Available online at www.sciencedirect.com
SCIENCE DIRECT®
Robotics and Autonomous Systems 54 (2006) 135–141
Robotics and Autonomous Systems
www.elsevier.com/locate/robot

Using cellular automata and gradients to control self-reconfiguration

K. Stoy

The Maersk Mc-Kinney Moller Institute for Production Technology, University of Southern Denmark, Campusvej 55, DK-5230 Odense M, Denmark

Received 15 September 2004; accepted 13 September 2005
Available online 22 November 2005

Abstract

Self-reconfigurable robots are built from modules, which are autonomously able to change the way they are connected. Such a robot can, through this self-reconfiguration process, change its shape. The process has proved to be difficult to control, because it involves control of a distributed system of mechanically coupled modules connected in time-varying ways. In this paper we present an approach to the control problem where the desired configuration is grown from an initial seed module. Seeds produce growth by creating a gradient in the system, using local communication, which spare modules descend to locate the seed. The growth is guided by a cellular automaton, which is automatically generated on the basis of a three-dimensional CAD model or a mathematical description of the desired configuration. The approach is evaluated in simulation and we find that the self-reconfiguration process always converges and the time to complete a configuration scales approximately linearly with the number of modules. However, an open question is how the simulation results transfer to a physically realized self-reconfigurable robot.

Keywords: Self-reconfigurable robots; Distributed control

1. Introduction

Reconfigurable robots are robots built from modules. These robots can be reconfigured by changing the way these modules are connected. If a robot is able to change the way the modules are connected autonomously, the robot is a self-reconfigurable robot.

Self-reconfigurable robots are versatile because they can adapt their shape to fit the task. Self-reconfigurable robots are also robust because, if a module fails, it can be ejected from the system and be replaced by a spare module. Potential applications for such robots include search and rescue missions, planetary exploration, building and maintaining structures in space, and entertainment. In order to realize these applications, research has to focus both on the development of the hardware of self-reconfigurable robots and on their control. The latter is the focus of this paper.

One of the fundamental control problems of self-reconfigurable robots is control of the self-reconfiguration process: the process by which the robot changes from one shape to another. An example of a self-reconfiguration process is shown in Fig. 1, where a simulated self-reconfigurable robot reconfigures from a random initial configuration to a configuration resembling a chair.

The method proposed here consists of two steps. In the first step a three-dimensional (3D) CAD model, representing the desired configuration, is transformed into a cellular automaton representation. The desired configuration and the corresponding cellular automaton is made porous to ensure that it does not contain local minima, hollow, or solid sub-configurations which can trap spare modules.

The second step is the actual self-reconfiguration process. The desired configuration is grown from a seed using the cellular automaton to guide the growth process. If a cellular automaton rule indicates that a neighbouring module is needed at an unfilled position, a gradient is created in the system using local communication. Spare modules then descend this gradient to reach the unfilled position. A local, distributed algorithm is also implemented to ensure that the self-reconfigurable robot stays connected during the self-reconfiguration process.

The solution to the self-reconfiguration problem presented here is a new combination of existing ideas. The combination we present provides a scalable, systematic, and convergent solution to the self-reconfiguration problem.

The paper proceeds as follows. In Section 2, the related work is described. In Section 3, we describe the algorithm we

E-mail address: kaspers@mip.sdu.dk.

0921-8890/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.robot.2005.09.017

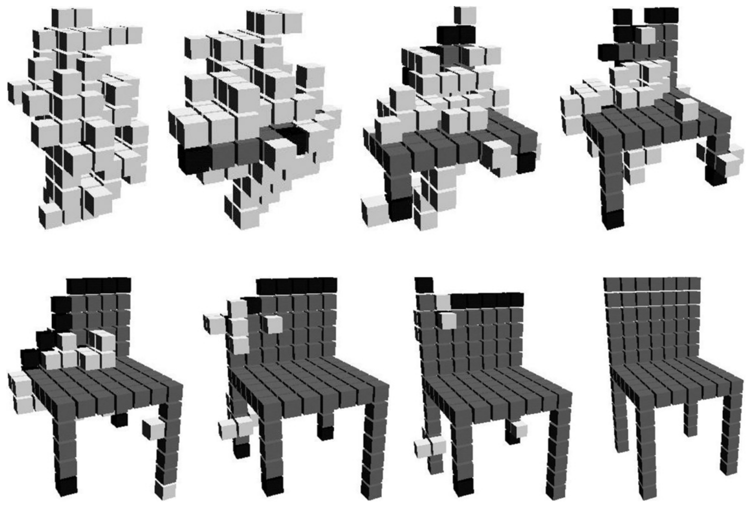

Fig. 1. This figure shows a simulated self-reconfigurable robot transforming from an initial random configuration to a configuration resembling a chair.

use to transform the 3D CAD model into a cellular automaton representation. Section 4 describes how the cellular automaton interacts with the other components of the system to realize the self-reconfiguration algorithm. Finally, in Section 6, the system is evaluated using a simulated self-reconfigurable robot where scalability and convergence properties are documented.

2. Related work

The self-reconfiguration problem is: given a start configuration, possibly a random one, how to move the modules in order to arrive at the desired final configuration. It is computationally intractable to find the optimal solution (see [1] for a discussion). Therefore, self-reconfiguration planning and control are open to heuristic-based methods.

Chirikjian, Pamecha, and Chiang propose iterative improvement as a way around the computational explosion [1–3]. The idea is to find a suboptimal sequence of moves which leads from the initial configuration to the final configuration. This suboptimal sequence of moves can then be optimized using local searches. Rus et al. simplify the planning problem by using a specific intermediate chain configuration, which is easy to configure into and out of [4]. Kotay, Unsal, and Prevas propose a hierarchical planner [5–7]. At the highest level, some of the motion constraints of the underlying hardware are abstracted away to facilitate efficient planning. Based on these high-level plans, the lower level then produces the detailed sequence of actions. Another approach, suggested by Kotay [5], is to use meta-modules consisting of a small number of modules. The idea is that, by planning at the meta-module level, there are no or few motion constraints. This makes the planning tractable. On the other hand, meta-modules consist of several modules, making the granularity of the robot larger. A related approach is to make sure that the robot maintains a uniform scaffolding structure, facilitating planning [8].

Butler implemented the distributed Pacman algorithm on the Crystalline robot [9]. The Pacman algorithm was improved and implemented on the Telecubes by Vassilvitskii [10]. These two robots have few motion constraints, making the planning problem easier.

The approaches reviewed so far are deterministic and planning-based. The following approaches are not deterministic but are much simpler, because a planning process is not needed. In early work, the idea was to use local rules as far as possible and then add randomness to deal with the problems that could not be solved by using local rules. This is, for instance, true for the work on Fracta [11,12] and also later work on other robots [13,14]. The problem tended to be that, even though the robot often ended up in the desired configuration, it was not always so. We refer to this as the problem of convergence. This problem was also present in the work of Yim et al. [15,16]. However, the chance of reaching the final configuration was higher in this work, because local communication was used to get some idea about the global shape at the local level.

There are two solutions to the problem of convergence. One solution, proposed by Bojinov et al. [17,18], is not to focus on a specific configuration. Instead, the idea is to build something with the right functionality. Using this approach it is acceptable if a few modules are stuck as long as the structure maintains its functionality. Alternatively, Jones et al. insist on a specific configuration, but achieve convergence by enforcing a specific sequence of construction [19]. In the work presented here, we make the configuration porous and thereby remove local minima, hollow, or solid sub-configurations, guaranteeing convergence.

In the work presented here, we use 3D cellular automata (CA). Cellular automata were introduced by Von Neumann [20], and were introduced in the context of self-reconfigurable robots by Butler et al. [21]. In their work, cellular automaton rules are used to control locomotion both in fixed configurations and in cluster-flow locomotion. In our work we use 3D cellular automata to represent configurations.

3. Cellular automaton generator

It is difficult to hand-code local rules, which result in a desired configuration being assembled. Therefore, we need

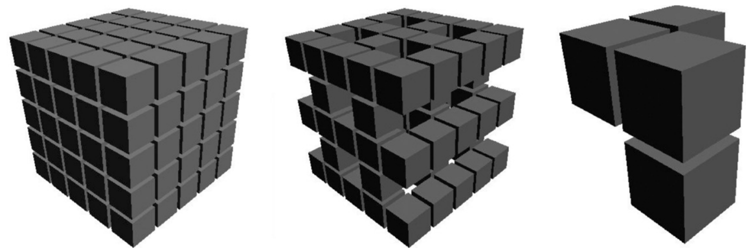

Fig. 2. A CAD model of a cube has been approximated with a configuration of 125 cubic modules (left). The configuration is made porous by removing modules that can cause local minima, create hollow, or solid sub-configurations (middle). This is achieved by enforcing a specific sub-structure on the configuration (right).

a systematic and automatic way of transforming a human-understandable description of a desired configuration into the local rules, which we will use for control. In our system, the desired configuration is specified using a connected three-dimensional volume in the Wavefront.obj file format. Most CAD programs can export in this format and can therefore be used to specify the desired configuration.

The three-dimensional model is transformed into a cellular automaton, which represents relationships between neighbour modules in the desired configuration. The algorithm that transforms the three-dimensional CAD model, representing the desired configuration, to a cellular automaton works as follows.

(1) The CAD model is approximated with a connected configuration of modules: the user places the first module inside the volume of the CAD model then neighbour positions are added recursively until the surface of the CAD model is reached.
(2) Modules are removed from the configuration to avoid configurations with local minima, hollow, or solid sub-configurations. A simple way of achieving this is to enforce a specific sub-structure on the configuration that guarantees these properties (see Fig. 2).
(3) Each module i in the configuration is assigned a unique number $s(i)$.
(4) For each neighbouring pair of modules i, j, a rule is generated. This rule is of the form: if the CA of the module in direction $\overrightarrow{ij}$ is in state $s(j)$ then this CA should change to state $s(i)$.
(5) The final cellular automaton contains the set of CA rules and starts in an initial state, called wandering, which is distinct from any $s(i)$ value.

The introduction of the sub-structure makes the configuration porous and thereby simplifies the reconfiguration problem, because it can be assumed that the configuration does not contain local minima, hollow, or solid sub-configurations. This simplification means that the system is convergent by design.

The self-reconfigurable robot can be thought of as simulating an infinite three-dimensional lattice of identical cellular automata, because wandering modules move into positions where a state change is needed. The use of cellular automata here is unusual, since modules only change state once: from wandering to the state of their final position in the configuration. However, in scenarios where the system goes through a sequence of configurations to achieve locomotion, the full functionality of cellular automata can be used.

4. From cellular automaton to desired configuration

Starting from a random configuration, the robot needs to reconfigure into the desired configuration specified by the CA. The self-reconfiguration algorithm consists of three components: a state propagation mechanism; a mechanism to create gradients in the system; and a mechanism the modules use to move without getting disconnected from the structure. We will look at these in turn.

4.1. State propagation

All the modules are initially connected in a random configuration, have a copy of the cellular automaton, and start in the wandering state. An arbitrary module is given a state number chosen randomly from the set of states $s(i)$ allocated by the CA generator. The idea is to grow the configuration from this seed module. The neighbouring modules connected to the seed change state according to the cellular automaton rules. The seed module can detect whether a neighbour is missing using sensors and the cellular automaton rules. If this is the case, the seed attracts a wandering module to the unfilled position. When a module has reached an unfilled position and has changed state, it also acts as a seed. Such modules are said to be finalized. A module stops acting as a seed when all the neighbour relationships, described by the cellular automaton rules, are satisfied.

4.2. Creating a gradient using local communication

Seed modules attract wandering modules by creating a gradient in the system that the wandering modules descend. The gradient is created as follows: a seed module acts as a source and sends out zero, representing the concentration of an artificial chemical, to all its neighbours. A non-source module calculates the concentration of the artificial chemical at its position by taking the minimum received value and adding one. This concentration is then propagated to all neighbours, and so on. A module that has not received a concentration value is defined to be *uninitialized* and one that has to be *initialized*. Note that, since we assume that the configuration is porous, wandering modules can descend the gradient of concentrations to locate the source without getting trapped.

If wandering modules have to rely on the basic concentration gradient to locate the source, they have to move around randomly for a while to detect the direction of the gradient. In order to avoid this, we introduce a gradient vector that makes the direction of the gradient available locally and thereby eliminates unnecessary moves. The basic gradient implementation is extended with a vector indicating the local direction of the gradient. This vector is updated by taking the vector from the neighbour with the maximum concentration, adding a unit vector in the direction of this neighbour, and renormalizing the result.

4.3. Descending the vector gradient

Wandering modules descend the vector gradient to reach unfilled positions. Unfortunately, the wandering modules cannot move independently of each other, because they depend on each other for connection to the robot. The problem is then to keep the system connected while allowing wandering modules to move. Modules or groups of modules, which get disconnected, will fall down and may be damaged or unable to reconnect. Yim et al. [16] assume that each module has a sensor that can detect whether it is about to disconnect a group of modules. We think it may be hard to realize this sensor and have therefore developed an algorithmic solution to this problem.

The assumption of the algorithm is that all modules that are finalized are connected, because of the way the state propagation mechanism works. These modules act as sources for a new connection gradient. Wandering modules use this connection gradient to make sure that they can move without disconnecting themselves and other wandering modules from the structure.

Theorem 1. *The self-reconfigurable robot stays connected provided that a module only moves if:* (1) *the connection gradient is initialized in the module and its neighbours;* (2) *moving it does not change the concentration of the connection gradient in neighbouring modules;* (3) *it is not a source. It is further assumed that, if a module moves, its connection gradient is set to uninitialized.*

The proof is by induction. Basis: given that there is one source, we need to show that the system stays connected. A module in the system can be in four situations.

(1) **The module is the source.** Due to condition 3, it cannot move and the system stays connected.
(2) **A module or some of its neighbours are uninitialized.** This can happen if the gradient has not been propagated yet. Due to condition 1, it cannot move and the system stays connected.
(3) **One of the neighbouring modules' concentration will change if the module moves.** This means that the concentration of one of the neighbours is higher than the module's concentration. This implies that the neighbour's shortest path to a source goes through the module. This, again, implies that the neighbouring module depends on the module for connection. Therefore, and due to condition 2, the module cannot move and therefore the system stays connected.
(4) **None of the neighbour modules' concentration will change if the module moves.** If moving a module does not change the concentration of its neighbouring modules, this implies that they have a concentration that is lower or equal to the one moving. This, again, implies that the neighbouring modules either are closer to the source or have an alternative path to the source. Therefore, the system will stay connected when the module moves.

Induction: we assume that the theorem holds for n sources and want to show that it holds for $n+1$ sources. Adding a source will cause the modules of the system to fall into three cases.

(1) **The module is a source.** This module cannot move due to condition 3,and therefore the system stays connected.
(2) **The concentration from the new source has not been propagated to the module.** This can happen when the concentration from the new source has not yet reached the module. In both cases, the situation is as it was for a system with n modules and, by assumption, we know this case holds.
(3) **The concentration from the new source has been propagated to the module.** In this case the module will, because of condition 2, only move if all neighbouring modules have an alternative path to either the new source or the old sources, and therefore the system will stay connected.

It is possible for more modules to move than this invariant allows, but this invariant represents a trade-off between keeping the system connected and avoiding deadlocks. The distributed nature of the invariant is important, because wandering modules can locally, by inspecting the concentration of the connection gradient of their neighbours, decide if they can move without causing the system to be disconnected. This means that a large fraction of wandering modules can move in any given time step and thus significantly reduce the overall time to reconfigure. Note that Theorem 1 does not hold if sources can be removed. Therefore, finalized modules in our system keep acting as sources for the connection gradient.

5. Simulated system

In our simulation, we use modules that are more powerful than any existing hardware platforms, but do fall within the definition of a Proteo module put forward by Yim et al. [16].

The modules are cubic and, when connected, form a lattice structure. They have six hermaphrodite connectors and can connect to six other modules in the directions: east, west, north, south, up, and down. Modules directly connected to a module are referred to as neighbours. A module can sense whether there are modules in neighbouring lattice cells. In this implementation, we do not control the actuator of the connection mechanism, but assume that neighbouring modules are connected and disconnected appropriately. A module can only communicate with its neighbours. It is able to rotate around neighbours and to slide along the surface of a layer of modules. Finally, we assume that direction vectors can be uniquely transformed from the coordinate system of a module to the coordinate systems of its neighbours. This is necessary to propagate the gradient vector and use the cellular automaton rules.

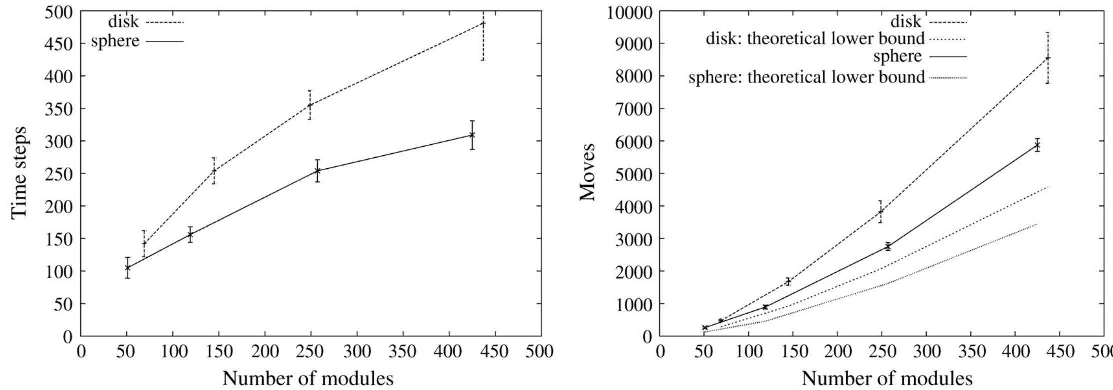

Fig. 3. This figure shows how the number of time steps (left) and moves (right) needed to reconfigure depends on the configuration and the number of modules. The theoretical lower bound on the number of moves is also shown.

The simulator is programmed in Java3D. The simulation uses time steps. In each time step, all the modules are picked in a random sequence and are allowed: (1) to process the messages they have received since the last time step; (2) to send messages to neighbours (but not wait for reply); and (3) to move, if possible. Note that this implies that it can take many time steps for gradients to be propagated in the system.

In order for a module to move, it has to satisfy the motion constraints of the system. This means that, in order to move to a new position, the new position has to be free. If using the rotate primitive, the position that the module has to rotate through to get to the new position has to be free as well. This is all assumed to be sensed locally. A less idealistic module will of course have more motion constraints. However, the working hypothesis is that, by designing a porous sub-structure suited to this specific module, some of the motion constraints may be ignored. This may make it possible to implement a variant of our algorithm.

In our implementation the problem of two modules moving into the same cell at the same time is not addressed because, in the simulation, only one module is allowed to move at a time. However, in a real parallel-running system, this problem has to be addressed.

6. Experiments

We pick two self-reconfiguration tasks taken from [16]. The two tasks are to reconfigure from a rectangular plane to a disk orthogonal to that plane and to a sphere. The rules for these configurations are automatically generated using the cellular automaton generator based on a mathematical description of a sphere and a disk. This automaton is then downloaded into the modules of the simulation and the assembly process is started.

We repeated the experiments twenty times with changing numbers of modules. For each experiment we recorded the time steps needed and the total number of moves, and calculated the theoretical lower bound on the required number of moves. The lower bound is the minimum number of moves needed to reconfigure from the initial configuration to the final configuration, assuming that interference with other modules can be ignored [22]. The time steps needed to reconfigure grow approximately linearly with the number of modules (see Fig. 3), which is comparable to the results reported by Yim et al. [16]. Fig. 3 shows that the number of moves grows more than linearly with the number of modules. However, this is also true for the theoretical lower bound.

In all experiments the system converges to the desired configuration, which supports the claim that the system is convergent by design.

The number of moves and the time to complete a reconfiguration are important characteristics of an algorithm. However, another aspect is the amount of communication needed. This information is summarized in Fig. 4. The system relies heavily on communication, and in one case 1.5 messages are sent on average per module per time step. These messages originate from propagating two changing gradients, propagating state information, and negotiations between neighbours for mutual exclusion and thereby the right to move. We can see that the number of messages depends on the number of modules, and also on the number of moves and time steps. This is indicated by the fact that the reconfiguration into the disk configuration uses significantly more messages than the reconfiguration into the sphere configuration, even though the two configurations contain approximately the same number of modules. In our current implementation, little is done to minimize the number of messages, so there is room to improve the performance. The trade-off is efficiency versus simplicity. If we minimize the number of messages, the complexity of the algorithm will increase. One thing worth noting is that, if the modules do not have to prevent disconnection from the structure, the *attraction* and *state* messages are the only ones needed, reducing the communication load significantly.

7. Conclusion and future work

We have presented the cellular automaton generator which takes as input a 3D CAD model of a desired configuration and outputs a cellular automaton that represents this configuration. A specific sub-structure is enforced on the desired configuration, making it porous. This reduces the

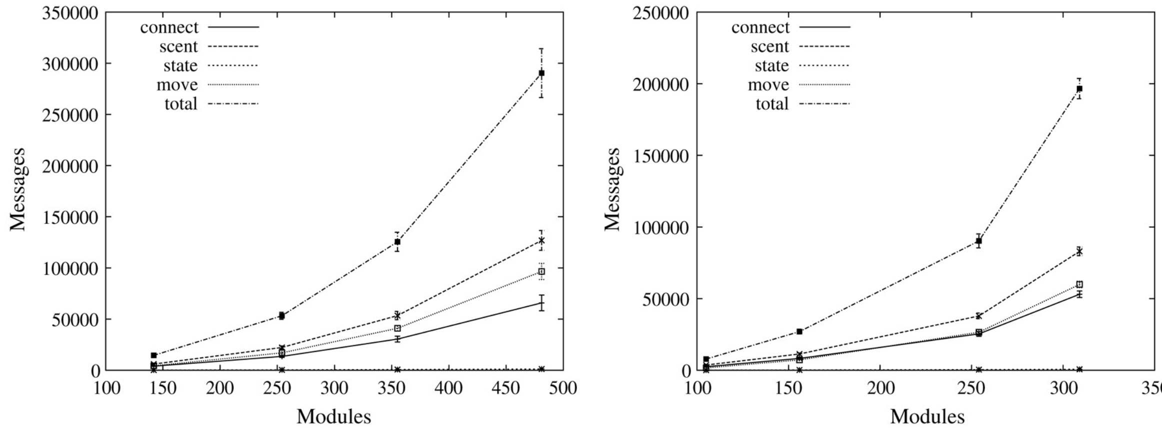

Fig. 4. This figure shows the amount of communication as a function of the number of modules for the disk (left) and sphere (right) configuration. The communication for the attraction and connection gradients is shown. The number of messages needed to keep the system connected, in addition to the messages needed to propagate the connection gradient, is also shown (move). The messages needed to propagate state information are also shown (state). Finally, the total number of messages is shown (total).

complexity of the self-reconfiguration problem, guarantees convergence, and improves the efficiency of the self-reconfiguration algorithm. The efficiency is further improved by using gradients to guide the self-reconfiguration process. Finally, we have presented a way to keep the system connected that is based only on local information and therefore allows for a high degree of parallelism in the system.

Overall, the system represents an interesting and novel solution to the self-reconfiguration problem, based on a combination of existing ideas, which is efficient, systematic, and convergent. However, an important question is to what degree these ideas transfer to a physically realized self-reconfigurable robot, and this will be the focus of our future research.

Acknowledgements

This research is in part funded by the EU contract IST-2001-33060 and the Danish Technical Research Council contract 26-01-0088.

References

[1] G. Chirikjian, A. Pamecha, I. Ebert-Uphoff, Evaluating efficiency of self-reconfiguration in a class of modular robots, Robotics Systems 13 (1996) 317–338.

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

[3] C.-J. Chiang, G. Chirikjian, Modular robot motion planning using similarity metrics, Autonomous Robots 10 (1) (2001) 91–106.

[4] D. Rus, M. Vona, Self-reconfiguration planning with compressible unit modules, in: Proc., IEEE Int. Conf. on Robotics and Automation ICRA'99, Detroit, MI, USA, vol. 4, 1999, pp. 2513–2530.

[5] K. Kotay, D. Rus, Algorithms for self-reconfiguring molecule motion planning, in: Proc., IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS'00, Maui, HI, USA, vol. 3, 2000, pp. 2184–2193.

[6] C. Ünsal, H. Kiliccote, P. Khosla, A modular self-reconfigurable bipartite robotic system: Implementation and motion planning, Autonomous Robots 10 (1) (2001) 23–40.

[7] K. Prevas, C. Unsal, M. Efe, P. Khosla, A hierarchical motion planning strategy for a uniform self-reconfigurable modular robotic system, in: Proc., IEEE Int. Conf. on Robotics and Automation, ICRA'02, Washington, DC, USA, vol. 1, 2002, pp. 787–792.

[8] C. Ünsal, P. Khosla, A multi-layered planner for self-reconfiguration of a uniform group of i-cube modules, in: Proc., IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS'01, Maui, HI, USA, vol. 1, 2001, pp. 598–605.

[9] Z. Butler, S. Byrnes, D. Rus, Distributed motion planning for modular robots with unit-compressible modules, in: Proc., IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS'01, Maui, HI, USA, vol. 2, 2001, pp. 790–796.

[10] S. Vassilvitskii, M. Yim, J. Suh, A complete, local and parallel reconfiguration algorithm for cube style modular robots, in: Proc. IEEE Int. Conf. on Robotics and Automation, ICRA'02, Washington, DC, USA, vol. 1, 2002, pp. 117–122.

[11] S. Murata, H. Kurokawa, S. Kokaji, Self-assembling machine, in: Proc., IEEE Int. Conf. on Robotics & Automation, ICRA'94, San Diego, CA, USA, 1994, pp. 441–448.

[12] E. Yoshida, S. Murata, K. Tomita, H. Kurokawa, S. Kokaji, Distributed formation control of a modular mechanical system, in: Proc., Int. Conf. on Intelligent Robots and Systems, IROS'97, Grenoble, France, vol. 2, 1997, pp. 1090–1097.

[13] E. Yoshida, S. Murata, H. Kurokawa, K. Tomita, S. Kokaji, A distributed reconfiguration method for 3-d homogeneous structure, in: Proc., IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS'98, Victoria, B.C., Canada, vol. 2, 1998, pp. 852–859.

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

[15] M. Yim, J. Lamping, E. Mao, J. Chase, Rhombic dodecahedron shape for self-assembling robots, Tech. rep., Xerox PARC, sPL TechReport P9710777, 1997.

[16] M. Yim, Y. Zhang, J. Lamping, E. Mao, Distributed control for 3d metamorphosis, Autonomous Robots 10 (1) (2001) 41–56.

[17] H. Bojinov, A. Casal, T. Hogg, Emergent structures in modular self-reconfigurable robots, in: Proc., IEEE Int. Conf. on Robotics & Automation, ICRA'00, San Francisco, CA, USA, vol. 2, 2000, pp. 1734–1741.

[18] H. Bojinov, A. Casal, T. Hogg, Multiagent control of self-reconfigurable robots, Proc., Fourth Int. Conf. on MultiAgent Systems, Boston, MA, USA, 2000, pp. 143–150.

[19] C. Jones, M.J. Matarić, From local to global behavior in intelligent self-assembly, in: Proc. of the IEEE Int. Conf. on Robotics and Automation,

ICRA'03, Taipei, Taiwan, 2003, pp. 721–726.

[20] J. von Neumann, Theory of Self-Reproducing Automata, University of Illinois Press, Urbana, IL, USA, 1966 (edited and completed by Authur W. Burks).

[21] Z. Butler, K. Kotay, D. Rus, K. Tomita, Cellular automata for decentralized control of self-reconfigurable robots, in: Proc., IEEE Int. Conf. on Robotics and Automation, ICRA'01, Workshop on Modular Self-Reconfigurable Robots, Seoul, Korea, 2001.

[22] G. Chirikjian, J. Burdick, The kinematics of hyper-redundant robot locomotion, IEEE transactions on robotics and automation 11 (6) (1995) 781–793.

Kasper Stoy is an Assistant Professor at The Maersk Institute for Production Technology, University of Southern Denmark. He received his MSc. in computer science from University of Aarhus, Denmark in 1999. He worked as a research scientist at University of Southern California's Robotics Labs, conducting research on biologically inspired multi-robot coordination. He received his Ph.D. from the University of Southern Denmark in 2003. His research interests include self-reconfigurable robots, embodied artificial intelligence, and biologically inspired multi-robot coordination.