How to Make a Self-Reconfigurable Robot Run

K. Støy
The Maersk Mc-Kinney Moller Institute for Production Technology
Campusvej 55
DK-5230 Odense M, Denmark
kaspers@mip.sdu.dk

W.-M. Shen
Information Sciences Institute and Computer Science Department
4676 Admiralty way
Marina del Rey, CA 90292, USA
shen@isi.edu

P. Will
Information Sciences Institute and Computer Science Department
4676 Admiralty way
Marina del Rey, CA 90292, USA
will@isi.edu

ABSTRACT

In this paper we present role based control which is a multi-agent based control algorithm for self-reconfigurable robots. We use role based control to implement quadruped and hexapod gaits in a real self-reconfigurable robot made from up to nine independent autonomous modules. We show that this implementation scales and argue that it is minimal, robust to module failures, to loss of communication signals, and to interchange of modules.

In role based control all modules of the robot run identical programs, but may play different *roles*. The modules decide what role to play based on their local configuration and information propagated down to them through the configuration tree. A role consists of a cyclic motion, the period of this motion, and a set of delays. The delays specify the phase delay of the cyclic motions of the child modules compared to the parent. These delays are used to coordinate the motions of the individual module to obtain a coordinated global behavior.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems, coherence and coordination*; I.2.9 [**Artificial Intelligence**]: Robotics—*Autonomous vehicles*

General Terms

Algorithms, Experimentation

1. INTRODUCTION

Reconfigurable robots are robots made from a possibly large number of independent modules that are 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.

An example of a self-reconfigurable robot is the CONRO robot. A module of the CONRO system is shown in Figure 1. Examples of other physically realized self-reconfigurable robots can be found in [4, 6, 7, 8, 9, 10, 11, 13, 17, 19, 20].

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 [8, 15].
- **Adaptability**. While the self-reconfigurable robot performs its task it can change its physical shape to adapt to changes in the environment [13].
- **Robustness**. Self-reconfigurable robots are made from many identical modules and therefore if a module fails it can be replaced by another [7, 19, 15].
- **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 compared to its complexity. [7, 8, 19].

Self-reconfigurable robots can solve the same tasks as traditional robots, but as Yim et al [19] point out, in applications where the task and environment are given a priori it is often cheaper to build a special purpose robot. Therefore, the 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 environment. 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 specific applications have been envisioned [13, 8, 19]: fire fighting, search and rescue after an earthquake, battlefield reconnaissance, planetary exploration, undersea mining, and space structure building. Other possible applications include entertainment and service robotics.

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. 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. This way each module plays the role of an agent in a multiagent system. A similar approach is also used by Bojinov et al [1, 2].

2. RELATED WORK

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

Yim et al [18, 19] demonstrate among other types of locomotion a four-legged spider like type of locomotion. In their system each module has a gait control table where each column 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 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 gait control table has to be rewritten.

Shen and Salemi 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 [13]. In later work the hormone is also propagated backwards making all modules synchronized before a new action is initiated [14, 12]. 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 can easily happen 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.

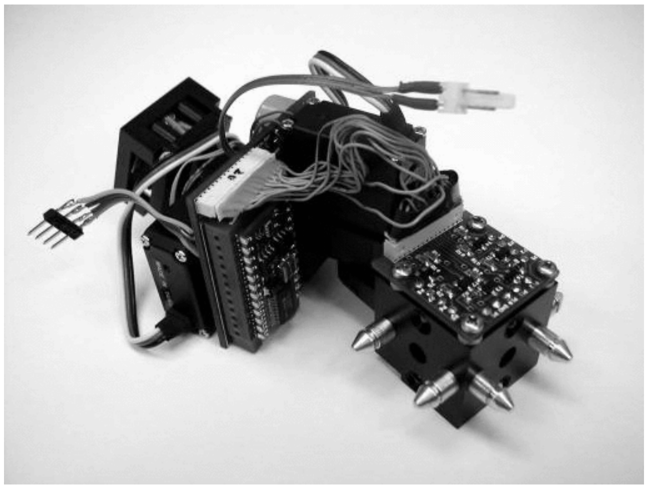

Figure 1: A CONRO module.

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. Every time a module reaches a specified position p in the cycle a message is sent through a specified child connector. If the signal is received the child module resets its position in its cycle making it delayed p 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. ROLE BASED CONTROL

In our approach we acknowledge the need for each module to be autonomous in order to obtain a robust and scalable system. We also acknowledge the need for a tight coupling between the modules to coordinate and produce the desired global behavior. However it is not desirable to coordinate at the level of the individual motor control command, because involving other modules in low level control will produce a less responsive system. Therefore, we abstract away the low level control and coordinate at a higher level. We coordinate at the level of *roles*. In the following we will define what a role is. We will describe the algorithm a module uses to play a role, and finally we will discus how modules can decide to change roles over time.

3.1 A Role

A role consists of three components. The first component is a function $A(t)$ that specifies the joint angles of a module given an integer $t \in [0 : T]$, where T is the period of the motion and the second component that needs to be specified. The third component is a set of delays D. A delay $d_i \in D$ specifies the delay between the child connected to connector i and the parent. That is, if the parent is at step $t_{parent} = t_1$ the child is at $t_{child} = (t_1 - d_i + T)\ modulus\ T$.

```
t = 0
while(1)
  if (t=d_1) then <send signal to child connector 1>
  ...
  if (t=d_n) then <send signal to child connector n>

  if <signal received from parent connector> then
    t=0
  endif

  <perform action A(t)>
  t = (t+1) modulus T
endwhile
```

Figure 2: The algorithm used to play a role. Refer to section 3.2 for further explanation.

3.2 Playing a Role

The algorithm that we are now about to describe is used to make a module play a role. However first some assumptions need to be made: a parent connector is specified and the remaining connectors are considered child connectors, connections can only be made between a parent connector and a child connector. These assumptions limit the configuration the algorithm can handle to tree configurations. Under these assumptions a role is played using the algorithm shown in Figure 2.

Ignoring the if-statements in the beginning of the loop, the module repeatedly goes through a sequence of actions parameterized by t. This part of the algorithm alone makes a single module repeatedly perform the sequence of actions specified by $A(t)$. However, in order to achieve coherent global behavior the module needs to be synchronized with neighbors. Therefore, at $t_{parent} = d_i$ a signal is sent through child connector i. Note that it does not matter to the parent if a child module actually receives the signal, because in that case the signal will just be lost. However, if child i receives the signal it sets $t_{child-i} = 0$. This enforces that the child is delayed d_i compared to the parent.

A simple rule that will come in handy when we calculate the delays for a specific locomotion pattern is that we can calculate what step a child $t_{child-i}$ is at based on what step the parent t_{parent} is at:

$$t_{parent} = t_1 \quad \Rightarrow \quad t_{child-i} = t_1 - d_i \tag{1}$$

The other way:

$$t_{child} = t_1 \quad \Rightarrow \quad t_{parent} = t_1 + d_i \tag{2}$$

In terms of execution time we can see that from the time the modules are connected, it takes time proportional to the height of the tree for all the modules to synchronize. However, once the modules are synchronized, the algorithm keeps the modules synchronized using only constant time. Below is an example of a caterpillar role.

$$A(t) = \begin{cases} pitch(t) = 50^\circ \sin(\frac{2\pi}{T}t) \\ yaw(t) = 0 \end{cases} \tag{3}$$

$$d_{north} = \frac{T}{5} \tag{4}$$

$$T = 180 \tag{5}$$

```
r = <start role>
t = 0
while(1)
  if (t=d(r)_1) then
    <send message M(r,1) to child connector 1>
    <update r>
  endif
  ...
  if (t=d(r)_n) then
    <send message M(r,n) to child connector n>
    <update r>
  endif

  if <message m received from parent connector> then
    t=0
    <update r based on m>
  endif

  <perform action A(r,t)>
  t = (t+1) modulus T(r)
endwhile
```

Figure 3: The algorithm used to enable modules to play different roles depending on their position in the configuration tree. Refer to section 3.3 for further explanation.

In earlier work we have demonstrated that when the modules are connected in a chain and all play this role they produce caterpillar like locomotion [15]. In this work we also demonstrated a locomotion pattern similar to that of a sidewinder snake.

3.3 Combining Roles

In simple locomotion gaits only one role is needed to obtain the desired global behavior. However, in more complex locomotion patterns more roles will be needed. For instance, in a walking robot the following roles can be identified: left leg, right leg, and spine. In order to handle these more complex locomotion patterns we need to extend our algorithm.

It is obvious that we have to define each of the roles needed to produce the desired global behavior. For each role r we supply an action sequence $A(r,t)$, a period $T(r)$, and a set of delays $D(r)$. We also need to define how a module decides when to change role.

A module can change its role in two ways. One way is as a reaction to changes in the local configuration. The other way is in response to a message from the parent. We encode the information needed to make these decisions into the communication signals that are already part of the synchronization mechanism. It does not make sense to implement it as a separate communication mechanism for two reasons. 1) It is of no value to the module to know what role to play before it is synchronized. 2) It adds more complexity to the system. The resulting algorithm is shown in Figure 3.

The algorithm looks very similar to the basic algorithm. The main difference is that now the synchronization signal contains a message that is a function $M(r,i)$ of the role r the module is playing and the connector i the message is sent through. This function can be used to uniquely specify what role each module in the system should play. The other difference is that based on the success of communication the module can detect its local configuration and use this information to update its role appropriately.

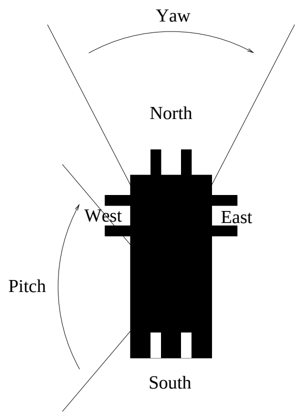

Figure 4: A schematic overview of the CONRO module. The connectors are labeled with compass directions. The arrows indicate the direction of increasing angle.

4. THE CONRO MODULES

For our experiments we use a self-reconfigurable robot made from the CONRO modules developed at University of Southern California's Information Sciences Institute [3, 5] (see figure 1). 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 located at one end facing south and three male connectors located at the other end facing east, west, and north (see Figure 4). Each connector has an 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 for videos of the experiments reported later in this paper.

5. IMPLEMENTING A WALKING GAIT

Our goal is to implement a walking gait in the CONRO self-reconfigurable robot configured as shown in Figure 6. In order to do so we need to define three different roles: spine, east leg, and west leg. The two main tasks are to specify the action sequences and the delays for each role. We will look at these two problems below.

5.1 Actions

We start out by specifying the actions for each role. Intuitively the legs should be lifted from the ground when moving forward and touching the ground when moving backwards. We use the following motion equation for the east legs:

$$A(east\,leg, t) \quad = \quad \begin{cases} pitch(t) & = & 35^\circ \cos(\frac{2\pi}{T}t) - 55^\circ \\ yaw(t) & = & 40^\circ \sin(\frac{2\pi}{T}t) \end{cases} \qquad (6)$$

The equation for the west legs is obtained by replacing t by $2\pi - t$ giving the same motion, but in the opposite direction. This motion is visualized in Figure 5.

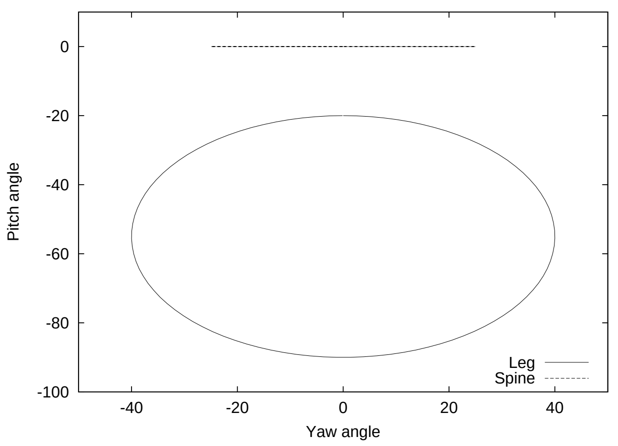

Figure 5: The motion of a module playing the leg role (top) and the spine role (bottom) visualized in joint space.

The spine module between two pairs of legs should bend from side to side to increase the length of each step. The parameters for this motion are shown below and are also visualized in Figure 5.

$$A(spine, t) \quad = \quad \begin{cases} pitch(t) & = & 0^\circ \\ yaw(t) & = & 25^\circ \cos(\frac{2\pi}{T}t + \pi) \end{cases} \qquad (7)$$

For simplicity we pick the same period T for all roles. The parameter T can later be used to control the locomotion speed.

5.2 Delays

What is left is to coordinate the motion of the modules. The question is how do we get from a general description of which modules should be coordinated to the delays needed in our algorithm. In a walking robot the left front leg and the rear right should be synchronized. The same goes for the right front leg and the rear left leg. Also, it would result in a more efficient locomotion pattern if the spine modules bend to increase the length of each step.

We configure our modules as shown in Figure 6. We first consider the spine module located in the top middle part of the figure labeled spine-1. The question is now what fraction of a period each child should be delayed. We first turn our attention to the front legs: east-1 and west-1. It is obvious that the motion of these legs should be half a period apart. This way one leg will touch the ground when the other is lifted and the other way around. Therefore, the delay between the two legs should be $T/2$ ($+nT$ where $n \in [0 : \infty]$ which we consistently omit from these calculations). Given t_{east-1} and t_{west-1} we use equation 2 to calculate what t this corresponds to in the spine-1 module.

$$\begin{aligned} t_{spine-1} &= t_{east-1} + d_{east} \\ t_{spine-1} &= t_{west-1} + d_{west} \end{aligned} \qquad (8)$$

Setting the two expressions for $t_{spine-1}$ equal to each other we get:

$$t_{east-1} + d_{east} \quad = \quad t_{west-1} + d_{west} \qquad (9)$$

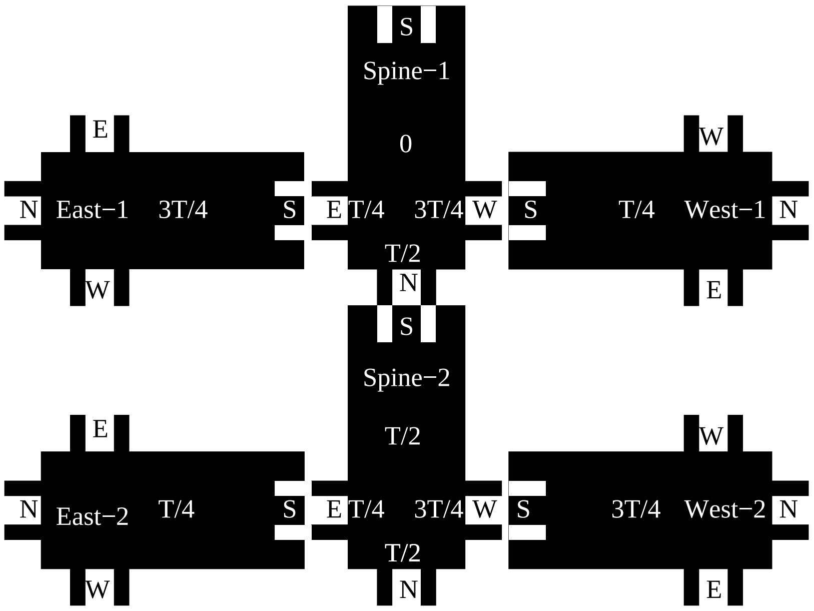

Figure 6: The black boxes represent modules. The modules are connected to form a quadruped robot. All the modules are named and labeled with compass directions. The expression next to a connector represents the delay d across that connector. The delay is expressed in fractions of a period T. For instance, the delay associated with the east connector of the spine module is $d_{east} = T/4$. The expressions located in the center of the modules represent the value of t of that module when the spine module spine-1 is at $t_{spine-1} = 0$. The t value of child is calculated using $t_{child-i} = t_{parent} - d_i$ where i is the connector to which the child is connected. Note that by using the delays shown here the top east leg and bottom west legs are synchronized. This also goes for the top west and bottom east leg.

We exploit that we want the difference between t_{east-1} and t_{west-1} to be $T/2$ to find the following constraint:

$$d_{east} - d_{west} = \tfrac{T}{2} \tag{10}$$

A similar consideration leads to the conclusion that the front west leg and the rear east leg should be synchronized. However the signal to the rear east leg is delayed when it goes though the north connector as well. Given t_{east-2} and t_{west-1} we again using rule 2 to transform into $t_{spine-1}$.

$$t_{spine-1} = t_{west-1} + d_{west} \tag{11}$$

$$t_{spine-1} = t_{east-2} + d_{east} + d_{north} \tag{12}$$

Setting the two expressions for $t_{spine-1}$ equal to each other we get:

$$t_{west-1} + d_{west} = t_{east-2} + d_{east} + d_{north} \tag{13}$$

We exploit that we want $t_{west-1} - t_{east-2} = 0$ and equation 10 to obtain:

$$d_{north} = \frac{T}{2} \tag{14}$$

Using our knowledge about how the legs should be synchronized we have come up with constraints for the delays. We know that a module playing the spine role receives a message from the parent at $t = 0$ and has to send one through the north connector at $t = T/2$. In order to spread out the communication over a period we pick the following values for d_{east} and d_{west} that satisfies constraint 10.

$$d_{east} = \frac{T}{4} \tag{15}$$

$$d_{west} = \frac{3T}{4} \tag{16}$$

Now all the delays are specified, but there is one piece missing. We have to make sure that the spine also is coordinated. We know from the spine motion defined in equation 7 that the spine is bent most to the east at $t_{spine-2} = T/2$. The east legs are closest together at $t_{east-2} = T/4$ (see equation 6). We want the difference between $t_{spine-2}$ and t_{east-2} to be zero. Again applying equation 2 we transform t_{east-2} into $t_{spine-2}$:

$$t_{spine-2} = \frac{T}{4} + t_{east-2} \tag{17}$$

$$t_{spine-2} = T/2 \tag{18}$$

Setting the two equations equal to each other we in fact see that our choice for t_{east} makes sure that the spine is indeed making the robot take longer steps. These results are summarized in Figure 6.

We have now calculated the delays for the spine module. If feet-modules were connected to the legs we would have to calculate delays for the legs as well. However, we are not planning to connect any modules so for simplicity we just give the leg modules the same delays as the spine modules.

5.3 Role selection

Now all the modules are synchronized and can play the role of a spine, a east leg, or west leg. The question is now how each module decides what part it plays. We have to define the function $M(r, i)$ that maps a role r and a connector i to a message. In our simple configuration we define M as:

$$M(r, i) = i, \quad where\, i \in \{south, east, west\} \tag{19}$$

This intuitively means that a message contains information about which connector it was sent through. Upon receiving a message m, the role selection is straight forward:

$$r(m) \;=\; \begin{cases} west\,leg, & if \quad m = west \\ east\,leg, & if \quad m = east \\ spine, & if \quad m = south \end{cases} \tag{20}$$

These role selection rules correctly assign the roles to modules, even if the modules are interchanged. There is one notable exception. The root of the tree never receives any messages, because it by definition does not have a parent. Therefore we need to introduce rules that based on the local configuration can detect if a module is the root and therefore should play the spine role. In the quadruped configuration this is easy. We simply say that if the module successfully communicate with a child connected either to the west or east connector, the module changes its role to a spine role (if it is not already a spine module).

The role selection rules are arbitrary since many role selection rules and functions M exist that would lead to the same behavior. In systems where there are more roles more effort has to be put into defining rules to make sure the rules are not ambiguous.

6. EXPERIMENTS

In general, it is problematic to report performance of a self-reconfigurable system, because there is such a tight coupling between hardware and software. In this work we report scalability of the algorithm. Furthermore we report the length of our programs as a measure of the complexity of the control algorithm. We also report the speed of the walking gaits, 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 the robot runs autonomously on batteries.

6.1 Quadruped Locomotion

In the first experiment we assembled the modules in the quadruped configuration shown in Figure 7. We then measured the time it took for the robot to walk a distance of 150cm. We found that the average of ten trials was 10.9seconds and the standard deviation was 0.57seconds. This corresponds to a speed of 13.8cm/second.

The main loop of the program excluding comments and labels takes up 120 lines of code. The initialization part contains 32 lines of code. The small size of the program emphasizes the point that the control algorithm is simple and minimal. In this system we can replace modules or move them around as desired, because they will pick the right role and synchronize correctly no matter where they are placed in the configuration.

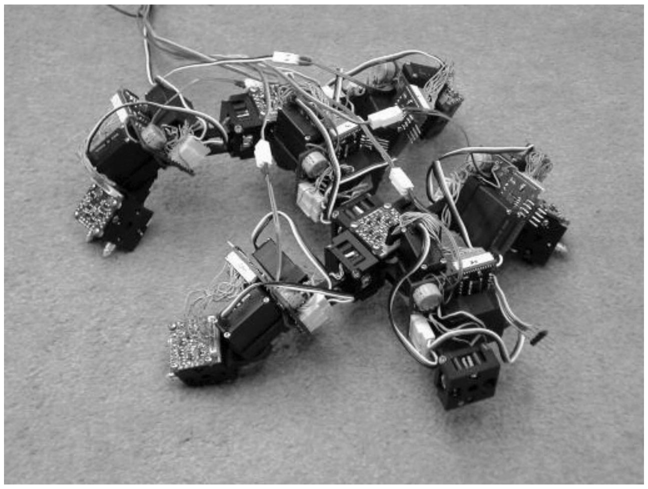

Figure 7: The robot performing quadruped locomotion. The wires connected to each module only provide power.

6.2 Hexapod Locomotion

We extended the quadruped with an extra pair of leg and a spine module to obtain a hexapod robot. This configuration can be seen in Figure 8. Note that the controllers of the module do not need to be changed, because the delays make sure that the third pair of legs is appropriately delayed. We repeated the experiments another ten times and found that the average time was 12.0seconds (12.5cm/sec.) and the standard deviation was 0.57seconds.

Initially we tested the hypothesis that the speed of the robot is independent of the number of modules. Unfortunately Student's t-test rejected this hypothesis (test probability $3.0 * 10^{-7}$). From close observation of the experiments we found that the quadruped robot makes longer step, because it slides a little forward with each step due to its momentum. In the hexapod this is not the case, because of the friction caused by the extra pair of legs. In order to remove this difference from our data we returned to the videos of the experiments and counted the number of steps taken by the robot in each experiment. We divided the time with the number of steps to produce a time per step measure. We then tested the hypothesis that the time per step is the same for both the quadruped and hexapod walker. This hypothesis was accepted on the 5% confidence level with test probability 0.78. This implies that the speed of the system does not dependent on the number of modules and therefore the algorithm scales. If we had more modules available we could extend the robot to make a 2n-legged walker. The algorithm can handle this because after the initial synchronization it only takes constant time per period to keep the modules synchronized.

7. DISCUSSION

The system achieves a high level of performance because we use a less aggressive synchronization mechanism. Intuitively the idea is that as long as each module keeps its children synchronized each period then all the modules will become and stay synchronized over time. This idea only works if it can be assumed that one period of motion takes

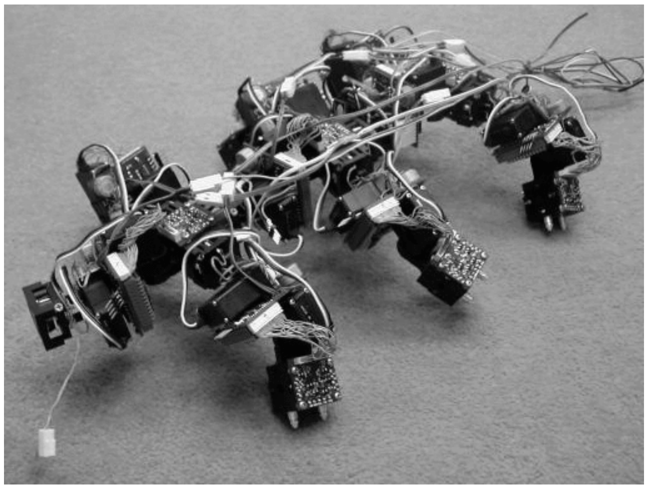

Figure 8: The robot performing hexapod locomotion. The wires connected to each module only provide power.

the same amount of time for all modules otherwise the modules will keep getting out of synchronization. To insure that the periods take the same amount of time is not a problem in our simple system, but as the system grows more complex and the module's resources are needed for other tasks it will be necessary to put some extra work into making the timing of each period precise enough. Fortunately, this can be achieved by using timers or interrupts.

When the modules are synchronized they can stay synchronized for some cycles without communicating, because the time to complete a cycle is approximately the same for all modules. This means that it does not matter much if a synchronization signal is lost as long as one makes it through from time to time.

In the algorithm the root module of the configuration tree emerges as the leader of the robot. This does not mean that there is a single point of failure, because if the root module fails its children will take over. In this situation each child will become the root of its own sub-tree. However, it can of course not be guaranteed that these sub-trees remain synchronized. This simple implicit leader selection mechanism is very powerful, but unfortunately it doesn't work if the configuration contains loops. In a loop synchronization signals can chase each other around without ever reaching each other. In [15] we have solved this problem by introducing IDs and combining a simple leader selection algorithm with the basic role playing algorithm.

Another point to note is that the action sequences are just open loop motor control commands. This is not desirable if the robot is to operate in complex environments where sensor feedback is essential to the survival of the robot. The method can be extend to include this form of feedback. For instance, the cyclic motion of a leg can be biased by feedback from the environment. This way the legs can change the motion to avoid obstacles or gaps. Including sensor feedback at the level of the individual module will only make the robot able to deal with problems that can be solved at the level of the individual module. If the robot has to avoid an obstacle it requires coordinated actions of all the modules to avoid the obstacle. A basic approach we have investigated in [16] is to let all the modules share sensor information through propagation. Each module then changes its role locally to react to this sensor information.

Finally another interesting question is: can this algorithm be generalized to a system without a parent-child relationship. In a real environment a single module should be able to influence the control of the entire robot based on some critical sensor information only that module has access to. How to do that is the focus of our future research.

8. SUMMARY

We have introduced a general multiagent based algorithm that can be used to implement locomotion patterns in a self-reconfigurable robot. In this algorithm each module plays a *role*. The *role* can be changed either by communication from a parent module or by detecting changes in the local configuration. It has been described how modules playing roles are synchronized. We have used this general algorithm to implement a walking gait in a self-reconfigurable robot consisting of up to 9 modules. We show in experiments that the implemented algorithm scales and is an efficient implementation of both a quadruped and hexapod gait.

9. ACKNOWLEDGMENTS

This research is supported by the DARPA contract DAAN 02-98-C-4032, the AFOSR contract F49620-01-1-0020, the EU-contract IST-20001-33060: HYDRA "Living" Building Blocks for Self-Designing Artifacts, and the Danish Technical Research Council contract 26-01-0088.

10. REFERENCES

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

[2] H. Bojinov, A. Casal, and T. Hogg. Multiagent control of self-reconfigurable robots. In *Proceedings of the Fourth International Conference on MultiAgent Systems*, pages 143 –150, Boston, Massachusetts, USA, 2000.

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

[4] K. Hosokawa, T. Tsujimori, T. Fujii, H. Kaetsu, H. Asama, Y. Kuroda, and I. Endo. Self-organizing collective robots with morphogenesis in a vertical plane. In *Proceedings of the IEEE International Conference on Robotics & Automation*, pages 2858–2863, Leuven, Belgium, 1998.

[5] B. Khoshnevis, B. Kovac, W.-M. Shen, and P. Will. Reconnectable joints for self-reconfigurable robots. In *Proceedings of the IEEE/RSJ International Conference 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 International Conference on Robotics & Automation*, pages 424–431, Leuven, Belgium, 1998.

[7] S. Murata, H. Kurokawa, and S. Kokaji. Self-assembling machine. In *Proceedings of the IEEE International Conference on Robotics & Automation*, pages 441–448, San Diego, USA, 1994.

[8] S. Murata, H. Kurokawa, E. Yoshida, K. Tomita, and S. Kokaji. A 3-d self-reconfigurable structure. In *Proceedings of the IEEE International Conference on Robotics & Automation*, pages 432–439, Leuven, Belgium, 1998.

[9] S. Murata, E. Yoshida, K. Tomita, H. Kurokawa, A. Kamimura, and S. Kokaji. Hardware design of modular robotic system. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2210–2217, Takamatsu, Japan, 2000.

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

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

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

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

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

[15] K. Støy, W.-M. Shen, and P. Will. Global locomotion from local interaction in self-reconfigurable robots. In *Proceedings of the 7th International Conference on Intelligent Autonomous Systems IAS-7*, Marina del Rey, California, USA, 2002.

[16] K. Støy, W.-M. Shen, and P. Will. On the use of sensors in self-reconfigurable robots. In *Proceedings of the Seventh International Conference on The Simulation of Adaptive behavior SAB'02 (to appear)*, Edinburgh, UK, 2002.

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

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

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

[20] E. Yoshida, S. Murata, S. Kokaji, K. Tomita, and H. Kurokawa. Micro self-reconfigurable robotic system using shape memory alloy. In *Distributed Autonomous Robotic Systems 4*, pages 145–154, Knoxville, USA, 2000.