Global Locomotion from Local Interaction in Self-Reconfigurable Robots

K. Støy[1], W.-M. Shen[2] and P. Will[2]

kaspers@mip.sdu.dk {shen,will}@isi.edu

[1]The Maersk Mc-Kinney Moller Institute for Production Technology
University of Southern Denmark, Campusvej 55, DK-5230 Odense M, Denmark
[2]Information Sciences Institute, University of Southern California
4676 Admiralty Way, Marina del Rey, CA 90292, USA

Abstract. We present a general distributed control algorithm for achieving locomotion of a self-reconfigurable robot. In this algorithm each module continuously performs a cyclic sequence of actions with a period T. When a specified fraction of this period d has elapsed a signal is sent to all child modules. Upon receiving this signal the child module resets its action sequence making it delayed d compared to its parent. The algorithm is minimal and robust to loss of synchronization signals and change in the number of modules. We show in three different experiments that the algorithm can be used to implement a caterpillar, a sidewinder, and a rolling wheel gait in a real self-reconfigurable robot consisting of eight modules.

1 Introduction

Reconfigurable robots are robots made from a possibly large number of independent modules connected to form a robot. If the modules from which the reconfigurable robot is built are able to connect and disconnect without human intervention the robot is a self-reconfigurable robot. Examples of physically realized self-reconfigurable robots can be found in [8, 6, 9, 15, 13, 11, 7].

Several potential advantages of self-reconfigurable robots over traditional robots have been pointed out in literature:

- **Versatility**. The modules can be combined in different ways making the same robotic system able to perform a wide range of tasks.
- **Adaptability**. While the self-reconfigurable robot performs its task it can change its physical shape to adapt to changes in the environment.
- **Robustness**. Self-reconfigurable robots consist of many identical modules and therefore if a module breaks down it can be replaced by another.
- **Cheap production**. When the final design for the basic module has been obtained it can be mass produced and thereby keep the cost of the individual module low.

Self-reconfigurable robots can solve the same tasks as traditional robots, but as Yim et al [15] point out; in applications where the task and environment are given a priori it is

often cheaper to build a special purpose robot. Therefore, applications best suited for self-reconfigurable robots are applications where some leverage can be gained from the special abilities of self-reconfigurable robots. The versatility of these robots make them suitable in scenarios where the robots have to handle a range of tasks. The robots can also handle tasks in unknown or dynamic environments, because they are able to adapt to these environments. In tasks where robustness is of importance it might be desirable to use self-reconfigurable robots. Even though real applications for self-reconfigurable robots still are to be seen, a number of applications have been envisioned [11, 15]: fire fighting, search and rescue after an earthquake, battlefield reconnaissance, planetary exploration, undersea mining, and space structure building. Other possible applications include entertainment, service robotics, and payload management.

The potential of self-reconfigurable robots can be realized if several challenges in terms of hardware and software can be met. In this work we focus on one of the challenges in software: how do we make a large number of connected modules perform a coordinated global behavior? Specifically we address how to design algorithms that will make it possible for self-reconfigurable robots to locomote efficiently. In order for a locomotion algorithm to be useful it has to preserve the special properties of these robots. From the advantages and applications mentioned above we can extract a number of guidelines for the design of such a control algorithm. The algorithm should be distributed to avoid having a single point of failure. Also the performance of the algorithm should scale with an increased number of modules. It has to be robust to reconfiguration, because reconfiguration is a fundamental capability of self-reconfigurable robots. Finally, it is desirable to have homogeneous software running on all the modules, because it makes it possible for any module to take over if another one fails.

It is an open question if a top-down or a bottom-up approach gives the best result. We find that it is difficult to design the system at a global level and then later try to distribute it, because often properties of the hardware are ignored and a slow robotic system might be the result. Therefore, we use a bottom-up approach where the single module is the basic unit of design. That is, we move from a global design perspective to a bottom-up one where the important design element is the individual module and its interactions with its neighbors. The global behavior of the system then emerges from the local interaction between individual modules. A similar approach is also used by Bojinov et al [1].

2 Related Work

In the related work presented here we focus on control algorithms for locomotion of self-reconfigurable robots.

Yim et al [14, 15] demonstrate caterpillar like locomotion and a rolling track. Their system is controlled based on a gait control table . Each column in this table represents the actions performed by one module. Motion is then obtain by having a master synchronizing the transition from one row to the next. The problem with this approach is that the amount of communication needed between the master and the modules will limit its scalability. Another problem is the need for a central controller, since it gives the system a single point of failure. If there is no master it is suggested that the modules can be assumed to be synchronized in time and each module can execute its column of actions open-loop. However, since all the modules are autonomous it is a questionable assumption to assume that all the modules are and can stay synchronized. In order to use the gait control table each module needs to know what column it has to execute. This means that the modules need IDs. Furthermore, if the configuration changes or the number of modules changes the table has to be rewritten.

Shen, Salemi, and others propose to use artificial hormones to synchronize the modules to achieve consistent global locomotion. In earlier versions of the system a hormone is propagated through the self-reconfigurable system to achieve synchronization [11]. In later work the hormone is also propagated backwards making all modules synchronized before a new action is initiated [12, 10]. This synchronization takes time $O(n)$ where n is the number of modules. This slows down the system considerably, because it has to be done before each action. Also, the entire system stops working if one hormone is lost. This a significant problem, because a hormone can easily be lost due to unreliable communication, a module disconnecting itself before a response can be given, or a module failure. In fact, the system has n-points of failure which is not desirable. The earlier version is better in this sense, but still performance remains low because a synchronization hormone is sent before each action.

In our system all modules repeatedly go through a cyclic sequence of joint angles describing a motion. This sequence could come from a column in a gait control table, but in our implementation the joint angles are calculated using a cyclic function with period T. Every time a module has completed a given fraction d of the period a message is sent through the child connectors. If the signal is received the child module resets its action sequence making it delayed d compared to the parent. This way the actions of the individual module are decoupled from the synchronization mechanism resulting in a faster and more reliable system. Furthermore, there is no need to make changes to the algorithm if the number of modules changes.

3 General Control Algorithm

We assume that the modules are connected to form a tree structure, that a parent connector is specified, and that this connector is the only one that can connect to child connectors of other modules. Furthermore, we assume that the modules can communicate with the modules to which they are connected.

The algorithm is then used by specifying three components. The first component is a cyclic action sequence $A(t)$. This sequence describes the actions that each module is to repeat cycle after cycle. The second, is the period T of this cycle. The third, is a delay d. This delay specify the fraction of a period the children's action sequences are delayed compared to their parents. The skeleton algorithm looks like this:

```
t = 0
while(1) {
  if (t=d) then <send signal to child connectors>
  if <signal received from parent> then t=0
  <perform action A(t)>
  t = (t+1) modulus T
}
```

Ignoring the first two lines of the loop, the module repeatedly goes through a sequence of actions parameterized by the cyclic counter t. This part of the algorithm alone can make a single module repeatedly perform the specified sequence of actions. In order to coordinate the actions of the individual modules to produce the desired global behavior the modules need to be synchronized. Therefore, at step $t = d$ a signal is send through all child connectors. Note that it does not matter if a child module is actually connected or not. If a child receives a signal it knows that the parent is at $t = d$ and therefore sets its own step counter to $t = 0$. This enforces that the child is delayed d compared to its parent.

From the time the modules are connected it takes time proportional to d times the height of the tree for all the modules to synchronize. To avoid problems with uncoordinated modules initially we make sure the modules do not start moving until they receive the first synchronization signal. After the start-up phase the modules stay synchronized using only constant time.

4 Experimental Setup

To evaluate our algorithm we conducted several experiments using the CONRO modules shown in Figure 1. The CONRO modules have been developed at USC/ISI [3, 5]. The modules are roughly shaped as rectangular boxes measuring 10cm x 4.5cm x 4.5cm and weigh 100grams. The modules have a female connector at one end and three male connectors located at the other. Each connector has a infra-red transmitter and receiver used for local communication and sensing. The modules have two controllable degrees of freedom: pitch (up and down) and yaw (side to side). Processing is taken care of by an onboard Basic Stamp 2 processor. The modules have onboard batteries, but these do not supply enough power for the experiments reported here and therefore the modules are powered through cables. Refer to http://www.isi.edu/conro for more details and videos of the experiments reported later.

5 Experiments

In general, it is a problem how to report performance of a specific part of a self-reconfigurable system because there is such a tight coupling between hardware and software. In this work we choose to report the length of our programs as a measure of the complexity of the control algorithm. This metric is used to support our claim that this control system is minimal. We also report the speed of the locomotion patterns, but this should only be considered an example, the reason being that in our system the limiting factors are how robust the modules physically are, how powerful the motors are, and how much power we can pull from the power source. To report a top speed is not meaningful before we run the robot autonomously on batteries.

5.1 Caterpillar Locomotion

We connect eight of our modules in a chain and designate the male opposite the female connector to be the parent connector. We then implement the algorithm described above with the following parameters.

$$\begin{aligned} T &= 180 \\ pitch(t) &= 50^\circ sin(\tfrac{2\pi}{T}t) \\ yaw(t) &= 0 \\ d &= \tfrac{T}{5} \end{aligned} \tag{1}$$

The motor control of our modules makes the motor go to the desired position as fast as possible. This means that way-points have to be specified to avoid jerky motion. The period T can be used to control the number of way-points and therefore the smoothness and speed of the motion. The action sequence is an oscillation around 0° with an amplitude of 50° and the yaw joint is kept straight. Each module is delayed one fifth of a period compared to its parent.

The modules are connected and after they synchronize a sine wave is traveling along the length of the robot. Refer to Figure 1. This produces caterpillar like locomotion at a speed

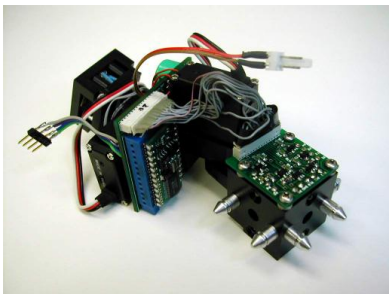
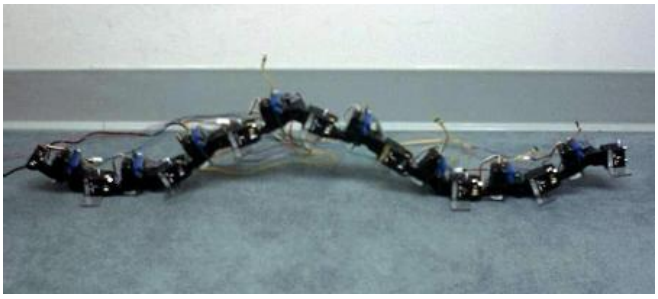

Figure 1: A CONRO module (left). A snapshot of caterpillar like locomotion (right).

of 0.13km/h. Note, that it is easy to adjust the parameters of this motion. For instance, the length of the wave can be controlled using the delay. The program is simple. The main loop contains 16 lines of code excluding comments and labels. The initialization including variable and constant declaration amounts to 18 lines of code.

5.2 Sidewinder Locomotion

We now turn our attention to a locomotion pattern similar to that of a sidewinding snake. A detailed mathematical analysis of this motion pattern has been reported in [2]. Here we just use the intuition that by having modules moving to one side lifted and those moving to the other touching the ground a sidewinder like motion is achieved. The result can be seen in Figure 2. The sidewinder moves at 0.24km/h. The main loop and the initialization contain respectively 19 and 17 lines of code. The parameters used are:

$$\begin{aligned} T &= 180 \\ pitch(t) &= 20^\circ cos(\tfrac{2\pi}{T}t) \\ yaw(t) &= 50^\circ sin(\tfrac{2\pi}{T}t) \\ d &= \tfrac{T}{5} \end{aligned} \tag{2}$$

5.3 Rolling Track Locomotion

If we maintain that each module can only have one parent, but remove the assumption that the structure forms a tree we include loops as structures that can be handled. The rolling track is an example of such a configuration. However, this poses a problem to our algorithm. In the previous experiments we have exploited the assumption that the modules form a tree to implicitly find a conductor. The conductor being the root of the configuration tree. This is a simple mechanism that guarantees that there is one and only one conductor. In a loop configuration this is not the case.

One solution to this problem is to introduce IDs. In our implementation we just make the modules pick a random number and use that as ID. It is not guaranteed to find a unique conductor, but it is a simple solution that works in most cases. The short comings of this approach can easily be avoided if each module has a unique serial number.

The synchronization part of the algorithm now works as before, but it is combined with a simple well-known distributed leader election algorithm [4]. The signals from parent to child now contains a number which is the ID of the module originally sending the signal. Upon receiving a signal a module compares the signal's number to its ID. If it is higher the

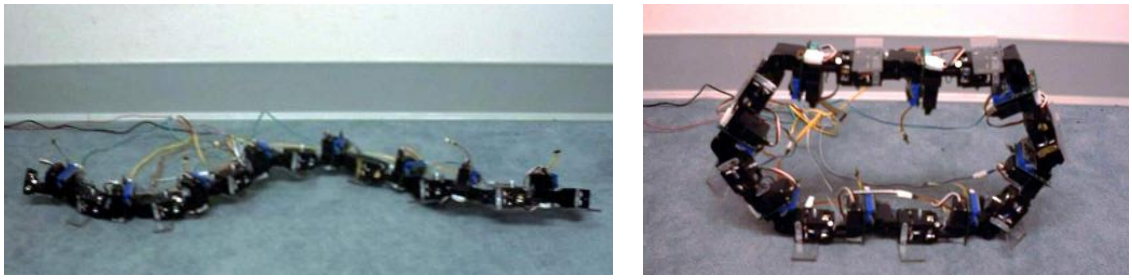

Figure 2: A snapshot of sidewinder like locomotion (left) and the rolling track (right).

module is synchronized and the signal and its ID is propagated along with the synchronization signal. Otherwise, the module consider itself the conductor and ignores the signal. After the system has settled the module with the highest ID dictates the rhythm of the locomotion pattern. The leader election algorithm runs continuously which means that the system quickly synchronizes if modules are replaced. The advantage of combining the algorithms is that there is no need to detect if the conductor fails.

We used this algorithm to implemented the rolling track which can be seen in Figure 2. The rolling track is the fastest gait and achieves a speed of 0.50km/h. The program is now a little more complex and the main loop and initialization contain respectively 35 and 28 lines of code. The parameters for the eight module rolling track is:

$$\begin{aligned} T &= 180 \\ pitch(t) &= \begin{cases} 60^\circ(1 - sin(\frac{2\pi}{T}t)) & if\ t =< \frac{T}{2} \\ 60^\circ & if\ t > \frac{T}{2} \end{cases} \\ yaw(t) &= 0 \\ d &= \frac{T}{4} \end{aligned} \tag{3}$$

Unlike the sidewinder and the caterpillar this control algorithm only works with 8 modules, because of the physical constraint. It might be possible to make a more general solution by making $pitch(t)$ and d a function of the number of modules. The number of modules in the loop could be obtained by the conductor by including a hop count in the signal.

6 Handling a General Configuration

We saw in the previous section that we had to introduce IDs to find a unique conductor in a configuration that contains loops. Introducing the ID mechanism unfortunately ruins the opportunity to use the synchronization algorithm to automatically find a conductor in a tree structure. In fact, the loop algorithm will fail in this situation unless the module with the highest ID also happens to be the root. In order to make a general algorithm the synchronization signal has to be propagated both upwards and downwards in the tree.

7 Discussion

We have presented a general control algorithm and presented examples of how it can be used to achieve three different locomotion patterns. We will now discus some of the properties of this control algorithm.

An important issue in the design of control algorithms for self-reconfigurable robots is if the algorithm scales with the number of modules. The presented algorithm is only initially

dependent on the number of modules, because it decides how long time it takes for the synchronization signal to be propagated through the system. After this start-up phase the time it takes to keep the modules synchronized is independent of the number of modules implying that the algorithm scales. Furthermore, all modules run identical programs making it easier to manage the development process when programming systems consisting of many modules.

The modules of the robot are only loosely coupled through the synchronization signal and therefore the system is highly robust to changes in the number of modules. In fact, the caterpillar can be divided in two and both parts still work. If they are reconnected in a different order they will quickly synchronize to behave as one long caterpillar again. This also implies that the system is robust to module failure. If a module is defect and it can be detected this module can be ejected from the system and the remaining modules when reconnected can continue to perform. Finally, if a synchronization signal is lost it is not crucial for the survival of the system. If a signal is lost it just means that the receiving module and its children will be synchronized a period later.

In the algorithm the synchronization signal is only sent once per period. This means that in order for the modules to stay synchronized the time to complete a period has to be the same for all modules.In the experiments presented here the cycles take the same amount of time, but in more complex control systems where other parts of the control system use random amounts of computation time this can not be assumed to be true. This problem can easily be handle by using timers. Even though timers are not precise enough to keep modules synchronized over a long period of time they can be used for this purpose.

8 Future Work

Our future work will go along two lines. Can the algorithm handle more complex locomotion patterns? We suspect it can be achieved by using different delays through different connectors. For instance, in a multi-legged robot where the head is the conductor, the synchronization signal could travel along the spine modules. When a spine module receives a signal it can first propagate it to the left leg and then the right before propagating the signal to the next spine module.

Another issue is that if the self-reconfigurable robot is to locomote automatically in a real complex environment the control algorithm has to be able to take feedback from the environment into account. A first step in this direction could be to mount sensors on the side of the caterpillar robot and use these to control the yaw joint of the modules.

9 Conclusion

We have presented a general control algorithm for self-reconfigurable robots. The algorithm has the following properties: distributed, scalable, homogeneous, and minimal. We have shown how the algorithm easily can be used to implement a caterpillar and a sidewinder like locomotion pattern. Furthermore, we have seen that with the introduction of IDs in the modules it is possible to handle loop configurations. We have demonstrated this using the rolling track as an example. Finally, we have pointed out interesting lines for future research.

10 Acknowledgements

This work is supported under the DARPA contract DAAN02-98-C-4032, the EU contract IST-20001-33060, and the Danish Technical Research Council contract 26-01-0088.

References

[1] H. Bojinov, A. Casal, and T. Hogg. Emergent structures in modular self-reconfigurable robots. In *Proceedings of the IEEE int. conf. on Robotics & Automation*, volume 2, pages 1734 –1741, San Francisco, USA, 2000.

[2] J.W. Burdick, J. Radford, and G.S. Chirikjian. A 'sidewinding' locomotion gait for hyper-redundant robots. In *Proceedings of the IEEE int. conf. on Robotics & Automation*, pages 101–106, Atlanta, USA, 1993.

[3] A. Castano, R. Chokkalingam, and P. Will. Autonomous and self-sufficient conro modules for reconfigurable robots. In *Proceedings of the 5th int. Symposion on Distributed Autonomous Robotic Systems*, pages 155–164, Knoxville, Texas, USA, 2000.

[4] E. Chang and R. Roberts. An improved algorithm for decentralized extrema-finding in circular configurations of processes. *Communications of the ACM*, 22(5):281–283, 1979.

[5] B. Khoshnevis, B. Kovac, W.-M. Shen, and P. Will. Reconnectable joints for self-reconfigurable robots. In *Proceedings of the IEEE/RSJ int. conf. on Intelligent Robots and Systems*, Maui, Hawaii, USA, 2001.

[6] K. Kotay, D. Rus, M. Vona, and C. McGray. The self-reconfiguring robotic molecule. In *Proceedings of the IEEE int. conf. on Robotics & Automation*, pages 424–431, Leuven, Belgium, 1998.

[7] S. Murata, E. Yoshida, H. Kurokawa, K. Tomita, and S. Kokaji. Self-repairing mechanical systems. *Autonomous Robots*, 10(1):7–21, 2001.

[8] A. Pamecha, C. Chiang, D. Stein, and G.S. Chirikjian. Design and implementation of metamorphic robots. In *Proceedings of the ASME Design Engineering Technical conf. and Computers in Engineering conf.*, pages 1–10, Irvine, USA, 1996.

[9] D. Rus and M. Vona. A physical implementation of the crystalline robot. In *Proceedings of the IEEE int. conf. on Robotics & Automation*, pages 1726–1733, San Francisco, USA, 2000.

[10] B. Salemi, W. Shen, and P. Will. Hormone controlled metamorphic robots. In *Proceedings of the IEEE int. conf. on Robotics & Automation*, pages 4194–4199, Seoul, Korea, 2001.

[11] W.-M. Shen, B. Salemi, and P. Will. Hormone-based control for self-reconfigurable robots. In *Proceedings of the int. conf. on Autonomous Agents*, pages 1–8, Barcelona, Spain, 2000.

[12] W.-M. Shen, B. Salemi, and P. Will. Hormones for self-reconfigurable robots. In *Proceedings of the int. conf. on Intelligent Autonomous Systems*, pages 918–925, Venice, Italy, 2000.

[13] C. Ünsal and P.K. Khosla. Mechatronic design of a modular self-reconfiguring robotic system. In *Proceedings of the IEEE int. conf. on Robotics & Automation*, pages 1742–1747, San Francisco, USA, 2000.

[14] M. Yim. *Locomotion with a unit-modular reconfigurable robot*. PhD thesis, Department of Mechanical Engineering, Stanford University, 1994.

[15] M. Yim, D.G. Duff, and K.D. Roufas. Polybot: A modular reconfigurable robot. In *Proceedings of the IEEE int. conf. on Robotics & Automation*, pages 514–520, San Francisco, USA, 2000.