estimation are denoted FE-PSO and FE-GA in the rest of the text.

The goal of the first set of experiments was to discover an appropriate setting of the PSO parameter v_{max}, determining the maximal velocity of the particle in search space, and the setting of the parameter r_{min}, the requested minimal reliability of an estimated fitness value to keep this value and not to compute the true fitness value, for both, the PSO algorithm and Genetic Algorithm. In the second set of experiments, all optimization methods were statistically compared with both CPGs and with five different robot shapes. The last benchmark test intended to verify, whether it is possible to increase the robustness of locomotion patterns by training them on robots with simulated failures.

All experiments were realized at Intel Core i7 CPU, 2.8GHz with 8 cores and 4GB RAM.



(a) The real module. (b) The detailed model. (c) The model used in the simulator.

Figure 4.1

4.1 Simulation environment

For the purposes of this diploma thesis, a simple ad-hoc simulator of CoSMO modular robots was created. The physical environment is granted by the Open Dynamics Engine library (ODE) [36] that includes also a library DrawStuff for a simple visualization. The physical library can cope with collisions of simple geometric structures, allows connecting objects together with joints of different types and offers functions for applying forces and torques to objects.

The CoSMO module was modeled as a pair of L-shaped parts connected together with a rotation joint. A single module in the simulator is shown in Fig. 4.1c. This simplified model does not describe the real hardware as precisely as more detailed models (Fig. 4.1b), but on the other hand, it provides fast simulations, which is demanded for statistical benchmark testing. However, some of the found locomotion patterns were also run on the real robots with satisfying results, thereby it showed up that even a simple simulator may prepare a sufficient initial solution, that can be further improved in a more complex simulator or on a real hardware. A simple proportional controller was implemented to control the module to a desired position. The modules are loaded into the environment and connected together with fixed joints, as if they were assembled in the real world, according to their position and orientation in a grid written in a simple text file. The used data format allows creating two-dimensional shaped organisms.

4.2 Settings

4.2.1 Genetic Algorithm

As a usage of a Genetic Algorithm for automatic locomotion generation for modular robots was already investigated in [23], we used the parameters of the algorithm presented there. The population size was set to 150. The generation gap, which defines the number of individuals replaced after the natural selection operator, was set to 0.6 (the fittest 40 % of the population is kept, the rest is filled by the crossover). The m_rate parameter defining a probability of mutation in each dimension of a genome was set to 0.05. As the crossover operator, the Unimodal Normal Distribution Crossover was used (3.2.2.1). The variance values used for generation of random samples were set as recommended in [33]: $\sigma_\xi^2 = \frac{1}{4}$ and $\sigma_\eta^2 = \frac{0.35^2}{n}$, where n is the number of dimensions of the search space.

4.2.2 Particle Swarm Optimization

We used a basic PSO algorithm (3.2.1), where the weights of **pbest** and **gbest**, c_1 and c_2 are equal and set to 2 and the inertia weight ω is constant and set to 1. A swarm size recommended in [27] is between 10 and 50 particles and in [31] no efficiency difference is reported for swarm sizes between 20 and 100 particles. Based on this knowledge, the chosen number of particles in a swarm for our experiments was 20.

The used randomized topology of a swarm was described in 3.2.1. This topology can vary in two parameters, a probability p_c of adding another particle in current particle's neighborhood and t_{keep}, the number of iterations for which the actual topology is kept before it is re-structured. The settings of these parameters was $p_c = 0.25$ (each particle has approximately 25 % of the rest of the swarm in its neighborhood) and $t_{keep} = 10$ (every 10th iteration the neighborhoods are dismissed and re-generated).

4.2.3 Harmonic oscillators

The parameters of harmonic oscillators (3.1.1) are a frequency f_i, an amplitude A_i and a phase shift ϕ_i. All of them are gained by optimization, thereby we only need to specify their boundaries. The minimal values of these parameters are 0, maximal frequency and amplitude can be naturally obtained from mechanical limitations of a real hardware. The constraints used for the parameters of harmonic oscillators were: $f_i \in [0, 0.2]$, $A_i \in \left[0, \frac{\pi}{5}\right]$, $\phi_i \in [0, 2\pi]$.

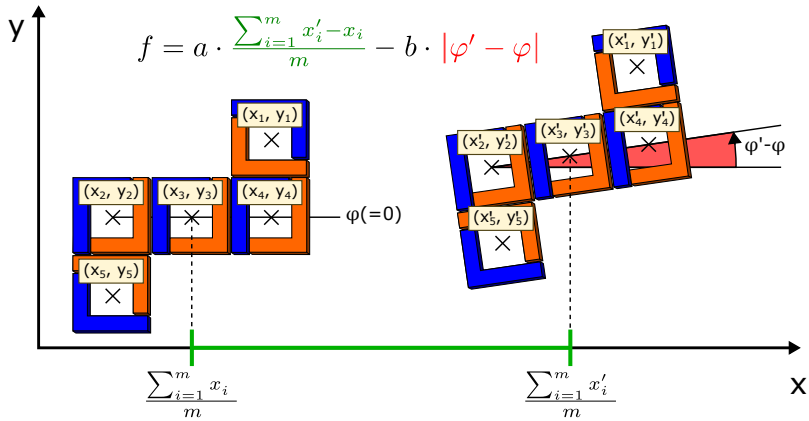


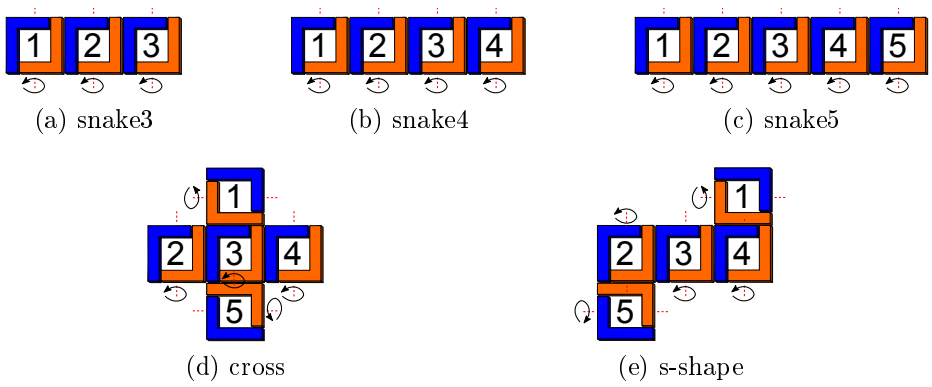
Figure 4.2: Used definition of the locomotion fitness value for a move forward along the x axis. The coordinates (x_i, y_i) stand for the position of the i-th module at the beginning of the simulation and (x_i', y_i') determine the position after a specified simulation time when applying the tested locomotion.

4.2.4 Coupled nonlinear oscillators

The CPG with coupled nonlinear oscillators was described in 3.1.2. Some of the parameters were fixed, these were two of initial conditions $r_i(0) = \dot{r}_i(0) = 0$ and a constant $a_i = 1$, and the rest was obtained from optimizers. The upper boundary of intrinsic frequency f_i was slightly lowered compared to harmonic oscillator, as there is another term added to the change of the phase. The initial phase $\theta_i(0)$ and the intrinsic amplitude R_i have the same meaning as the parameters ϕ_i and A_i in harmonic oscillators. The parameters w_{ij} and ϕ_{ij} determines weights and phase bias of influences between modules. In our experiments, the constraints of the parameters were set as: $f_i \in [0, 0.15]$, $R_i \in \left[0, \frac{\pi}{5}\right]$, $\theta_i(0) \in [0, 2\pi]$, $w_{ij} \in [-1, 1]$, $\phi_{ij} \in [0, 2\pi]$.

4.2.5 Fitness definition

To measure a quality of a set of parameters, a simulation is run for a specified amount of time and during the simulation the robot is driven by the CPG using the tested parameters. In this diploma thesis, locomotion patterns were optimized to transport the robot forward along the x axis (robots were oriented to head to this direction), while keeping the original heading. After the simulation is finished, the fitness value of a locomotion for a robot with m modules is computed as follows:



(a) snake3 (b) snake4 (c) snake5

(d) cross (e) s-shape

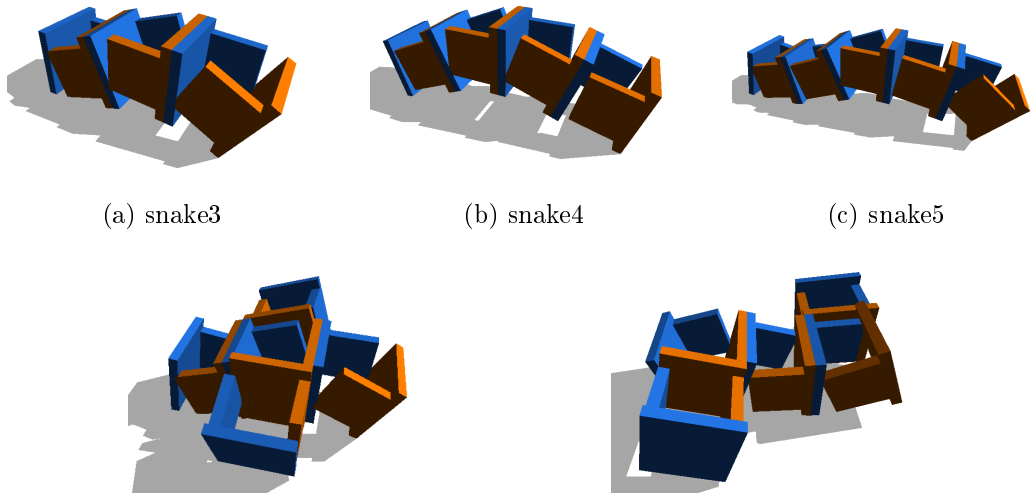
Figure 4.3: The organisms used for experiments with the numbering of modules and the orientation of rotation axes.

$$f = a \cdot \frac{\sum_{i=1}^{m} x_i' - x_i}{m} - b \cdot |\varphi' - \varphi| \tag{4.1}$$

where x_i is the x coordinate of i-th module at the beginning of the simulation, x_i' is its x coordinate after the simulation ends, φ is the original heading of the robot in radians (in our simulations it was always 0) and φ' is the heading at the end of the simulation. Due to the structure of used organism, the heading is common for all modules. The positive constants a and b determines weights of both terms in the fitness equation and were set to $a = 1$, $b = 2$. The Equation 4.1 implies that a locomotion is rewarded for a movement of the organism in the positive direction of the x axis and penalized for the change of the heading. The meaning of all terms in the fitness equation is depicted in Fig. 4.2. According to our definition of the fitness value, a higher value means a better solution. The fitness function can be easily extended to consider movements also in other directions.

4.2.6 Shapes of organisms

The investigated methods were experimentally evaluated by finding locomotion patterns for five shapes of modular robots. These were three snake-like robots with different length, *snake3* (3 modules), *snake4* (4 modules) and *snake5* (5 modules), a *cross* robot (5 modules) and an *s-shape* robot (5 modules). Schematics of all robot types used during experiments are pictured in Fig. 4.3 and their appearance in the simulator is shown in Fig. 4.4.



(a) snake3 (b) snake4 (c) snake5

(d) cross (e) s-shape

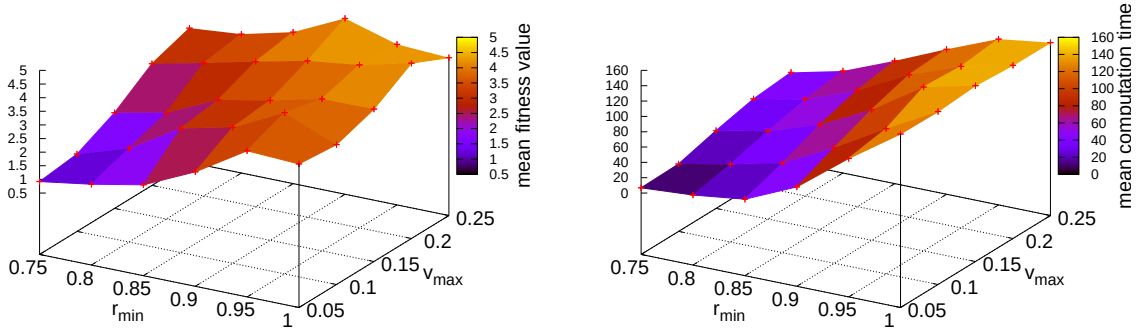
Figure 4.4: The appearance of used organisms in the simulation environment.

4.3 Results

4.3.1 Influence of v_{max} and r_{min}

Before running the benchmark tests of implemented methods, we examined the influence of the PSO parameter v_{max} (the maximal velocity of a particle) and the reliability threshold r_{min} of the Fitness Estimation method used for both PSO and GA.

If a particle moves faster, its consecutive positions are farther from each other and therefore the reliability of an estimated fitness decreases. As both v_{max} and r_{min} may affect the performance of PSO algorithm, we set up an experiment to determine the influence of their various combinations. The tested values were $v_{max} \in \{0.05, 0.1, 0.15, 0.2, 0.25\}$ and $r_{min} \in \{1, 0.95, 0.9, 0.85, 0.8, 0.75\}$. For every combination the PSO algorithm was run ten times with optimization being stopped after 30 iterations and the simulation time (for determining the fitness value) 30 simulated seconds. Measured values were the best fitness value found and the computation time. A character of results was similar for all robot types and both CPGs. Typical obtained data is visualized in Fig. 4.5. Please note that this experiment was processed before the Random Fitness Evaluation method (also described in 3.2.3) was implemented, which is the reason why a duration of the simulation drops so



(a) Mean fitness values.

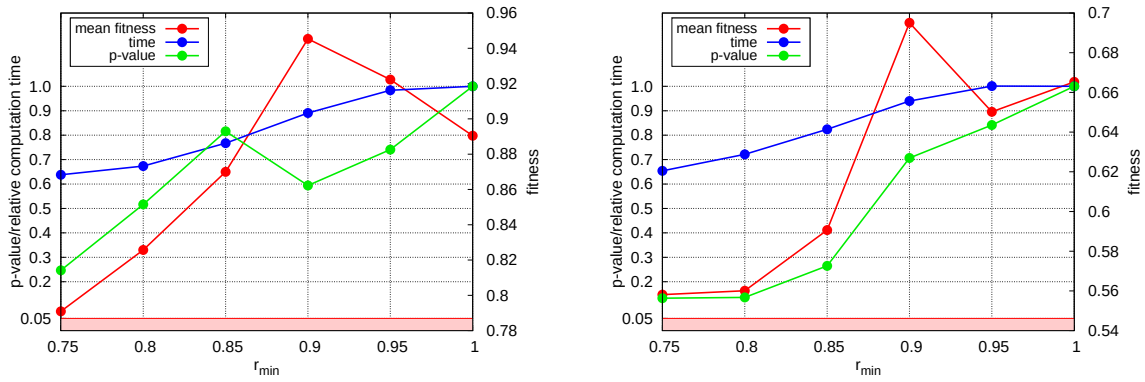
(b) Mean computation time in seconds.

Figure 4.5: Influence of the maximal particle speed v_{max} and the reliability threshold r_{min} in PSO and FE-PSO algorithms with a *snake3* robot.

significantly when r_{min} is low. Based on this experiment, we decided to fix the v_{max} parameter to a value 0.2 which seemed to work well in most cases (and was also consistent with a range stated in [37]) and further investigated an appropriate setting of r_{min} for PSO and GA.

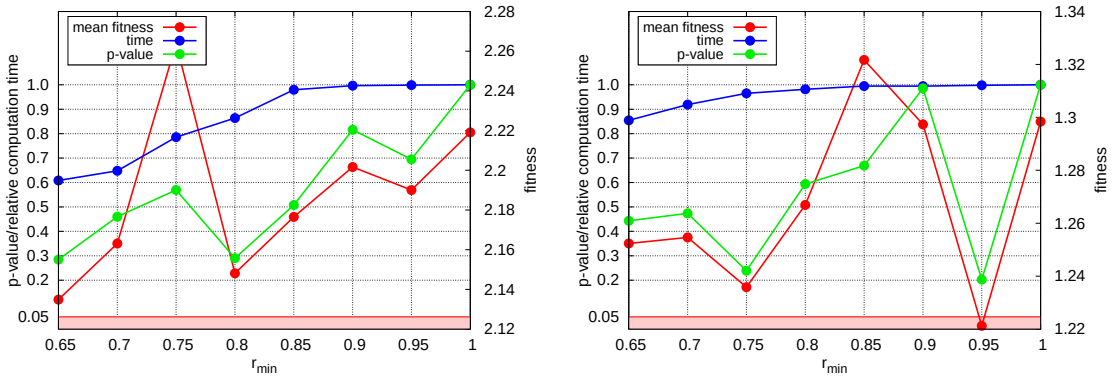
Another set of experiments was performed for various settings of r_{min} for FE-PSO and FE-GA, to determine a feasible value of this parameter for different robots. The desired setting lowers the computation time while providing statistically equal results as the algorithms without Fitness Estimation. For each tested r_{min}, the algorithms were terminated after 30 iterations, the fitness values were measured after 10 simulation seconds, every experiment was repeated 25 times. To compare the performances with various r_{min} to the performance with $r_{min} = 1$ (original PSO and GA), we used the t-test, assuming that the fitness values of found locomotion patterns follow the normal distribution. The null hypothesis is that the samples (fitness values) obtained with $r_{min} = 1$ and r_{min} being tested were generated under the same distribution. The p-value obtained from t-test gives a probability of generation of the observed samples if the null hypothesis was correct. If the p-value drops below the significance level (e.g. 5%), the null hypothesis is rejected. In other words, the higher the p-value is, the more likely the generators of two compared sets of samples are equal. Appropriate setting must lower the computation time and must be evaluated by sufficiently high p-value.

We plotted the data obtained from PSO and GA with all robots and both CPGs



(a) Sine CPG. (b) Nonlinear CPG.

Figure 4.6: Influence of the reliability threshold r_{min} in FE-PSO for an *s-shape* robot.



(a) Sine CPG. (b) Nonlinear CPG.

Figure 4.7: Influence of the reliability threshold r_{min} in FE-GA for a *snake3* robot.

	PSO		GA	
	Sine CPG	**Nonlinear CPG**	**Sine CPG**	**Nonlinear CPG**
snake3	0.8	0.9	0.75	0.65
snake4	0.8	0.8	0.65	0.65
snake5	0.8	0.9	0.7	0.7
cross	0.8	0.85	0.7	0.65
s-shape	0.85	0.9	0.65	0.65

Table 4.1: Selected values of the reliability threshold r_{min}.

and used the generated graphs for selection of the reliability threshold values for different cases. The graphs includes the mean values of best fitness values found, the p-values of the t-test comparing results of an algorithm without Fitness Estimation ($r_{min} = 1$) and with an given reliability threshold value, and relative mean values of computation time with respect to the mean value of time with $r_{min} = 1$.

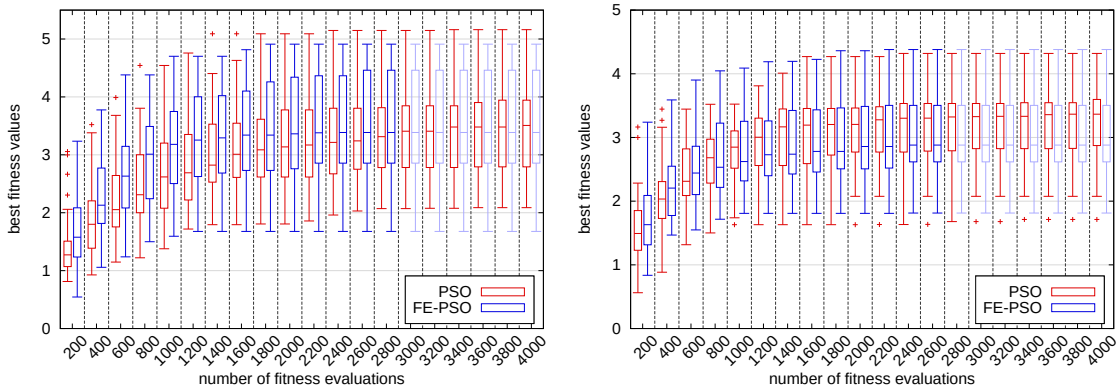
Typical results we got for the PSO algorithm are shown in Fig. 4.6. The first figure displays the data obtained for the *s-shape* robot when using Sine CPG. Due to these results, the setting $r_{min} = 0.8$ was used. For this value of the threshold, the p-value was about 50% and the computation time was more than 30% faster compared to PSO without Fitness Estimation. For the Nonlinear CPG, shown in the second figure, the p-value drops faster, thus a higher reliability threshold $r_{min} = 0.9$ with p-value around 70% and time saving about 10%.

The results for GA with the *snake3* robot are in Fig. 4.7. Generally, for GA the r_{min} had to be set lower compared to PSO to affect the computation time. In the presented case of the *snake3* robot the chosen values were $r_{min} = 0.75$ for Sine CPG (p-value $\approx$ 55%, time saving $\approx$ 20%) and $r_{min} = 0.65$ for Nonlinear CPG (p-value $\approx$ 45%, time saving $\approx$ 15%).

The selected values of r_{min} for all robot and CPG types are stated in Tab. 4.1.

4.3.2 Benchmark tests

To compare the performance of PSO and GA and the same methods enhanced by the Fitness Estimation method FE-PSO and FE-GA, statistical benchmark tests were processed. 50 measurements were taken for each of the four methods, using



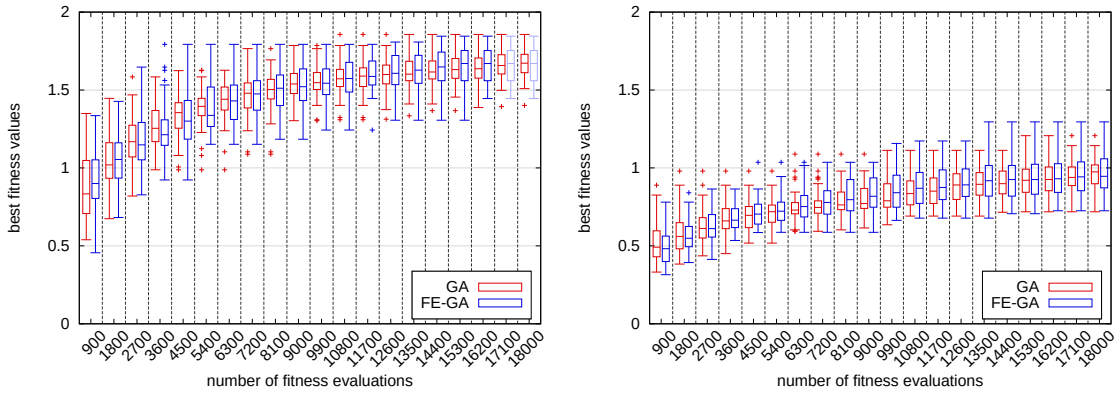
(a) A *snake5* robot with Sine CPG.

(b) A *snake4* robot with Sine CPG.

Figure 4.8: The comparison of PSO and FE-PSO.

every shape of a robot and both Sine and Nonlinear CPG. The algorithms were terminated after 200 iterations. During the optimization, the actual best fitness value and the number of true fitness evaluations were stored in each iteration. Thus, the experiments provided 50 vectors with 200 fitness values and 200 numbers of evaluations for each combination of an optimization method, a robot shape and a CPG type.

As the key parameters of optimization are the quality of generated solutions and the computation time, which in our case is dominated by the number of true fitness evaluations, the comparison of two methods was processed as follows. The interval from zero to the number of fitness evaluations (the higher of the two) was divided into several identical parts. For every dividing point, standing for the number of fitness evaluations, fitness values of both methods were determined, that needed this number of fitness evaluations to be found, and boxplots were generated from these fitness values. In other words, we generated a set of boxplots with the number of fitness evaluations on x axis and achieved fitness values on y axis. If one of compared algorithms terminates earlier, light blue boxes are used to display the final results of finished algorithm for easier comparison of both results. Using these graphs we can easily compare the performance of two methods with respect to required computation time.



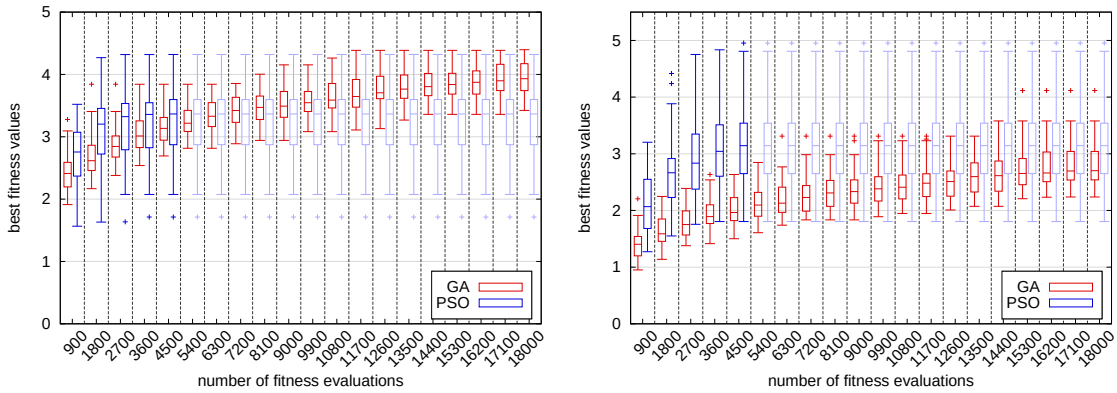
(a) An *s-shape* robot with Sine CPG. (b) An *s-shape* robot with Nonlinear CPG.

Figure 4.9: The comparison of GA and FE-GA.

4.3.2.1 Evaluation of the Fitness Estimation method

The results achieved in 4.3.1 were promising, as the Fitness Estimation method decreased the computation time down to 70% in some cases, while keeping a quality of found locomotions similar to original algorithms. The comparison method proposed above allows deeper investigation of Fitness Estimation performance, with impact on how much the reached fitness value depends on computationally expensive evaluation of the fitness function. The results differs for every shape of a robot and both CPGs. The FE-PSO algorithm performed best with organisms consisting of 5 modules - *snake5*, *cross* and *s-shape*, while with *snake3* and *snake4* the fitness curves were same the fitness curves of original algorithm or worse. Little effect was achieved when applying the Fitness Estimation method to GA, which corresponds with lower reliability of estimated fitness values in FE-GA (explained in 3.2.3). Probably the reliability thresholds were set too high for FE-GA, as the number of estimated fitness values was very low.

An example of a good performance of FE-PSO is shown in Fig. 4.8a, for a *snake5* robot and Sine CPG combination. The FE-PSO algorithm converges after 1600 fitness evaluations while the classic PSO algorithm needs 2800 fitness evaluations to reach the same results. In contrast, the FE-PSO algorithm failed for a *snake4* robot with Sine CPG. At the beginning FE-PSO provides similar or maybe slightly better results than PSO, but then it gets stuck at a local optimum in most cases and the median stays under the PSO median, as shown in Fig. 4.8b. The comparison of GA and FE-GA is demonstrated in Fig. 4.9, where the results are practically identical,



(a) Results for *snake4* and Sine CPG. (b) Results for *snake5* and Nonlinear CPG.

Figure 4.10: PSO to GA comparison. Typical graphs.

whether the Fitness Estimation method was used or not.

4.3.2.2 Comparison of PSO with GA

From the graphs obtained as described above, we can conclude some basic properties of PSO and GA and their differences. The PSO algorithm seems to reach a sufficient median values faster than GA in all cases except the Sine CPG with the *s-shape* robot. Also, the relative performance of GA gets worse if the Nonlinear CPG was used, which could imply that the PSO algorithm cope better with a raising dimensionality of the search space. On the other hand, it is clear that the GA provides more consistent results, as the variance of fitness values is much lower than the variance of PSO results. The final results of GA after 200 iterations of optimization are better compared to PSO for all robots if Sine CPG was used, but the number of fitness evaluations must be taken into account. After 200 iterations, the PSO algorithm had called the fitness evaluation 4000 times ($particle\_count \cdot iteration\_count = 20 \cdot 200 = 4000$), while the GA had called it 18000 times ($generation\_gap \cdot population\_size \cdot generation\_count = 90 * 200 = 18000$). The results of PSO would probably further improve if the algorithm went through more iterations, but as we did not processed the experiments for more than 200 iterations, we do not dispose of the required data to confirm this. Although PSO ran for less than a quarter of GA computation time, it found better parameters for Nonlinear CPG for all shapes of a robot but *s-shape*.

In most cases, one of the two typical results shown in 4.10 was obtained. For Sine CPG, the PSO algorithm usually performed better at the beginning but then gets

outperformed by GA with raising number of fitness evaluations. As an example, the results for a *snake4* robot were used in Fig. 4.10a. If we compared the quality of optimization by the achieved median, the PSO algorithm requires only 3600 fitness evaluations to provide equal results as GA after 6300 fitness evaluations, i.e., it reaches this results about 40% faster. On the other hand, with more iterations the results of GA further improves and the median raises. The other case is presented in Fig. 4.10b, showing the results for *snake5* and Nonlinear CPG. The GA algorithm does not achieve the PSO results even after four times more fitness evaluations.

Using the graphs in Fig. 4.10, we can observe the basic properties of distributions of results as well. The variance of the PSO results is much higher than the variance of the GA results. It implies that we should not rely on a single PSO run, as the optimized parameters may be far away from optimal values. On the contrary, the results of GA were more stable. All graphs comparing PSO and GA are placed in Appendix C.

4.3.3 Locomotion under failures

The failures in modular robotics were introduced in 2.1.2. We investigated a preventive solution based on generation of such a robust locomotion patterns, that would succeed in transporting the robot even if some hardware failures appear. We considered only two possible functional states of a module, the module is either working correctly or it is stuck in zero angle, and assumed that at most one or two modules can be broken at a time.

To allow the CPG to react if a behavior of a robot changes, we added a simple feedback to Nonlinear CPG, so the change of phase is given by:

$$\dot{\theta}_i = 2\pi f_i + \sum_{j \neq i} w_{ij} \left(\theta_j^{real} - \theta_i^{real} \right),$$

where θ_i^{real} is the real position of the i-th robot's actuator. Thus, in this section, we used three CPG types for locomotion generation, Sine CPG, Nonlinear CPG and Nonlinear CPG with feedback (FB-Nonlinear CPG).

The investigated method of generation failure-resistant locomotion patterns is based on learning the CPG parameters on a robot under failures. The simulation is processed multiple times with different combinations of failures and the overall fitness value is computed as a sum of fitness values from all simulations. After the optimization finishes, the fitness values of the best parameters found are computed under all combinations of errors and stored. The PSO algorithm terminating after 100 iterations was used for optimization. Each optimization was processed 25 times.

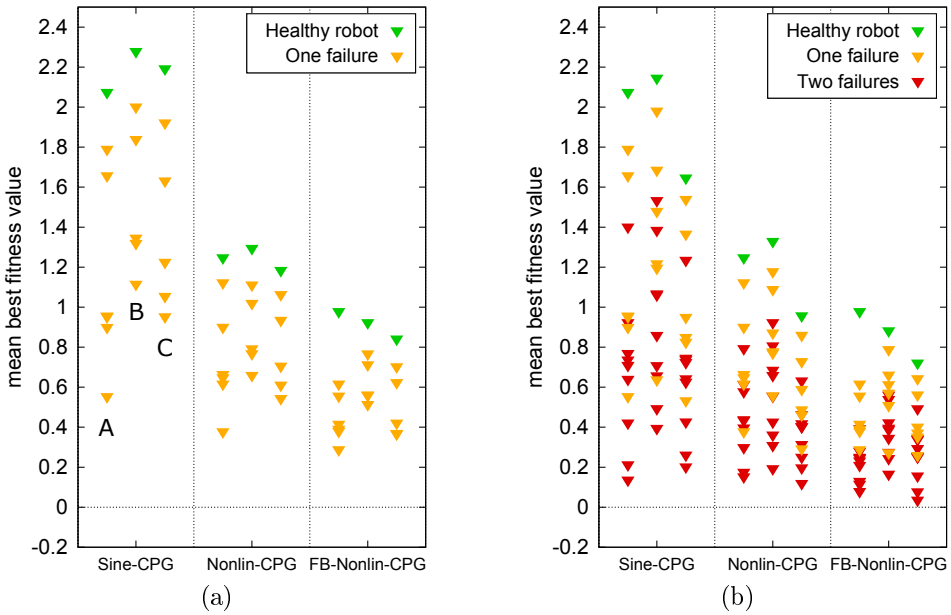


Figure 4.11: Locomotion of a *snake5* robot under failures.

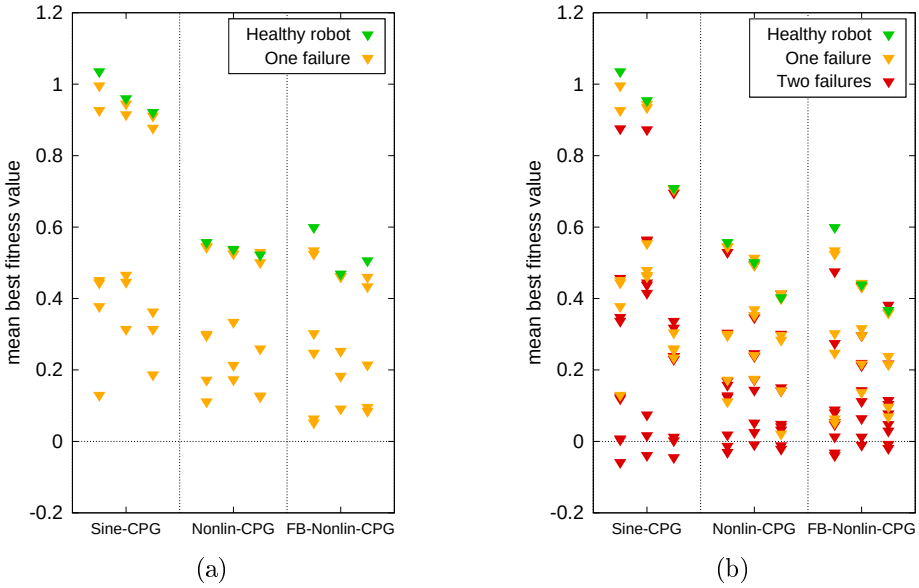
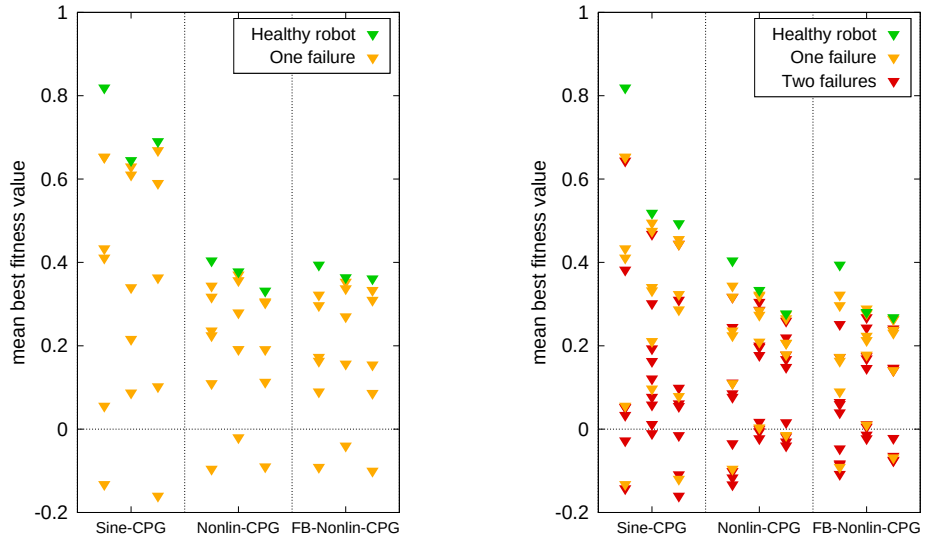


Figure 4.12: Locomotion of a *cross* robot under failures.



(a) (b)

Figure 4.13: Locomotion of an *s-shape* robot under failures.

The presented graphs show the performance of every CPG method trained on either a healthy robot, on all combinations of failures or on a subset of combinations of states of modules, with or without failures, randomly selected in every iteration of the optimization. 2 combinations were selected in the case of one allowed failure and 5 combinations in the case of two allowed failures.

For every CPG, there are three columns of points determining the achieved mean fitness values. The first column contains fitness values of locomotions optimized on a healthy robot, the second column contains results of an optimization considering all combinations of failures and the third column contains fitness values of locomotions optimized on a randomly selected combinations of states of modules. Each point represents results for a specific state of the robot. Green points stands for fitness values achieved by a healthy robot, each yellow point represents a failure of one of modules and red points denotes combinations of two broken modules.

If we do not consider the absolute values, all CPGs changed their performance similarly when the three optimization approaches were applied, thereby our experiments did not confirmed, that an appropriate choice of CPG type can improve a behavior of locomotions under failures, nor the introduction of a simple feedback in the manner as stated above. In the contrary, the obtained results apparently differ

if the failures are taken into account during the optimization. As the Sine CPG provided highest fitness values, we will use its result for the comparison of improved optimization approaches.

For a *snake5* robot, as shown in Fig. 4.11a, fitness values with broken modules (and even the fitness value of a healthy robot) increased when enhanced optimization was used. In the first column, a fitness value of the worst-case scenario of a failure lies between 0.5 and 0.6 (A). The optimization that considers all possible failures make this value double (B) and even the optimization that considers only two randomly selected states increases the worst fitness value to about 0.9 (C). If two modules can break (Fig. 4.11b), only the optimization considering all possible states (a middle column of each CPG) affects the robustness of found locomotion patterns. Similar results can be observed also with *cross* (Fig. 4.12) and *s-shape* (Fig. 4.13) robots.

The processed experiments allow us to better understand the importance of different modules of a robot and to determine a sensitivity of given robot shapes to failures, as well. For example, in Fig. 4.12b, there are three red points with fitness value near to zero in every column. These points represents three possible combinations that include two of the modules lying on the motion axis of the *cross* robot (denoted as 2, 3 and 4 in Fig. 4.12). If two of these modules break, there is no way the robot can move in desired direction, no matter how well the optimization is implemented.

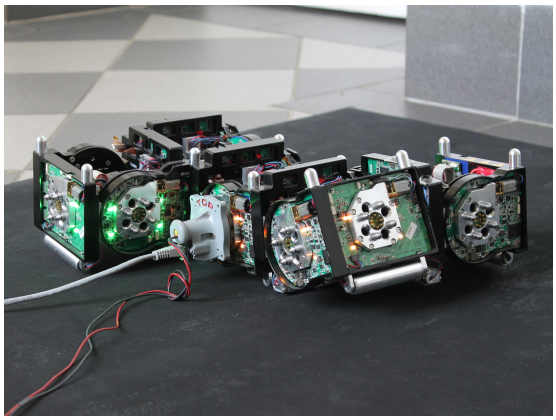


Figure 4.14: An experiment with the *s-shape* robot.

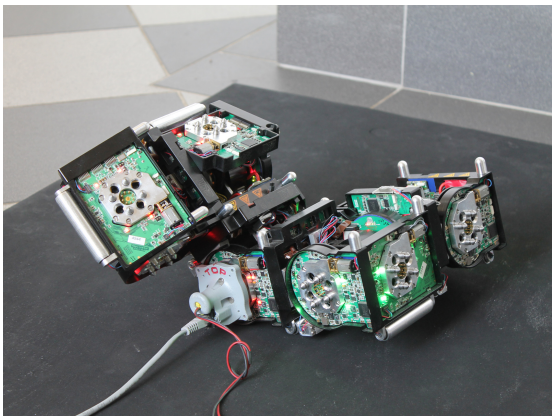


Figure 4.15: A spasm of one of the modules.

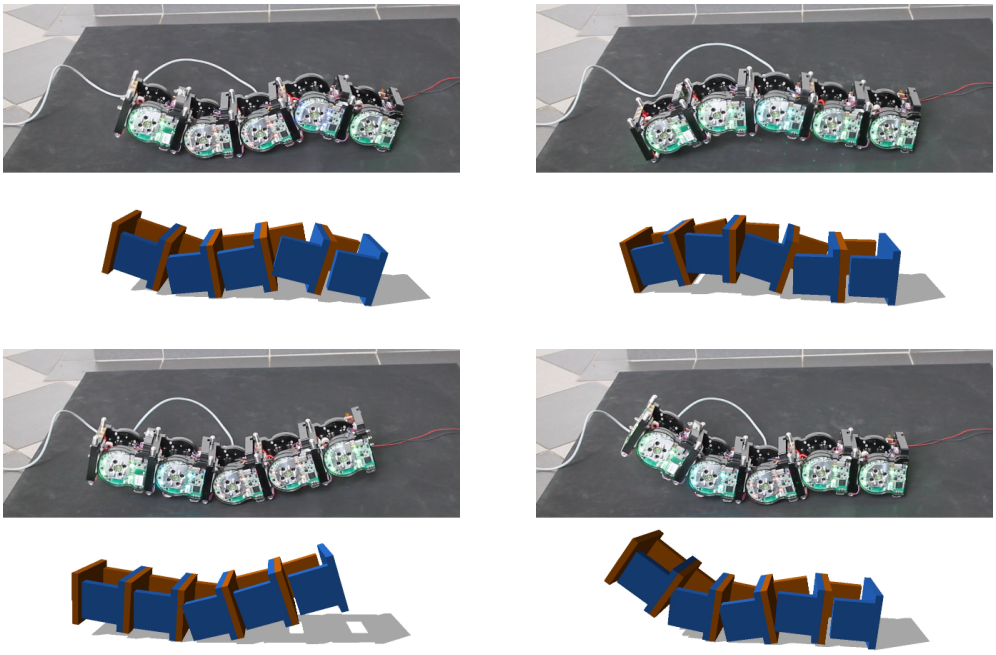


Figure 4.16: Comparison of an experiment with real *snake5* robot and a simulated experiment using the same Sine CPG parameters.

4.3.4 Experiments with real robots

To verify the simulator and the feasibility of investigated methods of automatic locomotion pattern generation, we used the Sine CPG with parameters optimized in the simulator to generate control inputs for a real robot. We tested a forward locomotion of *snake5* and *s-shape* robots and a rotational motion of a *s-shape* robot. Although the simulator operates only with a very rough model of a real module (Fig. 4.1c), it provided locomotion patterns that could be instantly used with a real hardware. Despite technical difficulties with some of the modules (e.g. Fig. 4.15) we had at our disposal, we managed to run these locomotions and thus to apply the simulated results practically.

The best results were obtained for a *snake5* robot, as demonstrated in Fig. 4.16. The same Sine CPG parameters were applied to the real robot and to the robot in the simulator, and locomotions of both were captured and synchronized. Another experiments were run for an *s-shape* robot (Fig. 4.14). For this shape, the differences between simulations and reality were bigger, nevertheless, some of the selected CPG

locomotions performed well. Together with forward locomotions, we tested also rotational motion, as the *s-shape* structure allows it.

Chapter 5

Conclusion

In this diploma thesis, methods of automatic locomotion generation for modular robots were investigated. The locomotion was achieved by the joint-control approach, where the movement is emerged by a precise cooperation of actuators of all modules. The Central Pattern Generation method was used for generation of the control inputs for each module. CPG is based on a set of coupled oscillators, that can be implemented variously. Two different types of oscillators were presented in this thesis, simple harmonic oscillators and coupled nonlinear oscillators. Both of them are determined by parameters, that have to be set properly to achieve a locomotion of an robotic organism.

The optimization of the parameters is being slowed down by a very expensive fitness function, as it has to computed by actually applying the parameters and observing the resulting movement. Therefore, only fast converging optimization algorithms are appropriate for this task. In this diploma thesis, two bio-inspired methods were investigated, a Genetic Algorithm and a Particle Swarm Optimization. To further decrease the number of needed fitness evaluations, the Fitness Estimation method was applied to both of them.

An important task of modular robotics is coping with unexpected errors in modules. A preventive method for overcoming failures of modules by generation of robust locomotion patterns, was designed and evaluated. Also, the CPG with coupled nonlinear oscillators was altered by adding a simple feedback into the oscillator, to examine if it could provide better reactions to changes of a behavior of broken modules.

For the purposes of this diploma thesis, a simple simulator for Symbrion CoSMO modules was implemented. Using the simulator, all investigated methods could be experimentally evaluated. Some of locomotion patterns obtained from the simulator

were afterward successfully run on a real hardware, validating a sufficient quality of the simulator and implemented methods. Moreover, by running the experiments with real robots, all the guidelines from the diploma thesis assignment were fulfilled including the optional part.

5.1 Future work

Based on results obtained in this diploma thesis, some further investigation may be suggested. Both optimization methods are characterized by different properties. PSO is capable of fast search for a feasible solution and cope well with a high dimensional search space, while GA provides more stable results, but converges slowly. Using swarms of particles obtained from several short PSO runs as an initial population of GA could connect advantages of both methods together. The Fitness Estimation method could be applied to PSO to further increase the performance, as it was shown to be capable of speeding up the convergence of PSO in some cases.

The preventive method of optimization with simulated broken modules was verified to improve the locomotion robustness in case failures occurs. The problem is the number of possible combinations of errors that have to be taken into account, increasing rapidly the computation time of a fitness evaluation. We propose a method based on determining, which combinations of failures cause the worst degradation of the locomotion fitness and optimizing the locomotion with respect to these worst-case combinations only.

Bibliography

[1] Mark Yim, Wei-Min Shen, Behnam Salemi, Daniela Rus, Mark Moll, Hod Lipson, Eric Klavins, and Gregory S Chirikjian. Modular self-reconfigurable robot systems [grand challenges of robotics]. *Robotics & Automation Magazine, IEEE*, 14(1):43–52, 2007.

[2] E. Yoshida, S. Murata, H. Kurokawa, K. Tomita, and S. Kokaji. A distributed reconfiguration method for 3d homogeneous structure. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 2, pages 852–859, 1998.

[3] K.C. Prevas, C. Unsal, M.O. Efe, and P.K. Khosla. A hierarchical motion planning strategy for a uniform self-reconfigurable modular robotic system. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2002.

[4] Daniela Rus and Marsette Vona. Crystalline robots: Self-reconfiguration with compressible unit modules. *Autonomous Robots*, 10(1):107–124, 2001.

[5] Robert Fitch and Zack Butler. Million module march: Scalable locomotion for large self-reconfiguring robots. *International Journal of Robotic Research*, 27(3–4):331–343, 2008.

[6] K. Tomita, S. Murata, H. Kurokawa, E. Yoshida, and S. Kokaji. Self-assembly and self-repair method for a distributed mechanical system. *IEEE Transactions on Robotics and Automation*, 15(6):1035–1045, 1999.

[7] Paul Levi and Serge Kernbach. *Symbiotic multi-robot organisms: reliability, adaptability, evolution*, volume 7. Springer Science & Business Media, 2010.

[8] V. Vonasek, S. Neumann, D. Oertel, and H. Woern. Online motion planning for failure recovery of modular robotic systems. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[9] Tom Larkworthy and Subramanian Ramamoorthy. An efficient algorithm for self-reconfiguration planning in a modular robot. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 5139–5146. IEEE, 2010.

[10] A. Pamecha, I. Ebert-Uphoff, and G.S. Chirikjian. Useful metrics for modular robot motion planning. *IEEE Transaction on Robotics and Automation*, 13(4):531–545, 1997.

[11] R. Primerano, D. Wilkie, and W.C. Regli. A case study in system-level physics-based simulation of a biomimetic robot. *IEEE Transactions on Automation Science and Engineering*, 8(3):664 –671, 2011.

[12] Ivan Tanev, Thomas Ray, and Andrzej Buller. Automated evolutionary design, robustness, and adaptation of sidewinding locomotion of a simulated snake-like robot. *IEEE Transactions on Robotics*, 21(4):632–645, 2005.

[13] Mikhail Prokopenko, Vadim Gerasimov, and Ivan Tanev. Evolving spatiotemporal coordination in a modular robotic system. In *From Animals to Animats 9*, pages 558–569. Springer, 2006.

[14] Vojtech Vonàsek, Karel Kosnar, and Libor Preucil. Motion planning of self-reconfigurable modular robots using rapidly exploring random trees. In *Advances in Autonomous Robotics - Joint Proceedings of the 13th Annual TAROS Conference and the 15th Annual FIRA RoboWorld Congress, Bristol, UK, August 20-23, 2012*, pages 279–290, 2012.

[15] E. Yoshida, H. Kurokawa, A. Kamimura, K. Tomita, S. Kokaji, and S. Murata. Planning behaviors of a modular robot: an approach applying a randomized planner to coherent structure. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 2, pages 2056 – 2061, 2004.

[16] V. Vonasek, M. Saska, K. Kosnar, and L. Preucil. Global Motion Planning for Modular Robots with Local Motion Primitives. In *IEEE International Conference on Robotics and Automation*, 2013.

[17] Vojtech Vonasek, Martin Saska, Lutz Winkler, and Libor Preucil. High-level motion planning for cpg-driven modular robots. *Robotics and Autonomous Systems*, 68(0):116 – 128, 2015.

[18] Eve Marder and Dirk Bucher. Central pattern generators and the control of rhythmic movements. *Current biology*, 11(23):R986–R996, 2001.

[19] Scott L Hooper. Central pattern generators. *Current Biology*, 10(5):R176–R179, 2000.

[20] Auke Jan Ijspeert. Central pattern generators for locomotion control in animals and robots: a review. *Neural Networks*, 21(4):642–653, 2008.

[21] Jörg Conradt and Paulina Varshavskaya. Distributed central pattern generator control for a serpentine robot. In *International Conference on Artificial Neural Networks (ICANN 2003)*, 2003.

[22] Auke Jan Ijspeert, Alessandro Crespi, Dimitri Ryczko, and Jean-Marie Cabelguen. From swimming to walking with a salamander robot driven by a spinal cord model. *science*, 315(5817):1416–1420, 2007.

[23] Akiya Kamimura, Haruhisa Kurokawa, E Toshida, Kohji Tomita, Satoshi Murata, and Shigeru Kokaji. Automatic locomotion pattern generation for modular robots. In *IEEE International Conference on Robotics and Automation*, pages 714–720, 2003.

[24] Thomas Bäck and Hans-Paul Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary computation*, 1(1):1–23, 1993.

[25] Randy L Haupt and Sue Ellen Haupt. *Practical genetic algorithms*. John Wiley & Sons, 2004.

[26] Duncan E. Jackson and Francis L.W. Ratnieks. Communication in ants. *Current Biology*, 16(15):R570 – R574, 2006.

[27] James Kennedy, James F Kennedy, and Russell C Eberhart. *Swarm intelligence*. Morgan Kaufmann, 2001.

[28] J Kennedy and R Eberhart. Particle swarm optimization. In *IEEE International Conference on Neural Networks*, pages 1942–1948, 1995.

[29] R Eberhart and J Kennedy. A new optimizer using particle swarm theory. In *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pages 39–43, 1995.

[30] Qinghai Bai. Analysis of particle swarm optimization algorithm. *Computer and information science*, 3(1):p180, 2010.

[31] Daniel Bratton and James Kennedy. Defining a standard for particle swarm optimization. In *IEEE Swarm Intelligence Symposium*, pages 120–127, 2007.

[32] Arvind S Mohais, Rui Mendes, Christopher Ward, and Christian Posthoff. Neighborhood re-structuring in particle swarm optimization. In *AI 2005: Advances in Artificial Intelligence*, pages 776–785. Springer, 2005.

[33] Hajime Kita, Isao Ono, and Shigenobu Kobayashi. Theoretical analysis of the unimodal normal distribution crossover for real-coded genetic algorithms. In *The IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence.*, pages 529–534, 1998.

[34] Mehrdad Salami and Tim Hendtlass. A fast evaluation strategy for evolutionary algorithms. *Applied Soft Computing*, 2(3):156–173, 2003.

[35] ZH Cui, JC Zeng, and GJ Sun. A fast particle swarm optimization. *International Journal of Innovative Computing, Information and Control*, 2(6):1365–1380, 2006.

[36] Russell Smith. Open dynamics engine, 2008.

[37] Russell C Eberhart and Yuhui Shi. Particle swarm optimization: developments, applications and resources. In *IEEE Congress on Evolutionary Computation*, pages 81–86, 2001.

Appendix A

DVD content

./dt.pdf	This diploma thesis.
./video	Contains video files captured during experiments.
./graphs	Contains graphs that were not included in the thesis.
./code	Contains C++ source files of implemented programs.

Appendix B

Using the Gram-Schmidt process for obtaining vectors $\mathbf{e}_1 \dots \mathbf{e}_{n-1}$

The vectors $\mathbf{e}_1, \dots, \mathbf{e}_{n-1}$ are perpendicular to the vector $\mathbf{d}$ and they span the secondary search space. The secondary search space and the direction of the vector $\mathbf{e}$ in a two-dimensional search space is visualized in 3.1. For an n-dimensional search space the number of these vectors is $n-1$, as these vectors span the subspace of the search space perpendicular to the primary search direction. To obtain the vectors $\mathbf{e}_1, \dots, \mathbf{e}_{n-1}$, the Gram-Schmidt orthonormalizing process may be used. The method takes a finite set of linearly independent vectors $\mathbf{v}_1, \dots, \mathbf{v}_n$ and generates an orthonormal set of vectors $\mathbf{e}_1 \dots \mathbf{e}_n$ spanning the same space as the original set. The orientation of the generated orthonormal set is given by the first vector provided. The Gram-Schmidt process works as follows:

$$\begin{aligned}
&\mathbf{u}_1 = \mathbf{v}_1, && \mathbf{e}_1 = \tfrac{\mathbf{u}_1}{\|\mathbf{u}_1\|} \\
&\mathbf{u}_2 = \mathbf{v}_2 - \tfrac{\mathbf{v_2}\cdot\mathbf{u}_1}{\mathbf{u}_1\cdot\mathbf{u}_1}\mathbf{u}_1, && \mathbf{e}_2 = \tfrac{\mathbf{u}_2}{\|\mathbf{u}_2\|} \\
&\mathbf{u}_3 = \mathbf{v}_3 - \tfrac{\mathbf{v_3}\cdot\mathbf{u}_1}{\mathbf{u}_1\cdot\mathbf{u}_1}\mathbf{u}_1 - \tfrac{\mathbf{v_3}\cdot\mathbf{u}_2}{\mathbf{u}_2\cdot\mathbf{u}_2}\mathbf{u}_2, && \mathbf{e}_3 = \tfrac{\mathbf{u}_3}{\|\mathbf{u}_3\|} \\
&\vdots && \vdots \\
&\mathbf{u}_n = \mathbf{v}_n - \textstyle\sum_{i=1}^{n-1} \tfrac{\mathbf{v_n}\cdot\mathbf{u}_i}{\mathbf{u}_i\cdot\mathbf{u}_i}\mathbf{u}_i && \mathbf{e}_n = \tfrac{\mathbf{u}_n}{\|\mathbf{u}_n\|}
\end{aligned} \tag{B.1}$$

The vectors $\mathbf{u}_1, \dots, \mathbf{u}_{n-1}$ form the orthogonal set and the vectors $\mathbf{e}_1, \dots, \mathbf{e}_{n-1}$ form the orthonormal set. In every step of the algorithm, the projections of the input vector $\mathbf{v}_i$ into existing orthogonalized vectors $\mathbf{u}_i$ are subtracted from vector $\mathbf{v}_i$ and then the resulting difference vector is normalized. If the set $\mathbf{v}_1, \dots, \mathbf{v}_n$ was linearly dependent, in some step the vector $\mathbf{u}_i$ would be a zero vector, which means that the

provided vector $\mathbf{v}_i$ can be already generated as a linear combination of the vectors $\mathbf{u}_1 \dots, \mathbf{u}_{i-1}$.

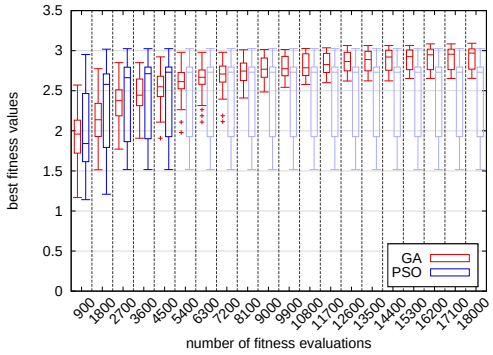
To generate a set of vectors perpendicular to the vector $\mathbf{d}$ in the n-dimensional space, we can use a standard n-dimensional basis $\mathbf{b}_1 = \begin{pmatrix} 1 & 0 & \dots & 0 \end{pmatrix}, \mathbf{b}_2 = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \end{pmatrix}, \dots, \mathbf{b}_n = \begin{pmatrix} 0 & \dots & 0 & 1 \end{pmatrix}$ extended by the vector $\mathbf{d}$ as the input set of the algorithm. In case the vector $\mathbf{d}$ is a multiple of any of the standard basis vectors, the required set $\mathbf{e}_1, \dots, \mathbf{e}_{n-1}$ is composed by the standard basis vectors without the one parallel with $\mathbf{d}$. Else, the basis $\mathbf{e}_1, \dots, \mathbf{e}_{n-1}$ can be computed through the Gram-Schmidt process:

$$\begin{aligned}
&\mathbf{u}_0 = \mathbf{d}, && \mathbf{e}_0 = \tfrac{\mathbf{u}_0}{\|\mathbf{u}_0\|} \\
&\mathbf{u}_1 = \mathbf{b}_1 - \tfrac{\mathbf{b_1} \cdot \mathbf{u}_1}{\mathbf{u}_1 \cdot \mathbf{u}_1} \mathbf{u}_1, && \mathbf{e}_1 = \tfrac{\mathbf{u}_1}{\|\mathbf{u}_1\|} \\
&\mathbf{u}_2 = \mathbf{b}_2 - \tfrac{\mathbf{b_2} \cdot \mathbf{u}_1}{\mathbf{u}_1 \cdot \mathbf{u}_1} \mathbf{u}_1 - \tfrac{\mathbf{v_3} \cdot \mathbf{u}_2}{\mathbf{u}_2 \cdot \mathbf{u}_2} \mathbf{u}_2, && \mathbf{e}_2 = \tfrac{\mathbf{u}_2}{\|\mathbf{u}_2\|} \\
&\vdots && \vdots \\
&\mathbf{u}_{n-1} = \mathbf{b}_{n-1} - \textstyle\sum_{i=1}^{n-1} \tfrac{\mathbf{b_{n-1}} \cdot \mathbf{u}_i}{\mathbf{u}_i \cdot \mathbf{u}_i} \mathbf{u}_i && \mathbf{e}_{n-1} = \tfrac{\mathbf{u}_{n-1}}{\|\mathbf{u}_{n-1}\|}
\end{aligned} \tag{B.2}$$

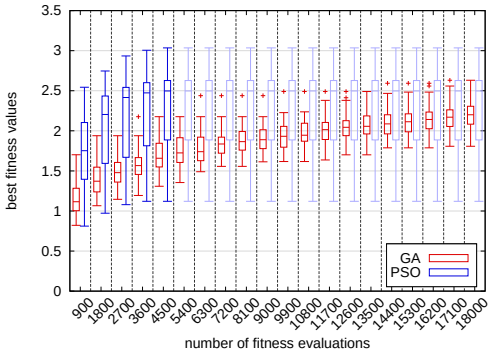
where the indexing was modified so the vector $\mathbf{e}_0$ is parallel to $\mathbf{d}$ and the vectors $\mathbf{e}_1, \dots, \mathbf{e}_{n-1}$ are the required set of vectors perpendicular to $\mathbf{d}$.

Appendix C

Graphs for comparison of PSO with GA

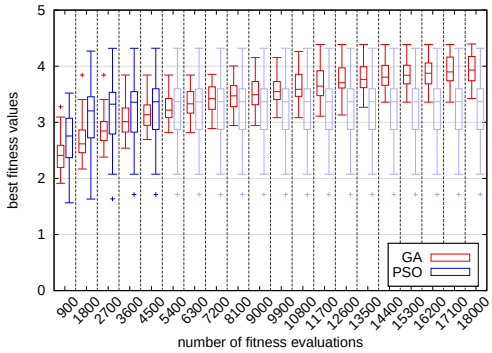


(a) Sine CPG.

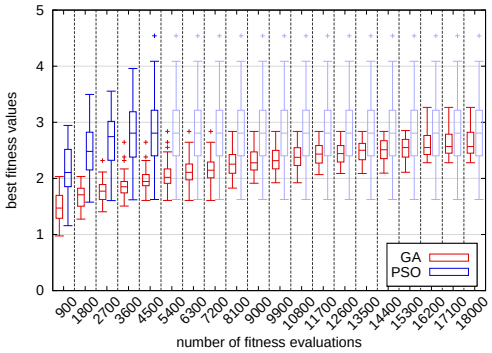


(b) Nonlinear CPG.

Figure C.1: *snake3*

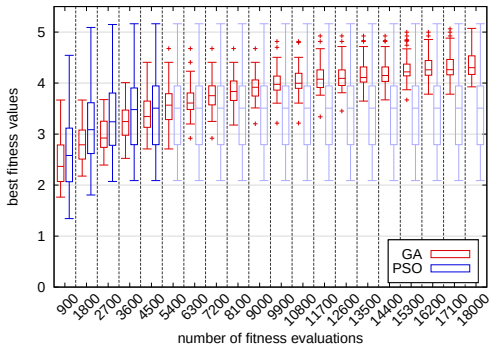


(a) Sine CPG.

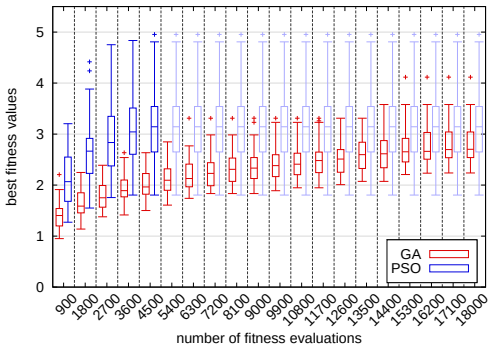


(b) Nonlinear CPG.

Figure C.2: *snake4*

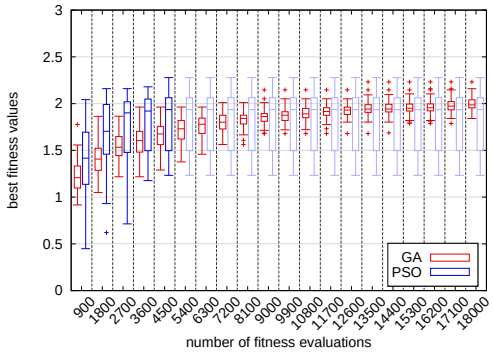


(a) Sine CPG.

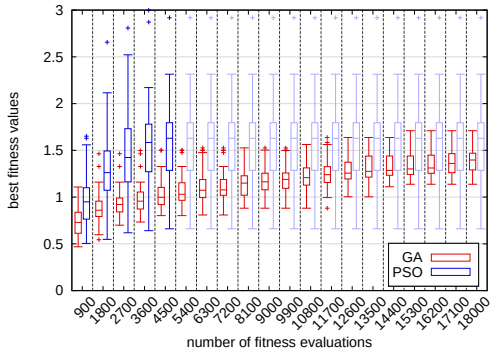


(b) Nonlinear CPG.

Figure C.3: *snake5*

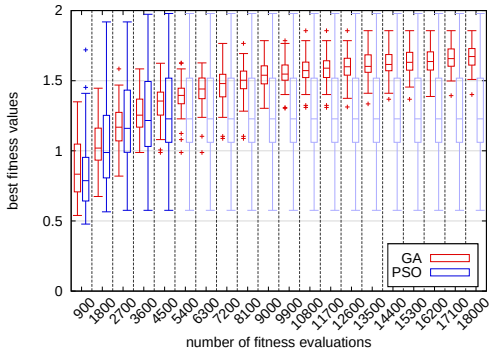


(a) Sine CPG.

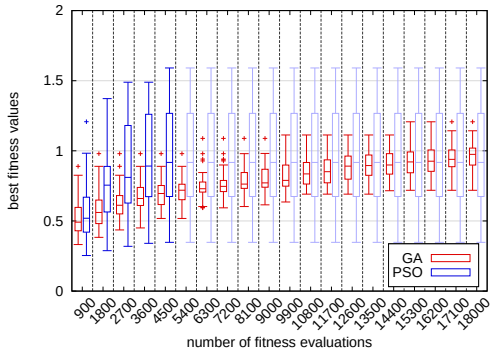


(b) Nonlinear CPG.

Figure C.4: *cross*



(a) Sine CPG.



(b) Nonlinear CPG.

Figure C.5: *s-shape*