

Roombots: Reconfigurable Robots for Adaptive Furniture



© BRAND X PICTURES & DIGITAL VISION

Abstract—Imagine a world in which our furniture moves around like legged robots, interacts with us, and changes shape and function during the day according to our needs. This is the long term vision we have in the *Roombots* project. To work towards this dream, we are developing modular robotic modules that have rotational degrees of freedom for locomotion as well as active connection mechanisms for runtime reconfiguration. A piece of furniture, e.g. a stool, will thus be composed of several modules that activate their rotational joints together to implement locomotor gaits, and will be able to change shape, e.g. transforming into a chair, by sequences of attachments and detachments of modules.

In this article, we firstly present the project and the hardware we are currently developing. We explore how reconfiguration from a configuration A to a configuration B can be

controlled in a distributed fashion. This is done using *meta-modules*—two Roombots modules connected serially—that use broadcast signals and connections to a structured ground to collectively build desired structures without the need of a centralized planner.

We then present how locomotion controllers can be implemented in a distributed system of coupled oscillators—one per degree of freedom—similarly to the concept of central pattern generators (CPGs) found in the spinal cord of vertebrate animals. The CPGs are based on coupled phase oscillators to ensure synchronized behavior and have different output filters to allow switching between oscillations and rotations. A stochastic optimization algorithm is used to explore optimal CPG configurations for different simulated Roombots structures.

I. Introduction

Our goal is to merge technologies from information technology, roomware, and robotics to design adaptive and intelligent furniture. We intend to design and control modular robots, called Roombots (RB), to be used as building blocks for furniture that moves, self-assembles, self-reconfigures, and self-repairs. Modular robots are robots made of multiple simple robotic modules that can attach and detach. Connectors between units allow the creation of arbitrary and changing structures depending on the task to be solved, therefore offering versatility and robustness against failure, as well as the possibility of self-reconfiguration.

The type of scenario that we envision is a group of RB units that autonomously connect to each other to form different types of furniture, e.g. stools, chairs, sofas and tables, depending on user requirements. This furniture will change shape over time, e.g. a stool becoming a chair, a set of chairs becoming a sofa. Roombots units will move to different locations depending on the users' needs. Additionally the Roombots will be capable of memorizing user preferences in terms of structures and places in order to facilitate repetitive use of the adaptive furniture.

Here we focus on two major, initial tasks of the RB modules. Firstly we investigate force-field guided reconfiguration movements of *RB metamodules* into furniture-like structures. Embedded connectors in our structured environment enable the RB metamodules to grab into and use them as pivot points for caterpillar-like movement sequences. Secondly we apply *central pattern generators* (CPG) for controlling the locomotion of multiple-unit Roombots robotic structures. This allows RB units to move independently from a structured environment. A CPG network produces robust, synchronized patterns for oscillatory and

rotational joint movements, with a minimum number of control parameters, and is well-suited for an optimization algorithm. CPG networks are by definition decentralized and hence very well suited for controlling modular robots. Derived locomotion patterns are very well-performing, and versatile for all tested RB structures.

We have organized the paper as follows. In Section II we look at applications and properties of self-reconfiguring modular robots, reconfiguration strategies for modular robots, and locomotion controllers, and we place our hardware and strategies within each of them. In Section III we describe the Roombots module concept and the currently existing hardware. Section IV shows setup and simulation results for the reconfiguration strategy we use with Roombots metamodules. Section V describes the applied CPG model, the CPG network and the optimization framework we apply to different Roombots robot structures. Section VI concludes our reconfiguration and locomotion strategies for the Roombots platform, and gives an outlook for future work.

II. Related Work

With the Roombots project we wish to extend but also test a future scenario, where technology is being merged into everyday environment, ranging from tables to walls, from furniture like shelves to electric installations, e.g. autonomously moving shades [1]. This new field named *roomware* [2] searches to design and evaluate computer-augmented room elements with integrated information and communication technology. The idea of using technology with touchable, shape or surface changing interfaces and functionalities is increasingly discussed in the field of *tangible interactions* [3]. Most of the work in the field of roomware is done on fixed topologies; here we aim towards a scenario where the user creates his or her own shapes of furniture. Ultimately adaptive Roombots furniture will be able to

*Alexander Spröwitz, Soha Pouya,
Stéphane Bonardi, Jesse van den Kieboom,
Rico Möckel, Aude Billard, Pierre Dillenbourg,
and Auke Jan Ijspeert*
École Polytechnique Fédérale de Lausanne (EPFL),
SWITZERLAND

transform and merge from one shape, e.g. two chairs, into another, e.g. a table.

We use the concept and the ideas of *self-reconfiguration modular robots* (SRMR) or *Dynamically Reconfigurable Robotic Systems* [4] as physical building blocks for our adaptive furniture. The field of self-reconfiguring modular robots, which are modular robot units that can actively attach and detach themselves with each other and the environment, is a robotic concept which was firstly implemented with CEBOT (“cell structured robot”) by Fukuda et. al [4] in the late 1980s. CEBOT already included all the main properties of modular robotic systems. (i) A robot is made from individual “cells” or modular units. The task is then performed by a collective assembly of those modules. (ii) Each cell is mostly autonomous, i.e. equipped with processing power, actuation, battery power, communication, sensors and has an own hardware frame. (iii) The shape of the assembly is task-dependent, as for certain tasks the number of degrees of freedom (DOFs) or their orientation matters. The modular robot community has been growing ever since, we count about 50 different modular robotic systems up to now. Using self-reconfiguring modular robots has advantages, as opposed to *monolithic* robots such as humanoid or quadruped robots. Depending on the capabilities of a single modular unit, large numbers of shapes can be created by remote control [5]. This is especially helpful if the task is initially unknown. If a quadruped-shaped modular robot locates a hole in the wall it can shape-change into a caterpillar-like structure, and go through. As many different robotic shapes can be created with the same set of units, transport is easy and less costly, e.g. to remote locations. Units are interchangeable such that modular robotic cells can be replaced in case of failure, what potentially makes these systems robust. However these advantages come with a price. Implementing autonomy in modular robots, equipping each of the units with a connection mechanism, actuators, and electronics makes them heavy, expensive, and hard to design. A robotic configuration built from modular robots will normally perform less well compared to a monolithic robot as the abilities and dynamics of a monolithic robot can be optimized—it serves a smaller number of dedicated, pre-known tasks.

The usefulness of a modular or monolithic approach therefore depends on the application. Research in modular robots often aims towards applications at disaster sites, remote or hazardous environments that are inaccessible to human operators, where their shape changing characteristics and robustness are crucial. A number of modular robot projects are working at micro-scale modular robots, i.e. they aim for rapid prototyping-like technologies [6]. For the Roombots project we chose self-reconfiguring modular robots for their abilities in building arbitrary, adaptive structures.

Finding and applying an automated controller to change shape is one of the main topics in reconfigurable robotics. Centralized strategies often use a graph-based approach, describing the combined modular robot structure using graph theory, where actions are represented by insertion and

deletion of edges and vertices [7]. Connector actions and joint rotations are the result of an optimization process attempting to morph the graph representing the initial structure, into the goal configuration. This allows for a very precise reconfiguration process, however graph methods do not scale well with increasing numbers of joints, connectors and modules. Common approaches for decentralized reconfiguration are “cluster flow” [8] locomotion or “water flow-like locomotion algorithms” [9] and describe locomotion by self-reconfiguration (or vice versa), or “dynamic reconfiguration” [9]. They facilitate large amounts of, usually abstracted modular units moving or changing shape through the environment, where units are simulated as cubes or spheres which slide along planes and around edges, or rotate around edges (“sliding cube”) [10]. Movements of single units can be guided by a global gradient [11] or triggered by hormone-like messages [12]. Cellular automata [13] oriented methods use local rules. Those can be learned by distributed, reinforcement learning algorithms to optimize the behavior of single units task dependently. Varshavskaya et. al [14] present such learning algorithms assuming only partial world-knowledge. Fitch et. al [15] demonstrate highly scalable systems with many modular units based on the “MeltSortGrow”-algorithm. They later extend their algorithm such that it also works in tight spaces [16]. Using a simplified modular robot unit presentation, like the “sliding cube” model, is helpful to derive a reconfiguration strategy on an abstracted level. To implement the strategy on a low-level, i.e. on an actual modular robotic system, the notion of *metamodules* is often formulated. Metamodules are clustered assemblies of modular robot units which are combined for the purpose of moving just as their sliding-model counterpart cubes, however by using the actual degrees of freedom available from the hardware units. Butler and colleagues [9, cf. page 7] mention the usefulness of such metamodules (Molecule’s *tile* [17] and Atom’s *grain* [18]). Dewey and colleagues [19] cluster the entire modular robot assembly in equal, non-dense generalized metamodules, which enables them to apply a very simple planner for module movement through the structure.

In addition to reconfiguration, RB robots can move using whole body motions, e.g. like a walking quadruped structure or a metamodule rolling on the ground. Hence no structured environment with connectors is needed. To control these types of locomotion, we use a dynamical systems approach inspired from the biological central pattern generators (CPGs), i.e. neural circuits capable of producing coordinated patterns of high-dimensional rhythmic output signals while receiving only simple, low-dimensional input signals [20]. The goal is to produce oscillations as the limit cycle behavior in a system of coupled nonlinear oscillators. Compared to other approaches used in modular robotics such as gait tables (Yim [21], Bongard et al. [22]) or sine-based controllers (Stoy et al. [23]), this approach benefits from many interesting properties including decentralized control, synchronization between multiple oscillators and robustness against perturbations. In

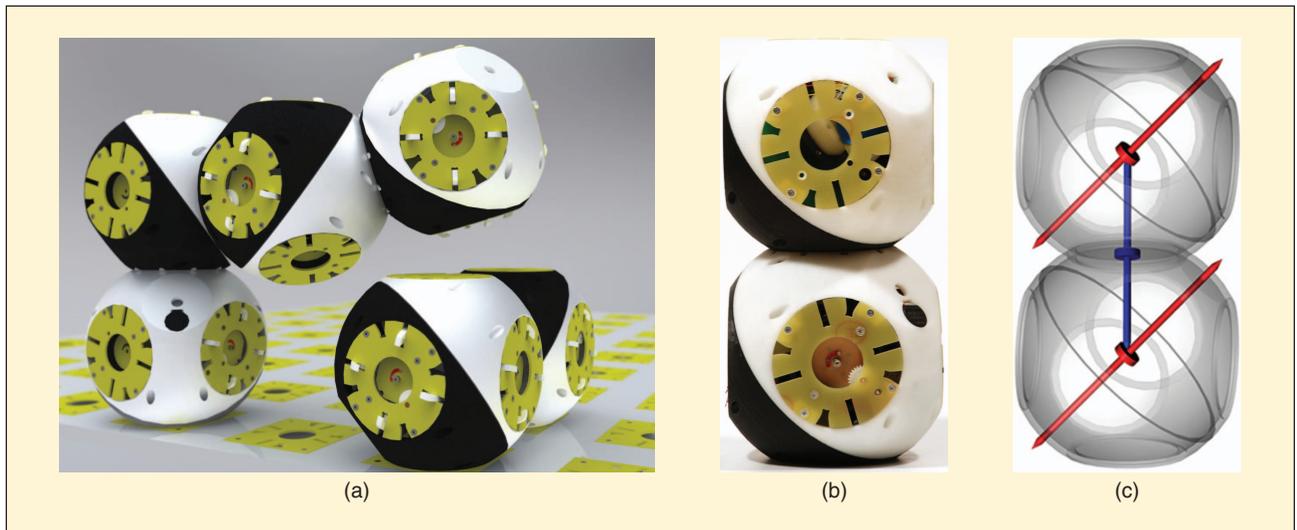


FIGURE 1 (a) Rendered visualization of one Roombots *metamodule* on the left and a *single* Roombots module on the right. Rectangular connector plates (yellow/green) are embedded in the floor. (b) Roombots module (real picture). (c) Three DOF per Roombots module: red axes are outer DOF, the blue DOF is rotating the two sphere-like parts of a Roombots module against each other. The ability to freely swivel the two outer joints against each other distinguishes a single Roombots module from plugging two Molecube [28] modules together. This loosely follows the concept of adding a center joint in Superbot [30], compared to M-TRAN II [31].

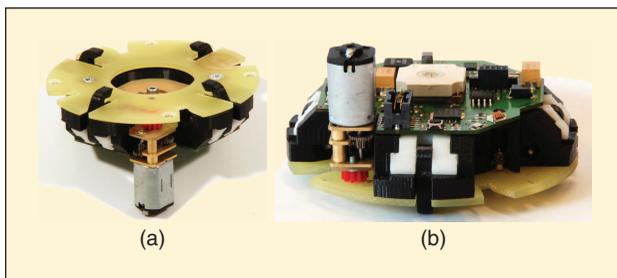


FIGURE 2 Active connection mechanism (ACM) of the Roombots. Four mechanical latching fingers grab synchronously into the neighbouring module or the structured surface. The mechanism is actuated with a mini-DC motor, with the position of the grippers sensed with a potentiometer (Fig. 2(b) at the center). The ACMs are designed to be mechanically autonomous and any other type of connector could be plugged into the corresponding Roombots sockets.

particular, CPGs allow much more freedom in modulating gaits than sine-based controllers since changes in the control parameters lead to smooth changes in the produced oscillations. CPG-based control of modular robots has been used firstly by Kamimura et al. ([24], [25]), who use two-neuron Matsuoka oscillators as a CPG model for MTRAN. In Kamimura et al. ([25]) the authors extend the CPG with a drift detection mechanism and demonstrate adaptive locomotion with M-TRAN in the face of external perturbations and varying environmental conditions. In previous work, we implemented various CPG models together with optimization algorithms [26], [27]. Here we present a CPG implementation that allows the generation of both rotational (i.e. with joint angles that monotonically increase) and oscillatory (i.e. with joint angles that go periodically back and forth around a rest position) movements. As will be shown later, this new architecture can fully benefit from all the locomotion possibilities of the Roombots modules.

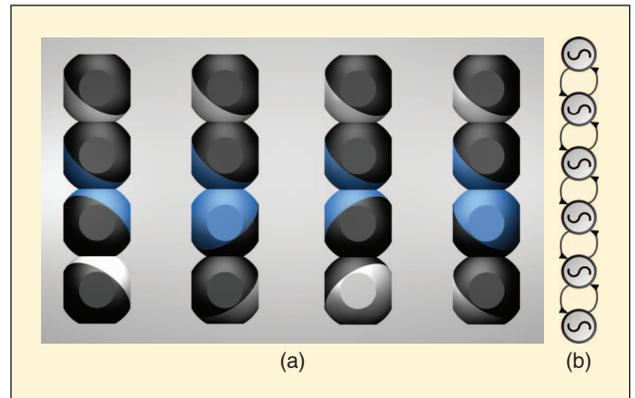


FIGURE 3 Roombots metamodules—each made of two Roombots modules—showing the four possible metamodule configurations. All three DOF within a Roombots module can take any arbitrary value of a full circle. However there are four distinct possibilities to connect two modules into a single metamodule, as the Roombots connectors [Fig. 2(a)] have a four-sided symmetry. The hemispheres connecting to the neighbouring RB module are colored blue. We use the relative orientation of the center axis of those hemispheres for naming: (from left to right) *shear-S SRS*, *shear-Z SRZ*, *perpendicular PER*, and *parallel PAR*. The orientation of each upper Roombots module is kept fixed. (b) depicts the CPG topology of a metamodule, see Section V. Each of the six oscillators is assigned to one DOF/joint of the metamodule. All oscillators have nearest-neighbor coupling.

III. Hardware Concept of Modules

Roombots (RB) are similar in their degrees of freedom (DOFs) to the *3D Molecubes* [28], and have inherited some of their main movement characteristics. An RB module features three DOF (Molecubes feature one DOF), and we combine two RB modules serially into one RB metamodule (Fig. 1a). We want to build furniture-shaped structures from metamodules, whereas a single metamodule (Fig. 3a) is 44 cm (17.3 in) long. Hence we can settle with medium-large number of modules for our

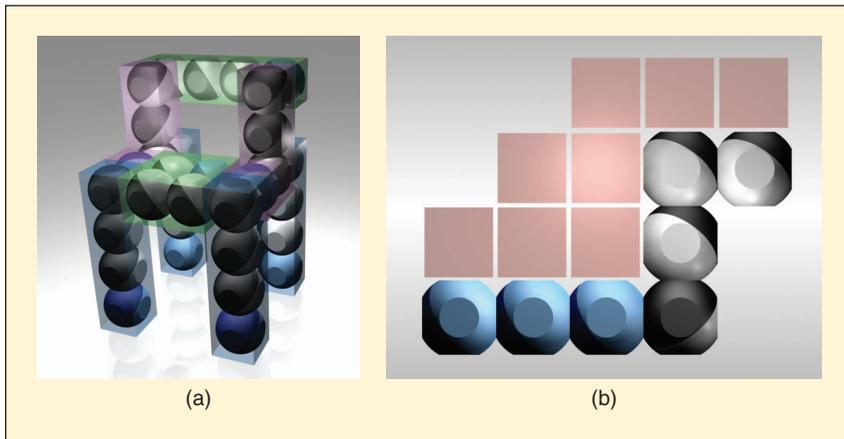


FIGURE 4 (a) A chair-like goal structure built from eight metamodules. Blueish hemispheres present the seeding points in the 3D regular grid. Metamodules are indicated with transparent blocks of different colors and the structure is assembled from I-shaped, U-shaped and L-shaped metamodules. Seeds are being iteratively activated, as soon as a previous seeding position is filled. (b) *Shape-transition* of a metamodule, from I-shape configuration (bluish, horizontal) to L-shape. Red boxes indicate the *collision cloud* a metamodule transition is producing, where every touched cube in the 3D grid is being recorded. Roombots movements are in 3D, this figure shows only a frontal projection of the cloud.

reconfiguration planner, e.g. around 20–50 RB modules would be sufficient. Also we can make use of connectors embedded in the environment (e.g. the yellow-green rectangular plates in Fig. 1a) and broadcast communication. Consequently we are able to omit some of the hard constraints such as constant connectivity between all modules, and local communication. We are still interested in a distributed system with low demands on communication bandwidth. A strong constraint of our Roombots system is the movement space of an RB metamodule—six DOF connected serially are very powerful, in terms of being able to overcome concave or convex obstacle edges. However an RB metamodule requires a rather large space around itself, and is not necessarily connected to another module or metamodule when moving, which needs to be considered in advance of the movement. Our current approach to automatic and distributed RB reconfiguration is limited to the planning level, assuming well-adjusted hardware. The somewhat “classical” problems of self-reconfiguring modular robot hardware, which are related to e.g. stiffness of the connectors and reliability of the docking procedure will be addressed on a different level, but not within the scope of this paper.

Similar to other modular robotic systems RB modules [29] are fitted into a regular cubic grid. We are using a grid size with 110 mm edge length. We connect two RB modules serially into an *RB metamodule* (Fig. 1a), four combinations are possible (PAR, PER, SRS and SRZ, see Fig. 3), and each resulting metamodule has its own range of motion and movement characteristic. Any of the three joints (Fig. 1c) of an RB module delivers sufficient torque to rotate a metamodule in the “worst case scenario situation”, i.e. out of a horizontal stretched position. RB modules are fabricated mostly from 3D printed ABS plastic pieces and plate-elements are milled out of glass-fibre sheet material. An RB module weights about 1.4 kg, that includes battery power for an estimated

30 min of continuous actuation, and the weight for electronic boards¹ Joints are equipped with high gear ratio gearboxes (about 360:1), actuated by strong DC motors which results in 5 Nm and 7 Nm torques for middle and outer joints, respectively. Any of the three joints is continuously rotational, i.e. can turn without mechanical stop. Electrical power and communication are transmitted with slip rings within the module. The two outer DOFs of a Roombots module (Fig. 1c, red) are of the same type as in the Molecube modules [28], [32]. Roombots modules have an additional actuated swivel joint (Fig. 1c, blue) in-between. The high torque demands and the resulting high gearbox ratio values limit Roombots’ maximum rotational speed. The center joint needs 3 sec to rotate 360° and both outer joints

roughly 2 sec. RB’s active connection mechanism (ACM) is genderless, four-way symmetric, with four mechanical latching fingers (Fig. 2a) which are completely retractable inside the body. ACMs fit into any of ten dedicated sockets of an RB module. In many ways the connector design is similar to the AMAS connection mechanism [33], although we use a different trajectory for the movement of the latching fingers [34]. We are in the process of finishing the Roombots hardware. Hence all the experiments in this article are implemented in Webots [35], a physics-based simulation environment.

IV. Distributed Reconfiguration

One of the visions of the Roombots project is to design adaptive furniture for home or office use. We plan to use metamodules which will need to configure into different pieces of furniture, several times during the day. This section describes our initial, currently simulated, approach to reconfiguration by locomotion on a structured surface, i.e. in a 3D environment with embedded connectors to which modules or metamodules can attach². Four different metamodule types are the moving units, their movements are guided towards the next active seeding position by a virtual force field. Metamodules send and receive broadcasts among each other to gather knowledge of their nearest neighborhood. A set of shape-transitions and corresponding collision-clouds (Fig. 4b) stored in a look-up table enables each metamodule to largely avoid collision, with itself, other meta-modules and the environment. We finish the section with initial results characterizing Roombots metamodules for this type of reconfiguration by locomotion.

¹The electronic hardware for Roombots is under development.

²It is likely that only a small area would need to be fitted with connectors for recharging and for locomotion. The rest of the living room could be accessed with “normal” locomotion.

A. Strategy

We will explain the distributed reconfiguration mechanism with Roombots metamodules on the example of building a chair-like structure, e.g. Fig. 4a.

a) Metamodule initialization

A typical initial configuration that we would envision in a house or an office scenario, is to have all modules forming a wall. RB metamodules will be placed in our structured environment (see the passive connectors in Fig. 1a). A metamodule starts by being attached to a connector with its foot hemisphere. It then determines its initial position and orientation (on the real modules this will be done by local communication with the connector or reading out a tag on the connector's surface). Roombots also have the ability to sense their own shape, by reading out internal joint angle sensor values.

b) Seeding recipe and metamodule shapes

The metamodule now receives information about its environment, e.g. obstacles or walls, but most importantly the *seeding recipe* of the goal structure. The seeding recipe is the “blueprint” for the structure which will be assembled from all the metamodules around, e.g. a chair-like structure (Fig. 4a). It will be provided by a human operator. The recipe includes the position and the order of the *seeding cubes*, which are attachment points for a metamodule within the goal structure. Metamodules are not assigned to a specific seeding cube, but the first arriving metamodule will fill the active position, and send a broadcast indicating the seeding cube is taken. Remaining metamodules will switch and go towards the next seeding cube in the seeding recipe. The recipe also includes the information of what type of metamodule-shapes the structure will be built from (indicated by semi-transparent, colored boxes combining paired Roombots modules in Fig. 4a). Metamodules can take five possible shapes: *I*, *L*, *S*, *U* and *3D-S*. Fig. 4b shows an *I*-shaped metamodule being rotated into an *L*-shaped metamodule.

c) Messages and locomotion

Metamodules use shape-to-shape transition for a caterpillar-like walking in 3D. Before a shape-transition, a metamodule sends a broadcast status message which contains its foot position and its ID. The broadcast messaging is meant as a replacement for close-range sensing of other metamodules, and serves to avoid colliding with them. This requires the knowledge of absolute coordinate points for all metamodules and the goal shape, which is possible in our semi-large environment. A module can derive its neighborhood from those status messages by comparing the senders position against its own. It will store this information for one step, and only for modules in close range.

d) Force-field guidance

The metamodule now knows its own absolute position $\vec{D}_{\text{foot}} = [D_x, D_y, D_z]$ in the 3D grid, the position of k number

of current seeds \vec{D}_{seed_i} and the positions \vec{D}_{meta_j} of n number of neighboring metamodules in range. It calculates a force vector \vec{V}_f by summing up the distance vector from the active seeds (attracting “sinks”). Depending on the strategy, neighboring metamodules are included in this calculation. They represent “sources” and emit a repelling force field, with a negative sign. At last the metamodule reaches for the next closest connector in the direction of \vec{V}_f . Once the metamodule head is connected to its new position, the module unlocks the foot, sends a new status message and repeats the cycle.

$$\vec{V}_f = \sum_{i=1}^k \frac{\vec{D}_{\text{foot}} - \vec{D}_{\text{seed}_i}}{|\vec{D}_{\text{foot}} - \vec{D}_{\text{seed}_i}|} - \sum_{j=1}^n \alpha(\vec{D}_{\text{foot}}, \vec{D}_{\text{meta}_j}) \times \frac{\vec{D}_{\text{foot}} - \vec{D}_{\text{meta}_j}}{|\vec{D}_{\text{foot}} - \vec{D}_{\text{meta}_j}|} \quad (1)$$

- a) $\alpha(\vec{D}_{\text{foot}}, \vec{D}_{\text{meta}_j}) = 0$ (α – greedy)
- b) $\alpha(\vec{D}_{\text{foot}}, \vec{D}_{\text{meta}_j}) = \frac{1}{4}(|\vec{D}_{\text{foot}} - \vec{D}_{\text{meta}_j}| - 4)$ (α – slope)
- c) $\alpha(\vec{D}_{\text{foot}}, \vec{D}_{\text{meta}_j}) = 1$ (α – step). (2)

e) Force vector strategies

We are interested in different strategies concerning the influence of neighboring metamodules on the \vec{V}_f calculation, and have designed three modes which are switched with the α function: (i) The α -greedy approach ($\alpha = 0$), where neighboring modules have no influence on the force field of other metamodules. During reconfiguration metamodules should go as straight as possible towards the next active seeding position. To minimize collisions, modules pause their step as soon as they detect (via status messages) another metamodule in a very close range, i.e. within four cubes distance. The lock is released with the next status message. (ii) An α -slope-function, where $\alpha = 1/4(|\vec{D}_{\text{foot}} - \vec{D}_{\text{meta}_j}| - 4)$. This gradually decreases the repelling force between the distance of four and eight cubes. (iii) An α -step-function, where $\alpha = 1$. Any metamodule within the distance of eight cubes provides a full force component. The hypothesis guiding this experiment is that with an additional, repelling force component metamodules will have a tendency to keep a minimum distance between each other. Hence less collisions should occur.

f) Look-up table and collision-cloud computation

As we do not apply sensing in the conventional sense, there is the danger of collision within a metamodule, between metamodules, or with an external object. We have designed a method that calculates in advance what we call a *collision cloud* (Fig. 4b) of a single metamodule for all permutations of initial and final metamodule shapes.³ The collision cloud represents

³There are five possible metamodule shapes, and four different metamodule configurations. Each can be assembled with different joint values. Three positions are possible for each of the four outer RB DOF in a metamodule, and four positions for the two inner DOF.

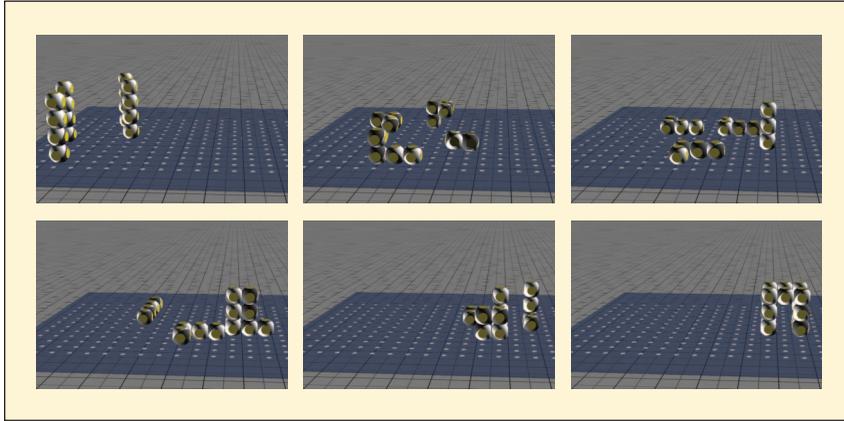


FIGURE 5 Snapshots series of four SRZ Roombots metamodules reconfiguring into a cube-like structure (from left to right, and from top to bottom). The applied force field strategy is α -slope-based. Metamodules start at a straight posture (left side). They attach and detach at passive connectors embedded in the ground (yellow-green tiles, Fig. 1a), and use them as pivot points for a caterpillar-like motion. Once a metamodule reaches a goal point within the cube-like-structure, it switches off.

the number and position of the virtual cubes being touched during the transformation, and is stored in a lookup table in an external device. At the beginning of each step the metamodule will request the collision cloud corresponding to its initial and final shape from the look-up table. It then checks, based on the cubic grid, if the cloud intersects with any known object or metamodule in range. The look-up table enables us to centrally store data which would be hard to compute in real-time for a single module, and is repeatedly requested from many metamodules.

B. Results and Discussion

We performed experiments in simulation in which four metamodules walk approximately 20 steps Manhattan-distance and have to assemble into a cube-like structure (Fig. 5)⁴. This set of experiments tested the four different metamodule types (PAR, PER, SRS, and SRZ, Fig. 3b), the three different reconfiguration strategies (α -greedy, α -slope, and α -step), with three different random initial conditions ($4 \times 3 \times 3 = 36$ experiments). In 26 of 36 experiments the final configuration was reached and the shape was created, and in all cases the area around the final configuration was reached. Hence we assume that the seeding order, or the way seed positions are taken, will play a large role in the future. However the presented initial experiments were aiming at characterizing the abilities and properties of the metamodules, and the influence of the force-field guidance. Concerning the latter, Table I indicates that PAR-type metamodules get stuck more easily within the last sequence of the reconfiguration, whereas the SRZ-metamodules only collided in two experiments with another metamodule, and never got stuck. Dead-locks are an issue due to the seeding recipe that is currently designed by hand. This seeding is not a trivial task to solve due to the rather complex movement characteristics of the Roombots

DOFs. We plan to automate and optimize the seeding in the future. Concerning the three tested force-field strategies: the greedy algorithm performed better than expected in terms of average number of necessary moves, a collision occurred only once within the valid experiments. On the other hand there are a large number of dead-locks (four out of twelve) with this strategy. It should be noted that for every of the nine experiment types at least one valid assembly, without collision, was achieved among all four metamodule-types. The unique design of Roombots metamodules (due to the high number of DOF per module) is such that, with sufficient space around, on-line switching from one type to

another (e.g. from PAR to SRZ) will be possible with a relatively small, intermediate reconfiguration sequence. Fig. 6 shows that the *force field correction* does affect the trajectories of metamodules, and metamodules tend to spread compared to the α -greedy strategy. However this does not seem to have a positive effect on the self-organization of collision-free reconfiguration moves, i.e. actually more collisions occur. Collisions can happen in this otherwise deterministic setup due to the *asynchronous steps* of RB metamodules, as they are not synchronized. In detail: a metamodule sends a status message, checks its environment, finds it unoccupied and starts to move. If another close-by metamodule starts moving with a delay, it assumes neighborhood knowledge on an outdated basis, and resumes movement in the shared space of another metamodule. There are at least two solutions available: (i) one could increase the safety distance between metamodules, e.g. to ten cubes. It is physically impossible for two moving metamodules to meet within one step, assuming that both move with about the same speed. However this requires large distances between metamodules, and is a very unattractive approach. (ii) Another option could be *consensus-based decision making* between metamodules, to agree on one's priority. This could require a global clock, i.e. synchronized cycles of movements as described in [36]. Dead-locks at the assembly phase of a structure happen as a result of the (currently hand-coded and) non-optimized seeding order, and the orientation of the foot hemispheres of metamodules within the assembled structure. Latter orientation strongly influences in which way the final shape of a metamodule is reached, i.e. how the metamodule is "folding" itself into that posture. We are planning on automating the seeding recipe by taking into account both constraints. From our initial experiment, we conclude that the SRZ metamodule together with the greedy reconfiguration strategy appear to be the most promising method for distributed reconfiguration. Additional tests are under way with more initial and final configurations to confirm this.

⁴Complementary reconfiguration videos are available at the Roombots webpage: <http://biorob.epfl.ch/page38279.html>.

TABLE 1 Table indicating the resulting number of average moves per experiment, collisions, and dead-lock situations for the cube-experiment (see Fig. 5). Table columns indicate three different strategies: greedy, based on a slope- α and based on a step function for the force vector estimation. Table rows show the four different metamodule configurations. Nine experiments (exp1–exp9) per metamodule type are implemented, the first column of each experiment indicates the average number of moves for four moving metamodules. Three experiments per configuration are shown, with the initial position of the metamodules shifted by a small number of steps. \checkmark in the second column indicates a successful assembly of the cube structure, * shows that no solution was found (dead-lock). In case of collisions between metamodules, but a successful assembly the number of collisions is indicated instead in each second column.

	α -GREEDY		EXP2		EXP3		α -SLOPE		EXP5		EXP6		α -STEP		EXP8		EXP9	
	EXP1																	
PAR	33	*	27	\checkmark	44	*	66	*	38	2	31	*	61	*	37	2	47	*
PER	31	*	23	\checkmark	25	*	39	*	24	1	26	\checkmark	29	\checkmark	27	1	20	\checkmark
SRS	18	\checkmark	22	\checkmark	22	\checkmark	38	*	26	3	28	5	26	\checkmark	27	\checkmark	27	1
SRZ	24	\checkmark	23	1	22	\checkmark	23	\checkmark	24	\checkmark	26	\checkmark	34	1	23	\checkmark	25	\checkmark

V. Distributed Locomotion

In addition to reconfiguration, the Roombots modules will be capable of moving around like legged robots. This section describes the three main components of our approach to decentralized locomotion control of modular robots in a non-structured environment (i.e. without using the ACMs to grab into connectors embedded in the floor): the control architecture, network topology and controller parameter optimization. The control architecture consists of a network of coupled oscillators representing a Central Pattern Generator (CPG). The control inputs for this CPG are high level parameters such as amplitude, offset and phase lags. Each oscillator is capable of producing either *oscillatory* or *rotational* joint angle signals.⁵ The oscillators are furthermore coupled such that their phases are synchronized. We applied an oscillator network topology which matched the hardware topology, e.g. a quadruped structure or a single Roombots metamodule. An evolutionary algorithm provides an automatic design of the control input parameters.

A. Controller Architecture

We designed a CPG controller which can produce two types of basic movements for each DOF: (i) *Rotational* movements that result from a continuously rotating (swivel) joint, and can provide wheel or Whegs-like [37] propulsion, and (ii) *Oscillatory* movements that periodically oscillate around a resting position. Since it is important for stable, reproducible locomotion to keep all DOF synchronized, independent of their mode, we built the controller as a distributed system of coupled phase oscillators, with one oscillator per DOF (joint) i :

$$\dot{\varphi}_i = 2 \cdot \pi \cdot \nu_i + \sum w_{ij} \cdot r_j \cdot \sin(\varphi_j - \varphi_i - \psi_{ij}) + f_\theta(\vec{s}) \quad (3)$$

$$\dot{r}_i = a_i(R_i - r_i) + f_r(\vec{s}) \quad (4)$$

$$\left. \begin{array}{l} \text{a) } \theta_i = r_i \cdot \sin(\varphi_i) + X_i \text{ (Oscillation)} \\ \text{b) } \theta_i = \varphi_i \text{ (Rotation)} \\ \text{c) } \theta_i = X_i \text{ (Locked)} \end{array} \right\} \text{servo inputs,} \quad (5)$$

⁵In the remaining part of the paper we will use *oscillator* to refer to *pattern generators* capable of producing both oscillatory and rotational output.

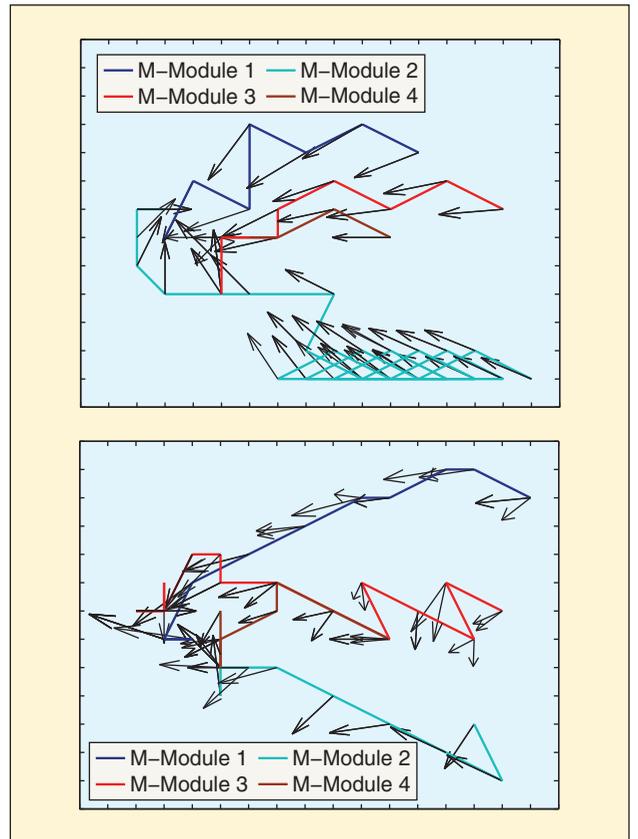


FIGURE 6 Top view at the trace-patterns of four metamodules moving towards their seeding points in the left center area. The trace follows the pivot point of the foot-hemisphere of each metamodule. Quiver plots indicate the direction of attraction at iterative steps. (a) Due to the *greedy* strategy in this plot SRS metamodules aim directly for their next seeding position. They will only be paused by a close-by metamodule with a higher priority. (b) The same experiment but with a strategy using a slope-like α value, and SRZ metamodule type. Quiver plots indicate that metamodules are repelled among each other on their way to the seeding position, however e.g. metamodule 1 is initially not affected by the presence of other metamodules, as they are sufficiently far away.

where θ_i is the servo input which can be derived with different functions corresponding to the desired servo movement. Variables r_i and φ_i are state variables which encode amplitude and phase of the oscillation. The parameters ν_i ,

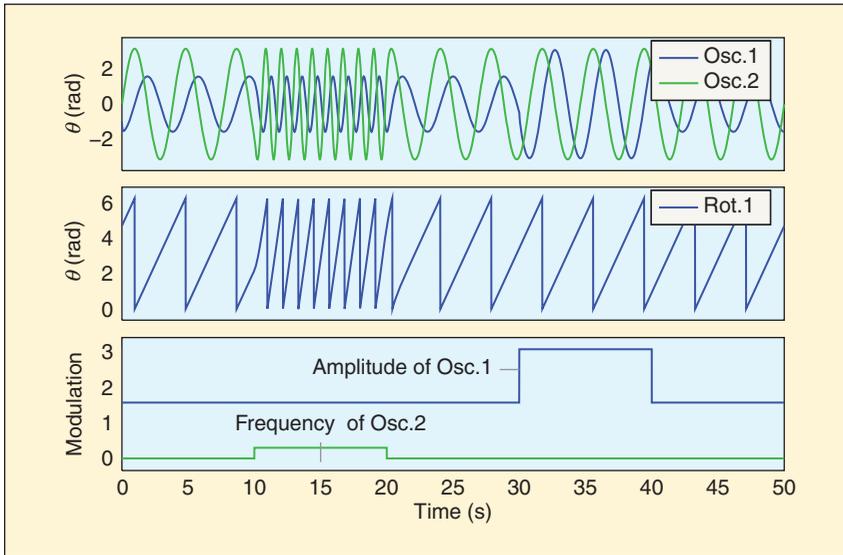


FIGURE 7 Synchronization behavior of three coupled oscillators: two in oscillatory mode (upper plot) and one in rotational mode (middle plot). All oscillators are coupled, hence they synchronize within the first seconds. Frequency modulation from $t = 10$ to 20 sec, amplitude modulation from $t = 30$ to 40 sec of the simulation.

w_{ij} and ψ_{ij} are respectively the frequency, coupling weight and phase bias of the coupling between oscillators i and j . a_i is a positive constant which determines the rise time of the amplitude to the desired value R_i . The parameters R_i , X_i , and ψ_{ij} are open parameters of which a subset (depending on the selected mode) is subject to optimization. Furthermore, this structure is capable of including sensory feedback. For this purpose the state variables can be influenced by sensory feedback signals through the functions f_θ and f_s , \vec{s} being a vector of sensor states. Note that sensory feedback is not applied in this article.

Equation 5 shows three possible modes which result in oscillations, rotations or a locked condition. In the oscillation mode, the output exhibits limit cycle behavior, thus producing a stable periodic trajectory. For rotation a constant-speed profile is generated leading to a monotonic increase of the joint angle. We also include a third mode

which allows the controller to lock some joints. One strength of this framework is that with the right parameters, rotational and oscillatory DOF will rapidly converge to a phase-locked regime, i.e. a regime with a constant phase difference between phase oscillators that are in different modes. This is highly desirable for the implementation of stable, coordinated gaits. It will also ensure that several joints remain phase-locked, even if they are controlled by oscillators implemented on different micro-controllers with slightly different clocks. Fig. 7 shows this synchronization behavior between three DOFs, with two activated in oscillation mode and one in rotation mode.

B. CPG Topology

When designing CPGs, the network coupling parameters w_{ij} and ψ_{ij} between different oscillators are important. For known types of locomotion gait patterns, such as quadrupedal or snake gaits, the coupling architecture can be specified based on biological observations. Here the goal is to find different and unexpected gaits, which an arbitrarily shaped modular robot could potentially create. Hence we do not specify a pre-defined oscillator network topology. We let the coupling structure of the CPG correspond to the robot's morphology, i.e. phase oscillators of neighbor DOF are coupled together. We use one common frequency for all oscillators ($\nu_i = 0.26$ Hz), bi-directional couplings follow the rule such that $\psi_{ij} = -\psi_{ji}$ (Fig. 8b). All coupling weights are set to 2, phase differences ψ_{ij} are open parameters and subject to optimization.

We do not induce symmetry artificially, i.e. we do not apply any mirroring of parameter sets along our network. Applying symmetry is usually a good strategy to reduce the number of open parameters. However it might also limit the resulting gaits, as it restricts the possible variety of parameters.

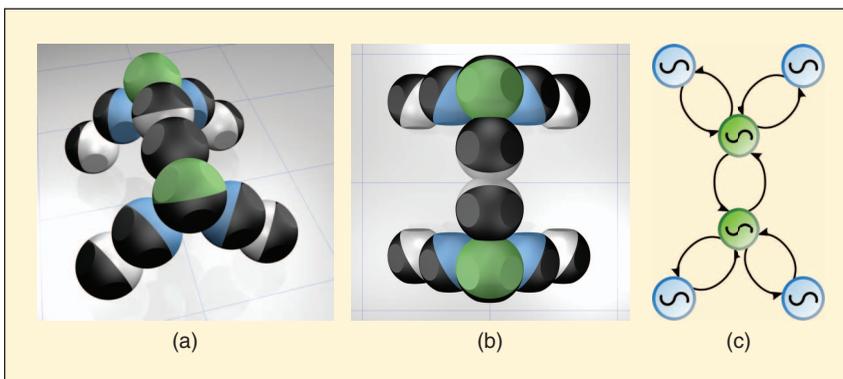


FIGURE 8 (a)–(c) *Quad*: Quadruped Roombots structure with six units. (c) shows the central pattern generator topology for this Roombots robot. One oscillator is assigned to each Roombots module, the network mimics the physical topology. Six oscillator units are depicted for this quadruped (c), one joint per Roombots unit is actuated.

C. Optimal Gait Generation

We are interested in the generation of optimal gaits for arbitrary robot morphologies. In particular, we want to evolve the assignment of different movement types (such as rotation and oscillation) for each degree of freedom, as well as the control parameters for each of these movement types. The optimization algorithm can then be described by two layers. The *outer* layer performs structural optimization

(selecting movement types for each DOF). The *inner* layer on the other hand performs parametric optimization on *continuous* valued parameters corresponding to the selected movement types by the *outer* layer.

In this paper, a modified version of the standard Particle Swarm Optimization with a constriction factor [38],[39] is used to perform the optimization process. PSO is a stochastic, population based optimization method using principles of collaboration rather than competition to evolve individuals. Compared to other stochastic algorithms (e.g. genetic algorithms, simulated annealing, genetic programming or evolution strategies), we preferred PSO because of its superior performance on parametric simulation (see for instance [40] for a comparative study).

In PSO, each individual is represented by a position and velocity vector, representing respectively the particle's parameter values and search direction. The evolution of each particle in the swarm is then governed by equation 6.

$$\vec{v}_i(t+1) = K \cdot [\vec{v}_i(t) + c_1 r_1 (\vec{p}_i - \vec{x}_i(t)) + c_2 r_2 (\vec{p}_g - \vec{x}_i(t))] \quad (6)$$

$$\vec{x}_i(t) = \vec{x}_i(t-1) + \vec{v}_i(t),$$

where $\vec{v}_i(t)$ is the velocity vector, $\vec{x}(t)$ is the position vector, K is a *constriction* factor, c_1 and c_2 are two constants, r_1 and r_2 are two pseudo-random numbers in the range $[0, 1]$, p_i is the best known solution vector of particle i and p_g is the global best known solution vector. Constants c_1 and c_2 were set to ensure convergence (see for more detail, [41]).

The PSO algorithm described thus far is used for the *inner* layer optimization of the continuous parameters of a specific selection of movement types. Particles are initially uniformly distributed over the possible combinations of movement types. In each such combination, particles share the same parameters, and an independent PSO optimizes their respective solutions. The task of the *outer* layer is then to do the structural optimization and to move particles from one combination of movement types to another.

The *outer layer* consists of a set of mutation operators inspired by Genetic Algorithms. Similar to the velocity update of the PSO, the probability of mutation of each actuated degree of freedom is composed of:

- P_e : exploration probability of mutation to a movement type (oscillation, rotation or locked) other than the current one
- P_l : local probability of mutation to the movement type which is part of the selection with the best results in the particle's history
- P_g : global probability of mutation similar to P_l but taken from the best results taken over all the particles

Governed by these three probabilities, particles will be mutated to different combinations of movement types during the optimization process. Once a particle moves to a different parameter space it is incorporated in the PSO running locally in that space.

A main challenge is to choose appropriate values for the different probabilities P_e , P_l and P_g . In general, we want to

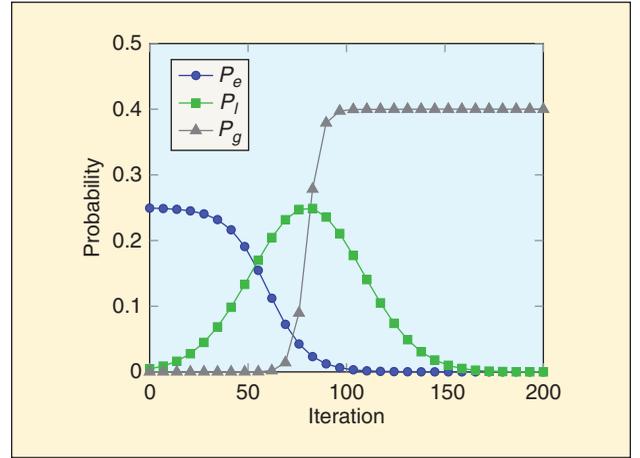


FIGURE 9 Mutation probability characteristics for the exploration probability P_e , local probability P_l and global probability P_g , emphasizing early exploration and late convergence.

stimulate exploration in the early phases of the optimization, visiting many possible combinations of movement types. Then, as the optimization progresses, particles should start exploring their local known best solutions in more detail. Finally we want the particles to converge in the best known space, as if selecting the best configuration of movement types. The system then starts behaving as a standard PSO with a fixed configuration of movement types as more and more particles are attracted.

The desired behavior can be designed by varying the probabilities P_e , P_l and P_g as the optimization progresses. In this paper, the exploration and global probability were modeled using a sigmoid function. The local probability was modeled using a gaussian function. Fig. 9 shows the probability characteristics used in all the experiments.

D. Experimental Setup

We performed several experiments applying our CPG and optimization framework. Firstly we were interested in exploring the locomotion abilities of the four types of metamodules configurations (PAR, PER, SRS, and SRZ, Fig. 3). Our motivation for testing metamodules is that they represent the simplest possible robot shape built from two Roombots modules (six DOF). In addition, one quadruped shape was designed by using six modules featuring symmetry. Three DOFs inside the spine and four DOFs in the hip joints are used allowing quadruped locomotion (Quad6 robot, Fig. 8). The latter structure was used to verify our approach on more complex shapes.

We are also interested to explore which type of movement, oscillatory or rotational, would lead to the highest locomotion speed. We conducted the following optimization experiments with different possible combinations of the four joint modes: (1) *pure rotation*, all DOF are in the rotation mode, (2) *hybrid rotation* with DOF either in rotation or locked mode, (3) *pure oscillation*, all DOF are in oscillation mode, and (4) *fully hybrid*, DOF are in oscillation, rotation, or locked mode, whereas different modes can coexist within the robot.

Each solution, generated by the evolutionary algorithm, is evaluated in Webots [35], a simulation tool based on ODE providing collision detection, rigid body dynamics and actuator properties. The average locomotion speed determines the *fitness value* of a particular solution, which is sent back to the optimizer. An RB module–module collision detection penalizes unrealistic solutions with a zero fitness value.

E. Results and Discussion

Fig. 10 shows the average and standard deviation of the robots' speed over ten optimizations with different initial conditions. Each optimization process uses 50 particles and 100 iterations to find the optimal solution. The results illustrate three interesting properties of this framework. (i) The capability of combining different modes in the fully hybrid setting results in finding solutions with higher values for both average speed and variance in all the robots. This results in faster and more diverse solutions in the same number of simulations. (ii) Results from pure rotation show a drastic reduction in the robot performance. Allowing the robot to lock some of its degrees of freedom, when the others are in rotational mode, helps to avoid self-collision during robot locomotion. The performance of this mode is comparable in terms of characteristics with the oscillation mode since both include locked joints (defined implicitly in oscillation mode due to zero amplitudes). (iii) The results indicate that the performance of oscillation and rotation modes are strongly dependent on the robot shape. In the case of PAR and Quad6, oscillation largely outperforms hybrid rotation. For PER, SRS and SRZ however, similar performance for both modes is observed. This shows that for a given robot shape, it is not trivial to select either oscillation or rotation. The complex

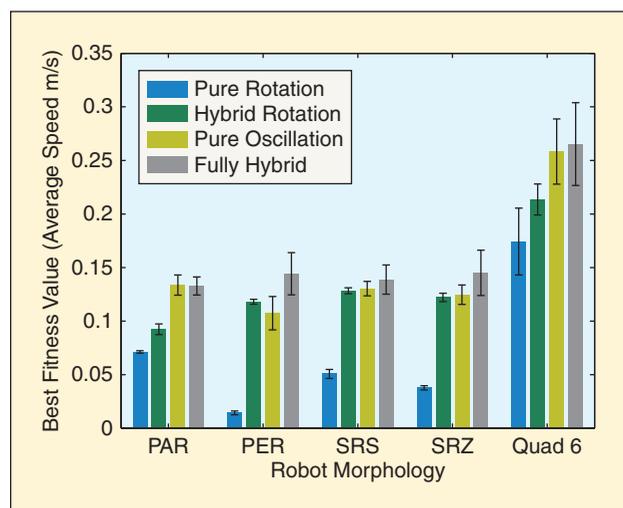


FIGURE 10 The optimization results from five different robot structures such as metamodules in PAR, PER, SRS and SRZ configuration, and a six-unit quadruped structure (Fig. 8) are shown. Every experiment was repeated 10 times, with different initial conditions. Fitness evaluation is based on traveled distance over time. The hybrid optimization mode is in all cases at least as good as any other optimization mode. When checking resulting gaits we find that it includes results both with rotating and oscillating joint patterns within one robot structure.

interaction of the different DOF of a robot shape and the environment determine whether rotation, oscillation or combination of them will provide the best performance. In almost all experiments, having a mixture of both movement types (fully hybrid) yields better results.

Fig. 11 and Fig. 12 show two examples of arbitrary gait patterns, for a PER–metamodule and the quadruped robot from Fig. 8, respectively⁶. The joint movements of the PER–metamodule in Fig. 11 are purely oscillatory, but the robot rolls over itself from cycle to cycle. This is possible in a metamodule–robot because six joints are connected serially—amplitude and velocity are adding up in those structures. The gait for the quadruped robot in Fig. 8 was derived in the hybrid optimization mode, and achieves oscillatory and rotational joint movements: both spine joints are in oscillatory mode, one of the outer joints is blocked, one is in rotational mode, and the remaining two leg joints oscillate. This setting results in a wind-up like gait which propels the robot with 33 cm/s, while the walking gait—the first solution that comes to mind when designing a gait for this robot—only results in a maximum speed of 21 cm/s (Fig. 10).

The CPG model has several interesting features that make it well suited for modular robotics. (i) Our model can produce stable rhythmic patterns such that the dynamical system rapidly returns to its rhythmic behavior after perturbations of the state variables. (ii) The applied CPG model only needs a few control parameters, in our case amplitudes, offsets and phase lags. Hence it can reduce the dimensionality of the control problem such that the optimization algorithm only needs to modulate a small number of control signals. (iii) Another useful CPG property is its ability to generate different gaits, which one can achieve by setting the network coupling weights and topology. In this way we can reproduce animal-like gaits. This has been done in several works for legged and modular robots ([42], [25], [27]). Yet one has the option to keep the network topology open, and to let new and unexpected gaits emerge. The latter approach is even more appealing for modular robots, where ideal gaits are initially unknown due to new robot topologies. Hand coding and editing the gaits is tiring and time-consuming, whereas the proposed framework can reproduce animal-like gaits or find alternative solutions. (iv) This CPG model can be used to generate different types of locomotion patterns. Our control architecture offers a high variety of basic locomotion patterns e.g. it can generate any combination of oscillatory and rotational movements. This allows us to apply those movements to the robot while ensuring that they are in their phase-locked regimes. We observed interesting and unexpected locomotion gaits being derived by our combined architecture. In this work we used specific patterns, such as sine-waves for oscillation, and constant speed for rotation. However the framework is kept open and more complex patterns can be

⁶Locomotion videos are available at the Roombots webpage: <http://biorob.epfl.ch/page38279.html>.

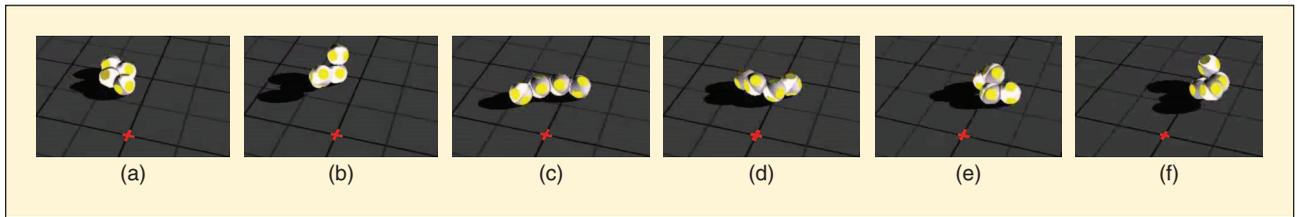


FIGURE 11 Snapshots for an evolved gait with a Roombots PER-metamodule, the module “rolls” from (a)–(f), about one cycle is shown. It starts in a folded posture and rotates while unfolding towards the right. When folding again, top and bottom Roombots module are switched. This speeds the robot up to 11 cm/s in an overall rather straight gait.

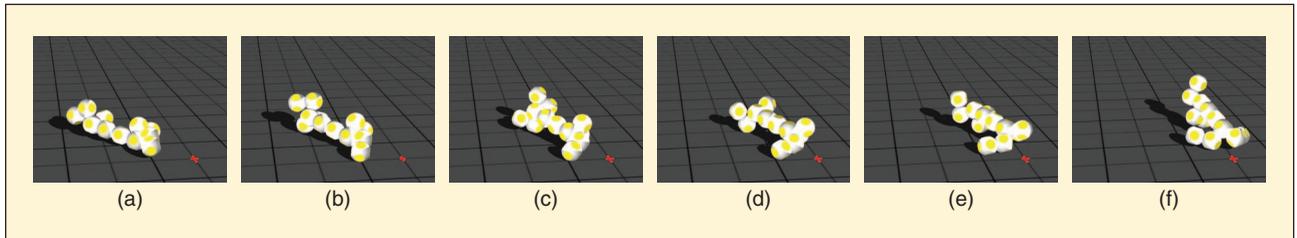


FIGURE 12 (a)–(f) The robot moves from left to the right, half a cycle is shown and markers are placed on the ground for reference. This quadruped robot structure shows one of the fastest “gaits”, however does not behave like a typical quadruped. Rather it propels with a winding-like mechanism: by leaving two extremities on the ground, it winds the remaining two of them around the body stem and vice versa in the next cycle. The overall direction of movement is sideways regarding the body stem with about 33 cm/s.

implemented easily, which eventually could lead to an even higher versatility of derived gaits.

The proposed framework provides an important feature: the optimization algorithm can choose and switch between oscillatory and rotational joint movements, for any joint, at any time during the optimization process, and is fully automated. The user is not required to, but can pre-assign a movement-type to a joint type. This feature gives the possibility to explore which kind of joint movements can result in more efficient gaits for a newly designed robot. To our best knowledge this is the first work in the field where a modular robot controller-optimizer framework can derive such a wide variety of locomotion patterns. The result for five different robots shows that the hybrid mode systematically leads to the best solutions. In other words it is better to let the optimization algorithm find suitable modes for each joint rather than fixing them by hand.

VI. Conclusion

In this paper we have presented our approaches for reconfiguration and locomotion for Roombots self-reconfiguring modular robots, on structured and non-structured environments. This is part of our long term vision for the Roombots project featuring adaptive and self-assembling furniture made from modular robots. Reconfiguration through locomotion uses Roombots metamodules applying caterpillar-like movements attaching at embedded connectors in the environment to move and shape change. Metamodules are attracted and guided by a virtual force-field, they use broadcast signals, look-up tables of collision clouds and simple assumptions about their near environment to reach their seeding positions, which are currently hand coded. We presented results from simulation tests with

four different metamodule configurations and three force-field models. We have derived a framework for locomotion control of modular robots in un-structured environments, where a central pattern generator (CPG) as the motion controller and the optimization algorithm are tightly connected. The CPG is implemented as a system of coupled oscillators, different output filters can be applied to derive the desired joint behavior such as synchronized oscillation, rotation or stop control. Open control parameters are amplitude, offset and phase lag, and the three joint modes. They are automatically selected, assigned and optimized by the optimization framework. This enables us to derive gait patterns for traditional robots like quadrupeds (oscillatory and stop joint control) but also for robots featuring the more capable, continuous rotational joints, e.g. Wheggs-like robots or in our case the Roombots modules. Extensive experiments are performed in simulation for four metamodule types (the same ones as used above for reconfiguration, but now in a non-structured environment), and quadruped robots made from multiple Roombots units. We have presented results of our optimization framework deriving pure oscillatory or rotational joint controllers based on CPGs, as well as hybrid controllers. Optimized robot gaits for the latter type result often in mixed-mode joint controllers with surprising characteristics and very competitive performance.

Research on locomotion control will be pursued in order to address the problems of (i) how to properly include sensory feedback for improving the efficiency and robustness of locomotion patterns, and (ii) of navigation, i.e. how to modulate speed and direction to reach a specific location in a room. Research on reconfiguration will explore how to include passive elements in the reconfiguration and how to create intuitive user interfaces. We plan to extend the hardware by passive,

light-weight elements like carbon-fiber plates to built more complex, and a larger variety of furniture shapes. For designing these furniture shapes, we are implementing a graphical user interface that will allow lay users to enter desired shapes in an intuitive way and then automatically generate the seeding recipe for our reconfiguration algorithm. The GUI will also be used to specify desired locations of the furniture in their environment that will be provided to the navigation controller. It is still a long road, but we hope to make steady progress towards our long term vision of adaptive furniture in our day-to-day environment.

VII. Acknowledgments

This project has received funding from the EPFL and from the European Community's Seventh Framework Programme FP7/2007–2013—Future Emerging Technologies, Embodied Intelligence, under the grant agreements no. 231688 (Locomorph) and no. 231451 (EVRYON). We gratefully acknowledge the technical support of André Guignard, André Badertscher, Peter Brühlmeier, Philippe Voessler, and Manuel Leitons in the design and construction of the robot modules.

We also thank Philippe Laprade and Mikael Mayer for their participation to the Roombots software design.

References

[1] M. Vona, C. Detweiler, and D. Rus, "Shady: Robust truss climbing with mechanical compliances," in *Proc. Int. Symp. Experimental Robotics*, 2008, pp. 431–440.

[2] N. A. Streitz, J. Geißler, and T. Holmer, "Roomware for cooperative buildings: Integrated design of architectural spaces and information spaces," in *Proc. 1st Int. Workshop Cooperative Buildings. Integrating Information, Organization, and Architecture, CoBuild'98*, Darmstadt, Germany, Feb. 1998, p. 4.

[3] E. Hornecker and J. Buur, "Getting a grip on tangible interaction: A framework on physical space and social interaction," in *Proc. SIGCHI Conf. Human Factors in Computing Systems*, Apr. 2006.

[4] T. Fukuda, S. Nakagawa, Y. Kawauchi, and M. Buss, "Self organizing robots based on cell structures—CEBOT," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 1988, pp. 145–150.

[5] H. Kurokawa, A. Kamimura, E. Yoshida, K. Tomita, S. Kokaji, and S. Murata, "M-TRAN II: Metamorphosis from a four-legged walker to a caterpillar," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems, 2003 (IROS'03)*, Oct. 2003, vol. 3, pp. 2454–2459.

[6] P. Pillai, J. Campbell, G. Kedia, S. Moudgal, and K. Sheth, "A 3D fax machine based on claytronics," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2006, pp. 4728–4735.

[7] M. Asadpour, M. H. Z. Ashtiani, A. Sproewitz, and A. Ijspeert, "Graph signature for self-reconfiguration planning of modules with symmetry," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems, 2009 (IROS'09)*, St. Louis, 2009.

[8] E. Yoshida, S. Murata, A. Kamimura, K. Tomita, H. Kurokawa, and S. Kokaji, "A motion planning method for a self-reconfigurable modular robot," in *Proc. 2001 IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2001, vol. 1, pp. 590–597.

[9] Z. Butler, K. Kotay, D. Rus, and K. Tomita, "Generic decentralized control for lattice-based self-reconfigurable robots," *Int. J. Robot. Res.*, vol. 23, no. 9, pp. 919–937, Sept. 2004.

[10] P. White and M. Yim, "Scalable modular self-reconfigurable robots using external actuation," in *Proc. 2007 IEEE/RSJ Int. Conf. Intelligent Robots and System*, 2007, pp. 2773–2778.

[11] K. Stoy, "Using cellular automata and gradients to control self-reconfiguration," *Robot. Auton. Syst.*, vol. 54, no. 2, pp. 135–141, Feb. 2006.

[12] W. Shen, P. Will, A. Galstyan, and C. Chuong, "Hormone-inspired self-organization and distributed control of robotic swarms," *Auton. Robots*, vol. 17, no. 1, pp. 93–105, 2004.

[13] J. V. Neumann, *Theory of Self-Reproducing Automata*, A. W. Burks, Ed. Urbana, IL: Univ. Illinois Press, 1966.

[14] P. Varshavskaya, L. P. Kaelbling, and D. Rus, "Automated design of adaptive controllers for modular robots using reinforcement learning," *Int. J. Robot. Res.*, vol. 27, no. 3–4, pp. 505–526, Mar. 2008.

[15] R. Fitch, Z. Butler, and D. Rus, "Reconfiguration planning for heterogeneous self-reconfiguring robots," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems, 2003 (IROS'03)*, 2003, vol. 3, pp. 2460–2467.

[16] R. Fitch and Z. Butler, "Million module march: Scalable locomotion for large self-reconfiguring robots," *Int. J. Robot. Res.*, vol. 27, no. 3–4, pp. 331–343, Mar. 2008.

[17] K. D. Kotay and D. L. Rus, "Scalable parallel algorithm for configuration planning for self-reconfiguring robots," in *Sensor Fusion and Decentralized Control in Robotic Systems III*, vol. 4196, G. T. McKee and P. S. Schenker, Eds. Boston, MA: SPIE, Oct. 2000, pp. 377–387.

[18] D. Rus and M. Vona, "Crystalline robots: Self-reconfiguration with compressible unit modules," *Auton. Robots*, vol. 10, no. 1, pp. 107–124, 2001.

[19] D. Dewey, M. Ashley-Rollman, M. D. Rosa, S. Goldstein, T. Mowry, S. Srinivasa, P. Pillai, and J. Campbell, "Generalizing metamodules to simplify planning in modular robotic systems," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems, 2008 (IROS'08)*, 2008, pp. 1338–1345.

[20] A. J. Ijspeert, "Central pattern generators for locomotion control in animals and robots: A review," *Neural Netw.*, vol. 21, no. 4, pp. 642–653, May 2008.

[21] M. Yim, "Locomotion with a unit modular reconfigurable robot," Ph.D. dissertation, Mech. Eng. Dept., Stanford Univ., 1994.

[22] J. Bongard, V. Zykov, and H. Lipson, "Resilient machines through continuous self-modeling," *Science*, vol. 314, no. 5802, pp. 1118–1121, 2006.

[23] K. Stoy, W. Shen, and P. Will, "Implementing configuration dependent gaits in a self-reconfigurable robot," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA2003)*, 2003.

[24] A. Kamimura, H. Kurokawa, E. Tshida, K. Tomita, S. Murata, and S. Kokaji, "Automatic locomotion pattern generation for modular robots," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA2003)*, 2003.

[25] A. Kamimura, H. Kurokawa, E. Yoshida, S. Murata, K. Tomita, and S. Kokaji, "Distributed adaptive locomotion by a modular robotic system, M-TRAN II," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS2004)*, 2004, pp. 2370–2377.

[26] D. Marbach and A. J. Ijspeert, "Online optimization of modular robot locomotion," in *Proc. IEEE Int. Conf. Mechatronics and Automation (ICMA 2005)*, 2005, pp. 248–253.

[27] A. Sproewitz, R. Moeckel, J. Maye, and A. J. Ijspeert, "Learning to move in modular robots using central pattern generators and online optimization," *Int. J. Robot. Res.*, vol. 27, no. 3–4, pp. 423–443, Mar. 2008.

[28] E. Mytilinaios, M. Desnoyer, D. Marcus, and H. Lipson, "Designed and evolved blueprints for physical self-replicating machines," in *Proc. 9th Int. Conf. the Simulation and Synthesis of Living Systems (Artificial Life IX)*, vol. 2004, MIT, 2004, pp. 15–20.

[29] A. Sproewitz, A. Billard, P. Dillenbourg, and A. J. Ijspeert, "Roombots—mechanical design of self-reconfiguring modular robots for adaptive furniture," in *Proc. 2009 IEEE Int. Conf. Robotics and Automation*, Kobe, Japan, 2009, pp. 4259–4264.

[30] W.-M. Shen, M. Krivokon, H. Chiu, J. Everist, M. Rubenstein, and J. Venkatesh, "Multimode locomotion via SuperBot reconfigurable robots," *Auton. Robots*, vol. 20, no. 2, pp. 165–177, Mar. 2006.

[31] S. Murata, E. Yoshida, A. Kamimura, H. Kurokawa, K. Tomita, and S. Kokaji, "M-TRAN: Self-reconfigurable modular robotic system," *IEEE/ASME Trans. Mechatronics*, vol. 7, no. 4, pp. 431–441, 2002.

[32] V. Zykov, E. Mytilinaios, B. Adams, and H. Lipson, "Self-reproducing machines," *Nature*, vol. 435, p. 163, 2005.

[33] Y. Terada and S. Murata, "Automatic modular assembly system and its distributed control," *Int. J. Robot. Res.*, vol. 27, no. 3–4, pp. 445–462, Mar. 2008.

[34] A. Sproewitz, M. Asadpour, Y. Bourquin, and A. Ijspeert, "An active connection mechanism for modular self-reconfigurable robotic systems based on physical latching," in *Proc. IEEE Int. Conf. Robotics and Automation, 2008 (ICRA'08)*, 2008, pp. 3508–3513.

[35] O. Michel, "Webots: Professional mobile robot simulation," *Int. J. Adv. Robot. Syst.*, vol. 1, no. 1, pp. 39–42, 2004.

[36] P. Mudry, J. Ruffin, M. Ganguin, and G. Tempesti, "A hardware-software design framework for distributed cellular computing," in *Proc. 8th Int. Conf. Evolvable Systems: From Biology to Hardware*. Springer, 2008, p. 82.

[37] R. T. Schroer, M. J. Boggess, R. J. Bachmann, R. D. Quinn, and R. E. Ritzmann, "Comparing cockroach and whegs robot body motion," in *Proc. IEEE Int. Conf. Robotics and Automation 2004*, Apr. 2004, pp. 3288–3293.

[38] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Networks*, 1995, vol. 4, pp. 1942–1948.

[39] R. Eberhart and Y. Shi, "Comparing inertia weights and constriction factors in particle swarm optimization," in *Proc. 2000 Congr. Evolutionary Computation*, 2000, vol. 1, pp. 84–88.

[40] Y. Bourquin, "Self-organization of locomotion in modular robots," Ph.D. dissertation, Bioinspired Robotics Group (BIRG), Ecole Polytechnique Fédérale de Lausanne (EPFL), 2003.

[41] M. Clerc and J. Kennedy, "The particle swarm—Explosion, stability, and convergence in a multidimensional complex space," *IEEE Trans. Evol. Comput.*, vol. 6, no. 1, pp. 58–73, 2002.

[42] H. Kimura, Y. Fukuoka, and A. H. Cohen, "Adaptive dynamic walking of a quadruped robot on natural ground based on biological concepts," *Int. J. Robot. Res.*, vol. 26, no. 5, pp. 475–490, May 2007.