Distributed Control Algorithm for a Multi Cellular Robot

Avinash Ranganath

Master of Science Artificial Intelligence

School of Informatics

University of Edinburgh

2009

Abstract

The project implements a framework for developing distributed controllers for locomotion and obstacle avoidance tasks in modular robotic organisms, where in the global action of a robotic organism emerges as a result of local action of individual modules. The framework is developed based on the Digital Hormone Method (DHM), proposed by *Shen et al.* (2004), according to whom, DHM is a way of controlling agents in a multi agent system, in a distributed fashion, based on the topological location, environmental variables and local communication. To achieve this, the underlying framework consists of modules for topology mapping and local communication, distributes sensing and actuation, which forms the building blocks using which a distributed controller is built. The framework has been tested by implementing locomotion and obstacle avoidance tasks on three multi cellular robotic organisms, caterpillar, 'V' and scorpion.

Acknowledgement

I would like to thank my supervisors Barbara Webb [School of Informatics, University of Edinburgh] and Marc Szymanski [Institute for Process control and Robotics, University Karlsruhe], for supporting me throughout the project. I would also like to thank Lutz Winkler [Institute for Process control and Robotics, University Karlsruhe], for all the help during the project.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

Table of Contents

1 Introduction

A self reconfigurable modular robotic system can be defined as a robotic platform where a robot is made up of several independent modules and which does not have a fixed morphology as it can reconfigure itself from one shape to another. Based on the task at hand and the situation, a robot can reconfigure itself to perform the given task. Consider locomotion, a robot can configure itself to match the shape of a snake to crawl through narrow passages, or if it is on an uneven terrain, it could reconfigure to a legged configuration, like a quadruped or a biped, to walk through uneven surface, as it can avoid obstacles more easily, or reconfigure into a ring/track shape to roll through smooth surface or roll down slopes, in quicker speeds. Individual modules, in such systems, are primarily standalone machines, which have independent sensors, actuators, and computational and power resources. Each module would have multiple connectors, through which they can physically connect to each other to form a single, bigger robot [an organism].

In such a system, an action performed by the organism [E.g. Right leg forward in a bipedal configuration] is made up of temporally coordinated local actions performed by each individual module. And combination of such actions results in a behavior. Since a multi cellular organism, in a modular robotic system, could be in one of many different configurations at any given point, the control algorithm should be able to choose the local action based on the current configuration. One approach is to have a central controlling module [which sends control instructions to all other modules], with the ability to detect the change in the configuration, and accordingly change the controlling approach to suit the new configuration. But in such a case, failure of the central module brings down the entire system. An alternative is to have a distributed control approach, where in each module chooses its own action based on some conditions.

1.1 SYMBRION

This project is a module under the EU sponsored research project called SYMBRION (Contract No: 216342). According to the authors of [1, 2, 3], the main goal of this project is to develop robots that are capable of operating as a large swarm of robots as well as building large organisms for investigating and developing novel principles of evolution and adaptation for symbiotic organisms. Each robot is fully autonomous and can act in the environment either alone

or can be physically connected to other robots to build a larger organism to solve tasks that it cannot solve alone. Such an organism will be provided with an internal bus system for sharing computing power. Additionally, robots can share electrical power within the organism. Combining evolutionary principles with swarm behaviors, the organism will autonomously manage their own hardware and software organization to become self-configuring, self-healing, self-optimizing and self-protecting. Combining the advantages of swarm robotics and the advantages from reconfigurable systems, adaptive, evolvable and scalable robotic systems can be built, which are able to co-evolve and cooperate with each other without human supervision in new and unpredictable environmental situations.

In this project I have attempted to construct a framework for developing a distributed control algorithm for locomotion and simple obstacle avoidance task in modular robotics, and I have tested the same on three different configurations. I followed a distributed control approach since it is,

a) Fault tolerant, as it does not depend on a single module, so if one or some modules fail, the robot could exist, even though not fully operational.
b) Since the modules are homogenous, the computation is well distributed and no single module has to bear the weight of the entire network.
c) No constrain on how big the organism could grow to.
d) All the modules, irrespective of where in the topology they are located in, can have an identical controller.
e) No restriction on where each module needs to be in the topology, as they are identical in hardware and software, and have no unique id assigned to them.

1.2 Module Design

All modules are identical and cube shaped. The body is made up of two ‘U’ [A 3D ‘U’] shaped parts which are interlocked to form an open ended cube. It has one degree of rotational freedom with a single motor placed in the center of the cube. It has a pair of screw driver wheels which gives it two degrees of motion, although I have never made use of wheels in my project. It has four connectors, a tilt sensor and multiple IR based distance sensors.

1.3 Simulation

All experiments of the system are done on a physic engine based Delta 3D simulator called Symbricator3D, which is developed as part of the SYMBRION project by the researchers at the Institute for Process control and Robotics [IPR], University Karlsruhe, Germany. Symbricator3D is completely developed in C++ and it has three main parts to it,

- Robot actor: This represents the physical and geometrical model of the robot, like the robot's body, sensors and actuators.
- Robot controller: This represents the software that is running on the robot that controls the robot in the simulation. This is the part under which I develop my framework.
- Simulation component: This controls the simulation environment, like checking the fitness of all the robots, creating population of robots in the environment, and for providing a user interface for the user to interact with the simulation.

The simulation allows three kinds of messaging, local message which is sent from one connected module to another connected module, organism message which is sent by one module and received by all the modules in that configuration, and global message which is sent by one module and received by all the modules in the simulation, irrespective of whether they are connected or now. I have only used local messaging in my project, as that is one of the fundamental principal behind DHM.

1.4 Configurations

I have tested the frame work by developing distributed control algorithms for three different configurations, and I have also been able to integrate the three algorithms into one and test it successfully. I have experimented with the following configurations,

1. Caterpillar: - A string of modules connected serially, shaped like a caterpillar.
2. V: - Two caterpillars joined together at one end with a center module, and with the other end open, to for a 'V' shaped organism.
3. Scorpio: - It's a 'T' shaped organism with two arms, a head and a tail.

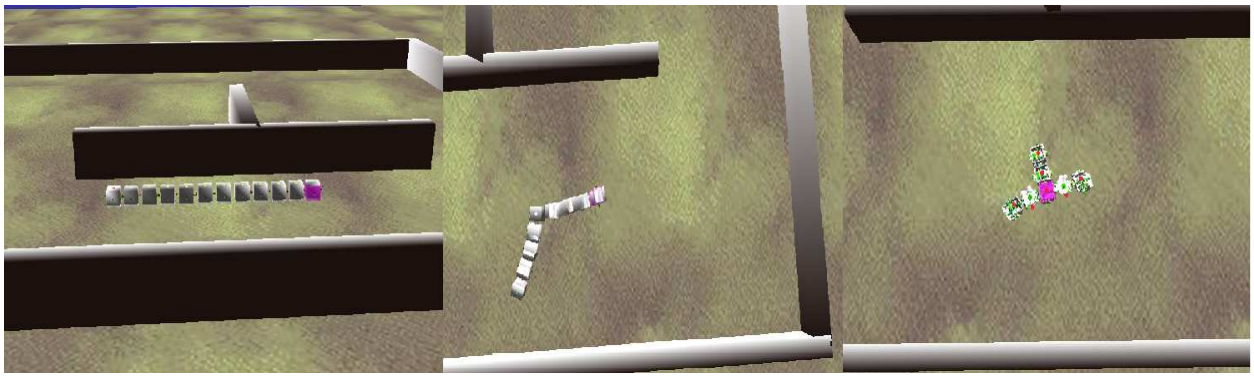


Figure 1: Three configurations.

1.5 Digital Hormones

For the control algorithm I have attempted the Digital Hormone Method, proposed by *Shen et al.* in [4]. According to the authors of [4], in biology there are different kinds of cells, each specializing in a specific task, and cells produce hormone which are diffused throughout the body and is being received by all other cells, but are being reacted upon only by the designated type of cells. Applying a similar technique in multi robotic platform, it would eradicate the need of having unique address for each robot/module. And also, in modular robotics, a module can then be in any position in the topology, such that on receiving a certain kind of digital hormone [A message], it is being reacted upon by the module which is currently in that position of the topology, rather than by a particular module that the message was being addressed to. For this, it is required to build a map of the topology so that each module knows where in topology it is located in

1.6 Topology Mapping:

I have implemented a topology mapping algorithm which runs throughout the execution of the program, which constantly sends handshake message through all of its four connectors to check if there are any connected modules. On receiving a handshake message, a module would respond by sending an acknowledgement, confirming its existence. This way each module knows what is connected to each of its connectors, or if any of its connectors are open, and with that it possibly knows where it is located on a given topology. This could be called as the Level-0 topology mapping, i.e. each module know what is connected to it directly, and there could be at most four other modules connected to each module. This is not sufficient to correctly distinguish each module in each of the three configurations, so I have extended the topology mapping to multi

level, where in each module can know how it is connected to a module which it is not directly connected to. For example, the end modules in a scorpion configuration and one of the end modules in a caterpillar configuration, all of which have a single module connected to the back connecter, make them identical modules at Level-0 mapping. But these modules need to perform different action based on which organism they are a part of. So to correctly distinguish between these two types, I need to check how they are connected to the third module, which is different in both the configurations, using level-2 mapping.

1.7 Locomotion

Locomotion gait for all three organisms are contained as a combined rule base, in the control code. This rule base is a look up table that a module needs to refer to select a local action based of the following factors,

a) Location in the topology of the current configuration.
b) Internal state like direction variable, tilt sensor and distance sensor information.
c) What the last action of the parent node was.

Each module receives a message from its parent node, which contains the last action of the parent node, based on which the module chooses and executes an action from the rule base, then sends a message to each of its child node, what its current action is. This process of passing action information to other directly connected nodes is explained as Digital Hormone Diffusion by *Shen et al.* in [4].

1.8 Obstacle Avoidance

Obstacle avoidance task is designed in a completely distributed fashion, as in, any node which is in the correct placement in a given configuration [E.g. the end nodes on a caterpillar configuration] can generate and diffuse ‘Obstacle Found’ hormones, which is reacted upon when it reaches the right kind of module.

Section 2 explains, in detail, the implementation of topology mapping, locomotion gait, and obstacle avoidance. Section 3 talks about the evaluation of the system and discuss about further improvements and a summary.

2 Implementation

2.1 Digital Hormone Method:

The digital hormone method as explained in [4] is a process of generating and diffusing state and action information, and reacting to the same when received by the designated module type(s), for self organization and self coordination in a multi robotic system. On generating a hormone, the module generating it, diffuses the same to all its connected modules, which is further diffused throughout the network through local communication by other modules. This kind of communication cannot be called as prorogation as it is not guaranteed that all the modules will receive the same message, as the messages are being modified along the path. I have extensively used the hormone generation, diffusion and reaction methods for all actions performed by each module in my project. Hormones are a collection of data packaged together and sent from one module to another. Each hormone message frames have the following structure,

Hormone Type	Connector	Hop Count	Hormone Data

Table 1: A hormone message frame prototype

- Hormone Type: - Each hormone type like locomotion hormone, obstacle found hormone and direction hormone have a unique id which helps in distinguishing between different hormone types by the hormone handling function on the receiving node.
- Connector: - This field contains one of the four connectors [Back, Front, Right or Left] of the diffusing node through which the hormone was sent. To a receiving module, the connector field helps in identify how it is connected to the module from which the message was sent.
- Hop Count: - This field is set to zero by the hormone generating module, and is incremented each time when diffused from one module to another. Hop count is used for multi level topology mapping and for obstacle avoidance task in scorpion configuration. It can also be used for knowing the module count in the configurations which can grow in size or be divided into segments over time. Module count could be used to control the speed and direction of the organism.
- Hormone Data: - The content and size of this field differs from hormone to hormone. Hormone data could be made up of one or more field. In the case of locomotion hormone,

this field contains the hinge angle and time step [time step used only by modules in scorpion configuration] and in case of ‘direction change’ hormone, this field contains the direction information. In ‘obstacle found’ hormone, this field contains information as to on which of the four sides of a module the obstacle was found. This field is empty in ‘direction flip’ hormone.

2.2 MDL2e:

The SYMBRION project was active for over a year before I started my work, and so there was already a programming platform setup and methodologies defined for me to follow. The implementation of the project was done on a two layered programming platform; a lower level behavior definition written in C++, and a higher lever behavioral collaboration done in a XML based language called as Motion Description Language Two Extended [MDL2e]. MDL2e is the work of *Szymanski et al.* [5] at the Institute for Process Control and Robotics, Karlsruhe, Germany. MDL2e is a way of describing the control algorithm of a robot, in a multi robotic platform, as a set of behaviors that resembles regular language. The order of such behavioral units, called as atoms in the context of MDL2e, corresponds to the chronology of the actions a robot/module takes. Each atom must have a unique id [in the form of a name], which corresponds to a function pointer of the function in the lower level C++ part of the implementation. The function associated with each atom gets executed as the control flows from one atom to another. Along with the name, every atom has an interrupt and timer value associated with it. The timer value is the upper limit as to for how long an atom can execute. An interrupt is associated with an internal sensor or value of the module, and it has to be true [also called active] for an atom to be executed. The timer value can be set to infinite, meaning that an atom will be executed for as long as the interrupt condition stays true. There are three types of atoms in MDL2e,

i. Executables: - It corresponds to a single action like rotate hinge motor to a certain angle, generate a certain kind of hormone, sleep for a certain amount of time, read incoming messages, etc. Executables accept values like angle or sleep time as input argument.
ii. Interrupts: - Returns a true or a false based on a certain condition defined in the corresponding function implemented in the lower level layer. It does not accept any input arguments.

iii. Variables: - Reads a value from a sensor or a variable, like distance sensor, tilt sensor, direction, etc. and returns the same to the program control. It does not accept any input arguments.

A set of atoms are tied together in what is called a ‘behavior’ and a set of behaviors make up the control program for a module. A behavior is equivalent to a function header in other programming languages. It has a name, duration and interrupt field, just as any other atom. The following example explains a simple behavior,

```
1 <BEHAVIOUR name="update-connector" interrupt="IFALSE" duration="infinite">
2       <MULT multiplicity="infinite">
3               <ATOM name="AUPDATE_CONNECTORS" interrupt="ITRUE" duration="1"/>
4       </MULT>
5 </BEHAVIOUR>
6
7 <BEHAVIOUR name="Push-Fwd" interrupt="EQ(VDIRECTION,1)" duration="infinite">
8    <MULT multiplicity="infinite">
9       <BEHAVIOUR name="update-connector" interrupt="NOT(IGOTHORMONE)" duration="infinite"/>
10
11      <BEHAVIOUR name="Scorp-Arm-Step-1-Fwd" interrupt="EQ(VDATAFIELD2,1)" duration="infinite">
12         <ATOM name="ADIFFUSE_HORMONE" interrupt="ITRUE" duration="1" arg0="0"/>
13         <BEHAVIOUR name="update-connector" interrupt="NOT(IGOTHORMONE)" duration="infinite"/>
14         <ATOM name="AROTATE_HINGE" interrupt="ITRUE" duration="2" arg0="0.75" />
15         <ATOM name="ADIFFUSE_HORMONE" interrupt="ITRUE" duration="1" arg0="0"/>
16      </BEHAVIOUR>
17
18      <BEHAVIOUR name="Scorp-Arm-Step-2-Fwd" interrupt="EQ(VDATAFIELD2,2)" duration="infinite">
19         <ATOM name="ADIFFUSE_HORMONE" interrupt="ITRUE" duration="1" arg0="0"/>
20         <BEHAVIOUR name="update-connector" interrupt="NOT(IGOTHORMONE)" duration="infinite"/>
21         <ATOM name="AROTATE_HINGE" interrupt="ITRUE" duration="2" arg0="-0.75" />
22         <ATOM name="ADIFFUSE_HORMONE" interrupt="ITRUE" duration="1" arg0="0"/>
23      </BEHAVIOUR>
24   </MULT>
25</BEHAVIOUR>
```

Table 2: MDL2e code snippet.

In the above example, line 1-5 is a template of a simple behavior, which is called in various places in the control program, like in line 9, 13 and 20. The duration of this behavior is set to ‘infinite’ during all three calls, meaning that this behavior will run as long as the interrupt condition stays true. The interrupt condition is ‘NOT(IGOTHORMONE)’, meaning that this behavior will run until a new hormone is received. The function associated with interrupt atom “IGOTHORMONE” returns true if there was a new hormone message received and returns false otherwise. A behavior can contain sub behaviors inside them, as well as make calls to other behaviors that are declared elsewhere in the file. Line number 7 and 25 is the start and end, respectively, of a main behavior called “Push-Fwd”, which contains two other sub behaviors

called “Scorp-Arm-Step-1-Fwd” and “Scorp-Arm-Step-1-Fwd” at line number 11 and 18 respectively. The following pseudo code explains the MLD2e code in table 2.

1. Execute if direction value == 1 [‘1’ indicating Forward and ‘-1’ indicating Reverse]
    1.1. Update connectors until a new hormone is received.
    1.2. If the Data Field-2 in the recently received hormone is equal to 1
        1.2.1. Diffuse the last received hormone.
        1.2.2. Update connectors until a new hormone is received.
        1.2.3. Rotate hinge motor to 0.75 radians.
        1.2.4. Diffuse the last received hormone.
    1.3. Else if the Data Field-2 in the recently received hormone is equal to 2
        1.3.1. Diffuse the last received hormone.
        1.3.2. Update connectors until a new hormone is received.
        1.3.3. Rotate hinge motor to -0.75 radians.
        1.3.4. Diffuse the last received hormone.
2. Go to step 1.

According to *Szymanski et al.* in [5], there are several advantages of using MDL2e for developing controller for robots in a multi robotic platform. The low level implementation of atoms can fit into the program memory of a robot while the high level behavior code along with an interpreter can reside on the data memory or external memory. This way, complex behaviors can be programmed into robots which are small in size and have memory constraints. Also this facilitates in code exchange between robots during runtime, which could be used in evolutionary methods, which otherwise would not be possible as you cannot write into the flash memory of most microcontrollers during the runtime. One distinct advantage of using MDL2e that I noticed during this project is that I had develop behavioral atoms at the low level once during the first half of the project when I was working on the first organism [Caterpillar], and to get the next two organisms up and running all I had to do was to modify the behavior at the MDL2e level. And furthermore, for implementing any new behavior, I had to develop atoms once in the lower level and use it straightforward in all the three configurations. So there is a clear advantage of code reusability by using MDL2e. One disadvantages of using MDL2e is that you cannot pass as an

argument to an executable atom, the return value from a variable atom directly at the MDL2e level.

2.3 Topology Mapping:

As the approach of developing a distributed controller for operating modular robots, that's I have followed, is based on knowing where in the topology a module is located in, it is necessary to develop an algorithm for the same which is reliable and stable during all situations. I was able to do it in the following way,

Each module has four connectors, one each on the back, front, right and left side of the body, through which it can physically connect to other modules. I devised a method where in each module sends out a message [Handshake Message] through each of its four connectors every few execution cycle. Similarly, each module will receive one message through each side, provided there is another module connected on to that side, every few execution cycle. This way each module can know which of its four sides are connected and which ones are not. Furthermore, the handshake message sent and received, contains the information as to from which of the four sides it was sent out from. Also, for each incoming message, the module knows through which of its four sides it came in from. Combining these two information, a module can correctly identify itself as to one of 625 possible configurations it could be in, based on what is connected to each of its four connectors. This type of mapping is called as level-0 mapping, as it is based on how a module is connected [directly] to utmost four other robots. Each robot is assigned a module type which is calculated in the following way,

L			R			F			B		

MSB LSB

Table 3a: Module type data field

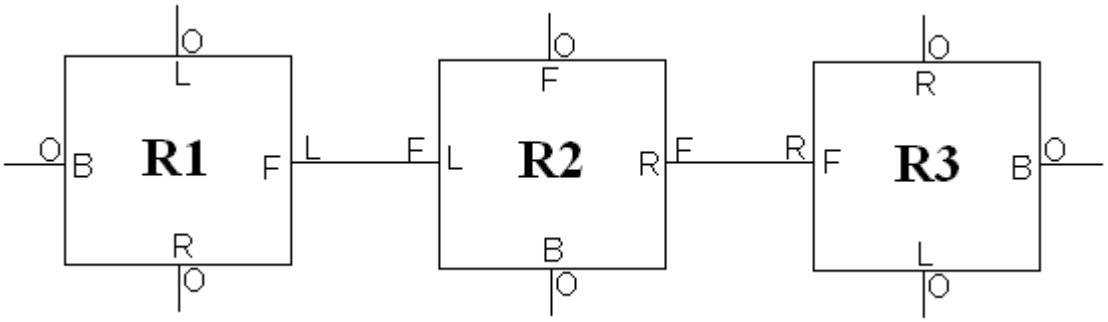
Possibility	Value
Open	0
Back	1
Front	2
Right	3
Left	4

Table 3b: Connector Status Value

Each side can be either open or connected to one of the four possible sides of another module, making it 5^4 (625) different possible configurations that a module could be in at any given point. The module type, for each module, is encoded as a twelve bit value, as seen in table 3a, where in three bits are used to encode the connection status of each of the four connectors. Table 3b

contains a list of values each three bit segment can hold based on what each of the four connector is connected to. So for example if a module's left and right connectors are connected to front connectors of two other modules [module R2 as seen in figure 2], with its front and back connectors open, then its module type is calculated as explained in table 4b, and will hold the value '1152'. Front connector of modules R1 and R3 are connected to left and right connector, respectively, of module R2, and the rest of the three connectors on each of these two modules are open. So the module type of R1 and R3 is calculated to be '32' and '24' respectively, as explained in the table 4a and 4c respectively.

Figure 2: A simple three module configuration



R1 = 32 Table 4a

L			R			F			B		
0	0	0	0	0	0	1	0	0	0	0	0

MSB LSB

R2 = 1152 Table 4b

L			R			F			B		
0	1	0	0	1	0	0	0	0	0	0	0

MSB LSB

R3 = 24 Table 4c

L			R			F			B		
0	0	0	0	0	0	0	1	1	0	0	0

MSB LSB

Table 4: Module Type explanation for types '24', '32' and '1152'

A handshake message frame for Level-0 topology mapping contains the following field,

Message Type	Acknowledgment	Hop Count	Connector

Table 4: Level-0 handshake message frame

- Message Type: - This field is used to differentiate handshake messages from other hormone message.
- Acknowledgment: - This field is set to '0' indicating that the module who has sent out this message is initiating the handshake process. And it is set to '1' indicating that the handshake message has been acknowledged.
- Hop Count: - Hop count is set to zero and it does not play a role in Level-0 topology mapping. It is used for multi level topology mapping.
- Connector: - This field ranges from 1 to 4 indicating from which of the four sided the handshake message was sent from. So if a module receives a handshake message, through its left connector, with the connector field containing the value '2', it implies that this module's left connector is connected to the front of another module.

Each module has a container [An array of four character variables] which stored the status of each of its four connectors, and a variable to hold the calculated module type value of the module. Both these values are initialized to zero in the beginning of the program, indicating that a module has all of its connectors open. As it makes connections and sends and receives handshake messages it accordingly updates the status of each of its connectors and the module type value. For example, a module receiving a handshake message through its back connector containing '4' in the "Connector" field will update the back connector value to '4'. The module type of a module is calculated by doing a bitwise OR operation between the module type value and the connector value of each of the four connectors in the following way,

Step 1: Module Type = Module Type OR (Back Connector * 8^0)

Step 2: Module Type = Module Type OR (Front Connector * 8^1)

Step 3: Module Type = Module Type OR (Right Connector * 8^2)

Step 4: Module Type = Module Type OR (Left Connector * 8^3)

And at the end of step 4, Module Type variable will be updated with the correct module type based on the connection information. Module Type variable can contain a value ranging between 0 [If all the four connectors are open] and 2340 [if all the four connectors are connect to left connectors of another module], but there are only 625 valid values that the Module Type variable can contain. A simple check on the value in the 'Connector' field of an incoming handshake message would make sure the Module Type value is correctly calculated.

This way of encoding the connection information and calculating module type for differentiating between modules based on topological location is quick, easy, effective, reliable, computationally economical and utilizes a small amount of memory. But it is insufficient when the complexity of the controller increases as it includes control algorithm for more than one kind of organism, or when the number of modules in an organism grows as the structure of the organism becomes more and more complex. Consider the following example,

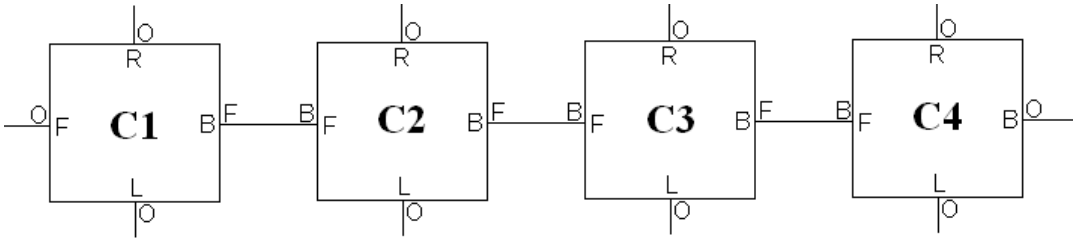


Figure 3: Topology of a caterpillar organism

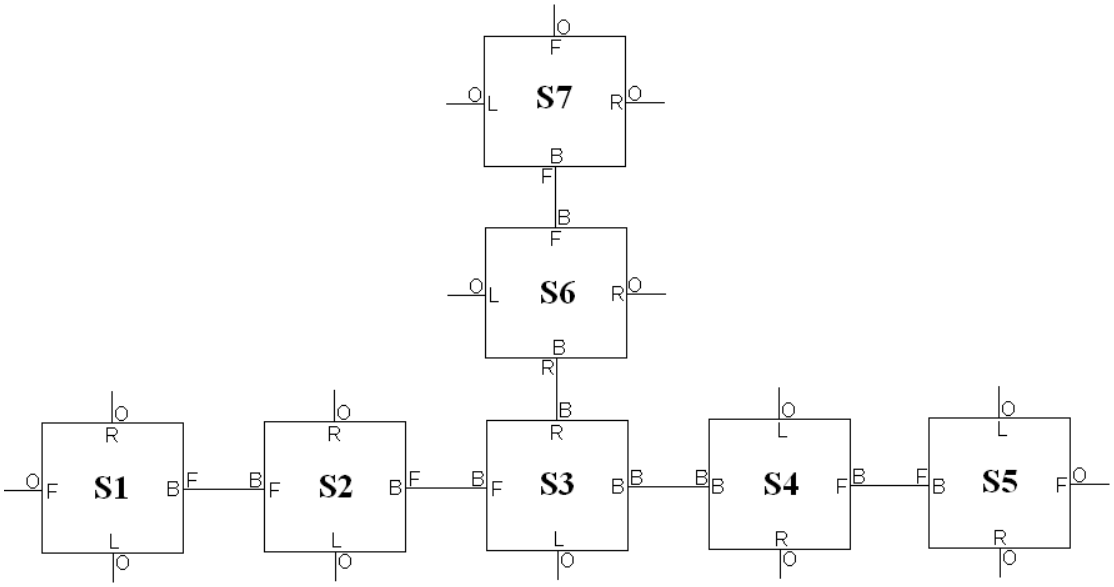


Figure 4: Topology of a Scorpion organism

Figure 3 represents the configuration of a caterpillar organism, and figure 4 that of a scorpion organism. Module ‘C1’ of caterpillar organism and modules ‘S1’, ‘S5’ and ‘S7’, of the scorpion organism, are all of module type ‘2’ since each of them have their back connector connected to the front connector of another module, and the rest of the connectors are open. Although they are of the same module type, based on how they are connected to other modules directly, they need to perform different local actions to produce the correct global action of the organism they belong to. In some case module ‘C1’ performs the same action that its parent module ‘C2’ performs and other wise module ‘C1’ initiates the hormone flow by generating the locomotion hormone. Whereas module ‘S1’ and ‘S5’, from the scorpion organism, rotate their hinge motors alternatively between +0.75 and -0.75 radians and generates locomotion hormone. And module ‘S7’ does not rotate its hinge motors, but constantly scans for obstacle using distance sensors. So to correctly identify if a module of type ‘2’ belongs to the caterpillar organism or if it is one of the two front end modules or the tail module in the scorpion organism, we need to extend the topology mapping to identify how each module is connect to other modules that it is not directly connected to, but which are a part of the same organism. What we have seen until now can be called as a Level-0 topology mapping. A Level-1 mapping would be, how a module is connected to another module which is exactly one module’s distance away from it. Example for modules that are one modules distance away from each other are, modules ‘C1’ and ‘C3’, and modules ‘C2’ and ‘C4’ in the caterpillar organism in figure 3. Similarly modules ‘S6’ and ‘S2’, in the scorpion configuration in figure 4, are one modules distance away from each other, and modules ‘S7’ and ‘S4’ are two modules distance away from each other. Multi level topology mapping can be extend to any number, provided there is sufficient memory to store the information.

I have implemented multi level mapping by propagating the handshake message a module receives from one of its connected modules to all other connected modules, by adding the connector information of its own to the message frame. Consider the following example,

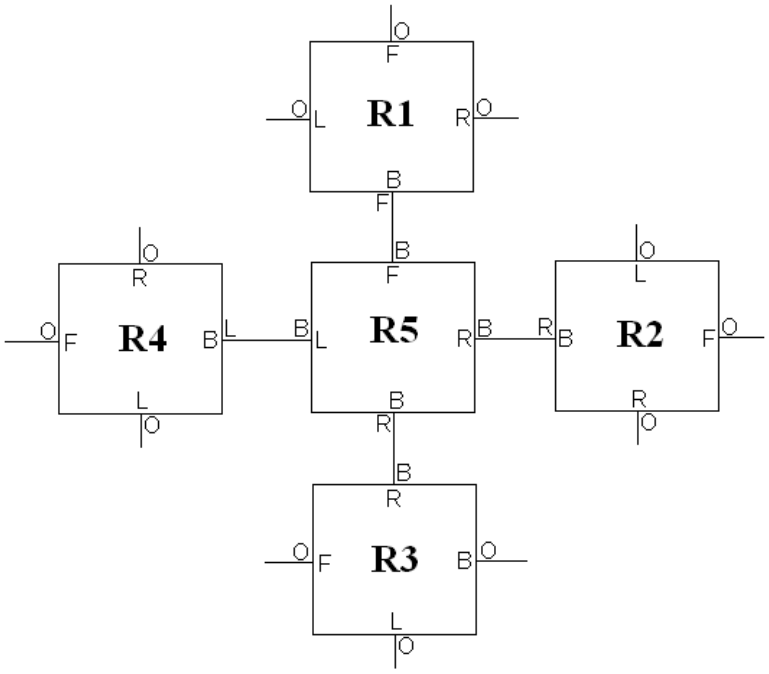


Figure 5: A five module configuration for explaining level-1 topology mapping.

In the above figure 5, each of the modules ‘R1’, ‘R2’, ‘R3’ and ‘R4’ has module ‘R5’ at level-0 and each other at level-1 distance. Whereas module ‘R5’ has the remaining four modules at level-0 and none at level-1 distance. To implement level-1 mapping, for every message that module ‘R5’ receives, it appends the message frame with the connector through which it received this message from, increments the hop count by one, and sends it out to the remaining three modules. For example, when ‘R5’ receives a handshake message from ‘R4’ the message frame contains the following data,

Message Type	Acknowledgment	Hop Count	Connector
Handshake Message	1 or 0	0	Back

Table 5: Level-0 handshake message frame.

When this message is appended and retransmitted to module ‘R2’ from ‘R5’, the message frame contains the following data,

Message Type	Acknowledgment	Hop Count	Connector	
Handshake Message	0	1	Left	Back

Table 6: Level-1 handshake message

As you can see above, the hop count is increased to 1, indicating that it is a level-1 handshake message, and the connector field now contains ‘Left’, in addition to ‘Back’, which can be decoded as ‘My back connector is connected to the right connector of a module [I know this by level-0 handshake message] whose left connector is connected to the back connector of another module’.

I have implemented multi level topology mapping by introducing a four way linked list for storing the connector and connected module information, at various distances in the network. The structure of the linked list contains four connector variables, for holding the connector information of each of the four connectors of a module, and four address variables for holding the address of four other linked list, each representing a connected module. Each module starts off with a single node in its memory, with all the eight variables initialized to zero during the beginning of the program. As the module makes connections, sends and receives handshake message, at different level, it according populates its virtual map of the topology, and the linked list grows in size. Although each module could potentially know how it is connected to each other module in the entire network, it may not be required to know the same in all situations. For instance, a caterpillar configuration, with ten modules, would need up to a level-8 mapping for each modules to know how it is connected to every other module in the network. But this may not be required, so a simple check on the hop count of a received handshake message, before propagating it, can restrict the mapping to any level. I have used up to a level-2 mapping for correctly identifying all the different module types in the three organisms.

The multi level topology mapping, which I have implementer, is dynamic in nature, as it build up the virtual map of the topology, by creating nodes, during runtime. A module can be connected to utmost four other modules at level-0, twelve modules at level-1, thirty six modules at level-2, and so on. So the maximum number of modules a module could be connected to at a certain level ‘n’ can be formulated as,

Max. no. of modules at level-n = $\sum_{i=0}^{n} 4(3^i)$ (1)

Even though there could be a maximum of 52 modules connected to a module at a distance of level-2, topology mapping at this level would not require the same number nodes to be created in the linked list that resides as a virtual map on a module’s memory. One node is created for every

connected module on receiving a handshake message. So for instance, implementing a level-2 topology mapping on a module, that is part of a caterpillar configuration, which has three modules connected towards its back and three on its front, then it will only have seven nodes in the linked list based topology map, three on each side and one node representing itself.

Another important feature of the topology mapping algorithm that I have developed involves correctly updating the connection status when one or more modules gets disconnected or added to the network, during runtime. At level-0, each module sends out a handshake message every few execution cycle, through all the four connectors, irrespective of whether it's currently connected to another module or now. This makes sure that when one of the previously open connectors, of a module, gets connected to another module, during run time, it gets correctly identified and the module type gets updates accordingly. Also, after sending out a handshake message, a module waits for a certain amount of time before resetting the connector status to '0' [which indicates that the connector is open], if it does not receive back an acknowledgement message through that connector. Also, when a module's previously occupied connector becomes open, it sends out a level-1 hand shake message to all of its other connected modules, informing about the change on one of its connector, and the same gets propagated throughout the network. For instance when modules 'R5' and 'R2' [from figure 5] gets disconnected, module 'R5' sends out a level-1 handshake message to the remaining three modules. The following shows the message frame sent out from module 'R5' to module 'R4',

Message Type	Acknowledgment	Hop Count	Connector	
Handshake Message	0	1	Right	Open

Table 7: Level-1 handshake message informing losing a connection.

The above message, when received by module 'R4', is decoded as, 'My back connector is connected to the left connector of a module [which is known by level-0 handshake message] whose right connector is now open'. Accordingly, module 'R4' deletes the node representing the module 'R2' from the linked list network in its memory and changes the connector variable, representing the right connector in the node which represents module 'R5', from 'Back' to 'Open'. The same is done in modules 'R1' and 'R3' as well..

Going back to the caterpillar and scorpion example, explained on page [??], where in both the organisms had modules of common module types, based on level-0 mapping, it is now possible,

with the extended topology mapping, to correctly distinguish between these modules based on how they are connected to other modules which are two module's distance away.

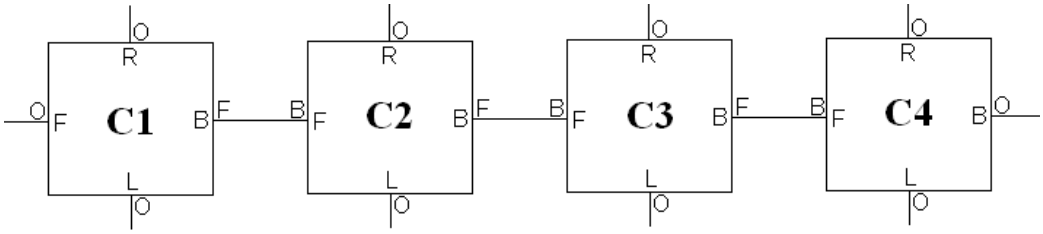


Figure 6: Topology of a caterpillar organism

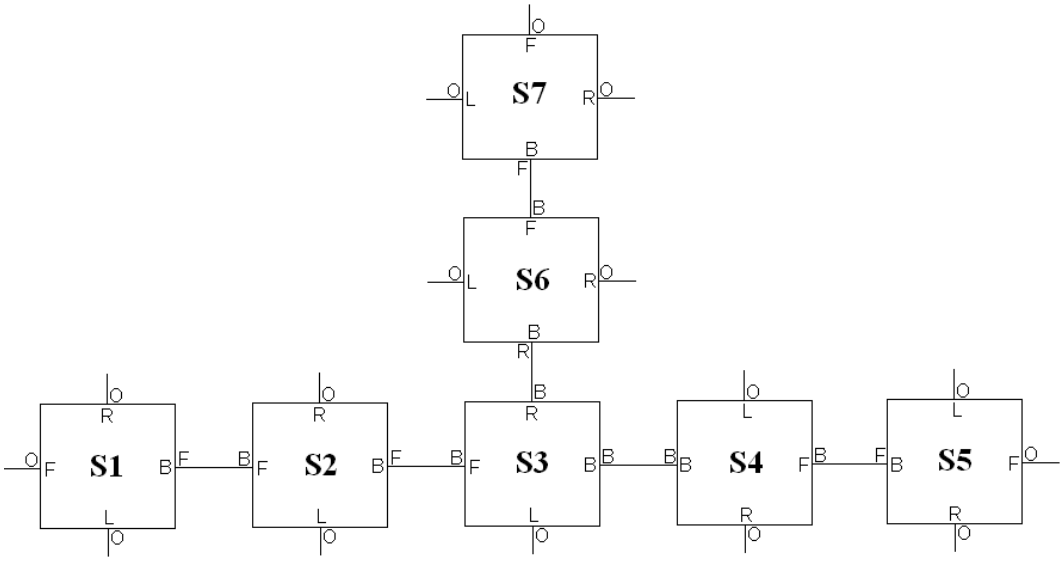


Figure 7: Topology of a Scorpion organism

Modules 'C1', 'S1', 'S5' and 'S7', are all of module type '2' based on level-0 mapping. But consider how each one of them is connect to modules that are two modules away from them. Module 'C1' has only one module, 'C4', that is at level-2 and it is connected to it in the following way

- C1 to C4: {(B, F), (B, F), (B, F)}

Modules 'S1', 'S5' and 'S7' have two modules each, that are at level-2, and following is how they are connected,

- S1 to S4: {(B, F), (B, F), (B, B)}

- S1 to S6: {(B, F), (B,F), (R, B)}

- S5 to S2: {(B, F), (B, B), (F, B)}
- S5 to S6: {(B, F), (B, B), (R, B)}

- S7 to S2: {(B, F), (B, R), (F, B)}
- S7 to S4: {(B, F), (B, R), (B, B)}

Each of the module type '2' module is connected differently at level-2, and can be correctly discriminated using a level-2 topology mapping, so that they can perform the correct local action based on the organism they are a part of and based on their location in the topology.

2.4 Locomotion:

Locomotion in modular robots is achieved by coordinated local action of individual module that translates into a global action, resulting in the locomotion of the robot organism. Each module has one degree of freedom, and the robot body is made up of two 'U' shaped segments connected together as shown in figure 8, with a motor in the center. The two segments are connected together with a hinge allowing each segment to swing relative to the other. The rotation of the motor causes both the body segments to swing up or down based on the direction of the rotation. Each module also has a pair of screw driver wheels, as seen in figure 8. Rotating both the wheels forward, moves the module forward, rotating them reverse makes it move backward, rotating one of them forward and the other backward makes the module move sideways on one direction, and alternating the wheel rotation makes it move sideward on the other direction. So this wheel mechanism gives each module the freedom to move in all four directions. The design and development of the real modules and simulated models of the modules are the work of researchers working under the SYMBRION project at the IPR lab, Karlsruhe, Germany.

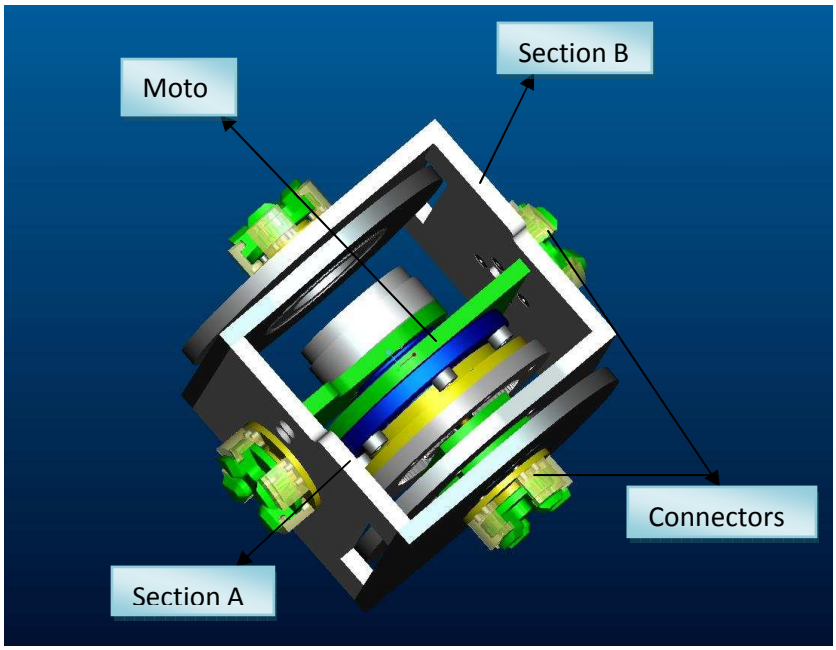


Figure 8: Top view of the module [Source: IPR]

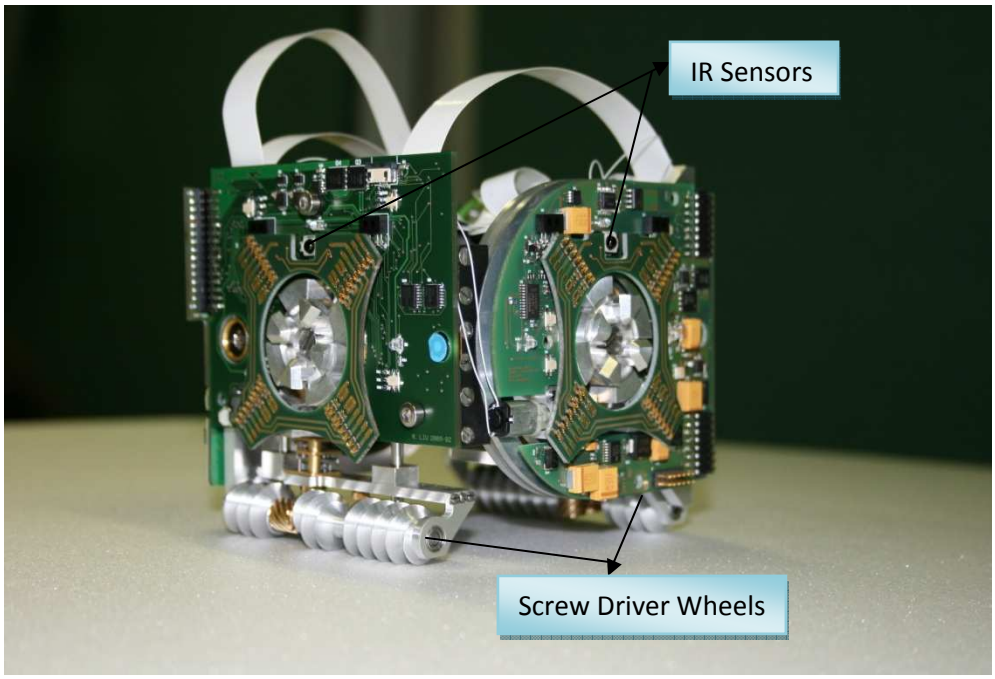


Figure 9: Side view of the module [Source: IPR]

2.4.1 Caterpillar:

Modules in the caterpillar configuration are connected serially one behind the other, with each module's back connector connected to the front connector of a module behind it, and its front

connector connected to the back connector of a module in front of it. Rotating the hinge motor of the module by '+45' degrees makes the two body segments swing upwards making it lift the two connected modules. And by rotating the hinge motors by '-45' degrees, makes the two segments swing downwards pushing the two connected modules downwards along with it. By rotating the hinge motor, of each module one after the other, alternatively between '+45' and '-45' degrees every 'n' time steps produces a sine wave, as seen in figure 10, which propagates the organism forward.

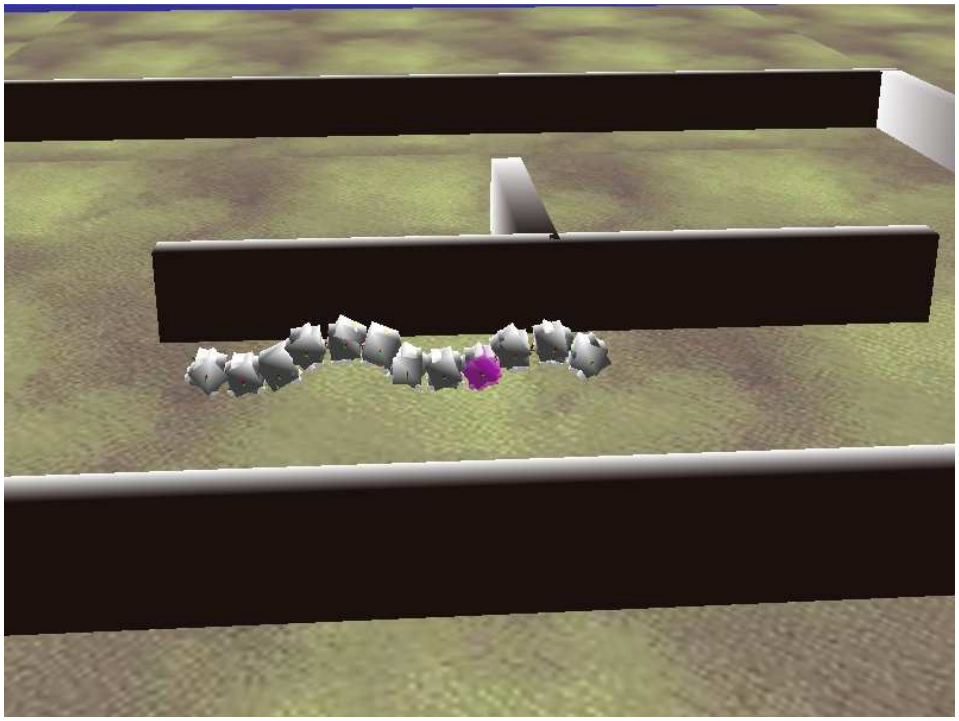


Figure 10: Sine wave seen in caterpillar gait.

The value 'n', i.e. the number of time steps between which the rotation of the motor is alternated, determines the frequency of the sine wave. If the value of 'n' is too small then the frequency of the sine wave increases, but there is no significant change in the speed of motion as the organism takes faster but smaller steps, and the motion at higher frequency is not very smooth. Making the value of 'n' too big creates very big sine wave making the organism unstable, as it can topple towards its side. Alternating the rotation between every four steps produces a smooth, stable and fast locomotion.

To achieve the sine wave gait, the alternating hinge motor rotation of modules should happen one module after the another, i.e. if there are five modules in a caterpillar organism, with the first module on the rightmost side and the fifth one on the leftmost side, then the following gait should be followed,

Time Step	Module	Motor Angle [Degrees]
1	Module 1	45
2	Module 2	45
3	Module 3	45
4	Module 4	45
5	Module 5	45
	Module 1	-45
6	Module 2	-45
7	Module 3	-45
8	Module 4	-45
9	Module 5	-45
	Module 1	45

Table 8: Caterpillar locomotion gait for organism length of five modules

If all the modules rotate to '+45' degrees at the first time step and to '-45' degrees at the fourth time step then it does not produce the desired results. The force gets transferred from left to right, without the organism moving anywhere.

In the caterpillar gait, the end from where the rotation start on the first time step, produces a wave that move in the direction of the other end, and the organism moves in the direction of this wave. That is, if the rightmost module start the rotation, then the organism moves from right to left and if the rotation starts with the left most module, then the organism moves from left to right.

To produce the caterpillar gait, using Digital Hormone Method, a locomotion hormone has to be diffused from module to module which contains the last hinge motor rotation information of the module which sends out the hormone message. This way, on receiving a hormone, the receiving module would perform the same action as its parent module and sends out a hormone to its child module containing its hinge motor's current angle. This flow of hormone from module to module produces the sine wave and propagates the organism forward. This way of hormone generation and diffusion involves one module never receiving a locomotion hormone and one module never

having to diffuse a locomotion hormone. In the caterpillar configuration, the module on the extreme right, which has its front connector connected to the back connector of the module in front of it, and rest of its connectors open, is of module type ‘8’, the module on the extreme left, which has its back connector connect to the front of a module behind it, is of module type ‘2’, and the rest of the modules are of type ‘10’. Depending upon which direction the organism needs to move, the hormone generation gets initiated from the module on the opposite direction. If the organism need to travel from left to right, then the module on the extreme left [The one whose module type is ‘2’] starts the hormone diffusion, and if the organism needs to move in the opposite direction then the hormone diffusion is initialized be the module on the other end [The one whose module type is ‘8’]. So the other modules, the ones in between which are of module type ‘10’, on receiving the locomotion hormone from one side, acts on the hormone and diffuse it through the other side. This way it does not matter, for the modules in between, as to which direction they need to move. The end modules constantly refer to the inter direction variable, depending on which, the one responsible starts the locomotion hormone diffusion. The module that starts hormone diffusion, does not depend on any incoming hormone to take its local action of rotating its hinge motors, it oscillates its hinge motors between ‘+45’ and ‘-45’ degrees, every ‘n’ time steps, which is specified in the algorithm.

To make the caterpillar organism turn around as it moves, one of the modules in the center of the organism is pitched 90 degrees towards left, as shown in figure 11.

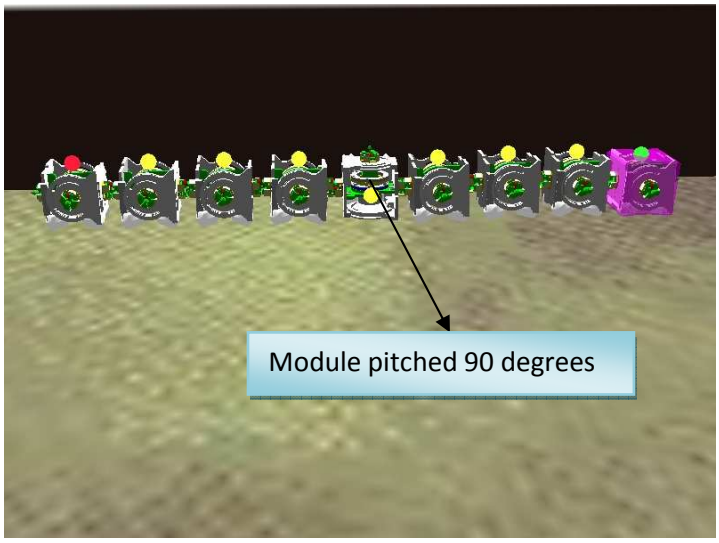


Figure 11: Caterpillar organism with the center module pitched 90 degrees towards left.

By rotating the hinge motor of the pitched module by a positive degree, for a short bust of time [one time step], and getting it back to zero degrees, would slightly tilt the organism towards the left [i.e. if the organism is moving from right to left] as it navigates. Repeating this process more number of times makes the turn sharper. Also, the amount of positive rotation determines the sharpness of the turn. Rotating the hinge motor by a negative degree makes the organism turn right.

The hinge motor of the pitched module has to stay at zero degrees, while moving forward or reverse, for the organism to move on a straight path. So this module, on receiving a locomotion hormone, should pass it on to the next module without altering the message and without taking any action. A module is correctly identified as being pitched by constantly reading the tilt sensor of all the modules.

The turning of the caterpillar organism using the pitched module is controlled by diffusing a ‘found obstacle’ hormone, which is designated to either a pitched module or the locomotion initiating module. On receiving such a hormone, the pitched module rotates its hinge motor according to turn the organism either left or right, based on the data contained in the ‘found obstacle’ hormone, which tells specifies which side the obstacle was found. The ‘found obstacle’ hormone is generated by the end module which is not the locomotion hormone generator. That is, if the organism is moving from right to left, then the left most module will generate the obstacle hormone and diffuse it through its back connector [Its only connector that has a module connected to it]. This hormone gets passed on from module to module, until it reached the pitched module, which is designated to react to such hormones by rotating its hinge angle accordingly. After reacting to an obstacle hormone, the pitched module does not diffuse the hormone any further. If the caterpillar configuration does not have a pitched module, with all the modules up straight, then the only other way to avoid an obstacle is to move in the reverse direction. So when the end module generates an obstacle hormone, it gets passed on until it reaches the other end of the configuration [The locomotion hormone generating module]. On receiving such a hormone, the locomotion initiating module, changes its internal direction variable, and generates and diffuses a ‘direction change’ hormone, which contains its updated value of the direction variable, and stops diffusing the locomotion hormone [Since its internal direction variable has now changed]. On receiving a ‘Direction Change’ hormone, each module

will change its direction variable and diffuse the same. When this hormone reaches the end of configuration [To the module that found an obstacle and initiated the obstacle hormone], the end module makes change to its internal direction variable, which makes it the locomotion initiating module, so it starts generating locomotion hormone, which results in the organism moving in the opposite direction.

2.4.2 'V'

Connecting the open front connector of one end of a caterpillar organism to the left connector of the end module of another caterpillar organism creates a 'V' organism [Figure 12a and 12b], which is an evolved version of the caterpillar organism.

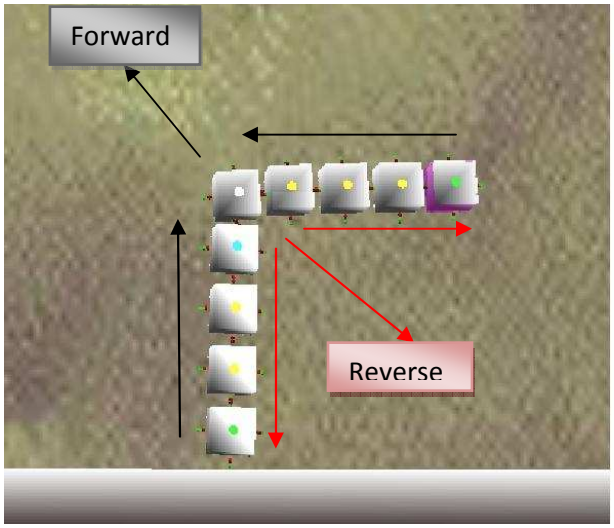


Figure 12a: 'V' organism motion direction

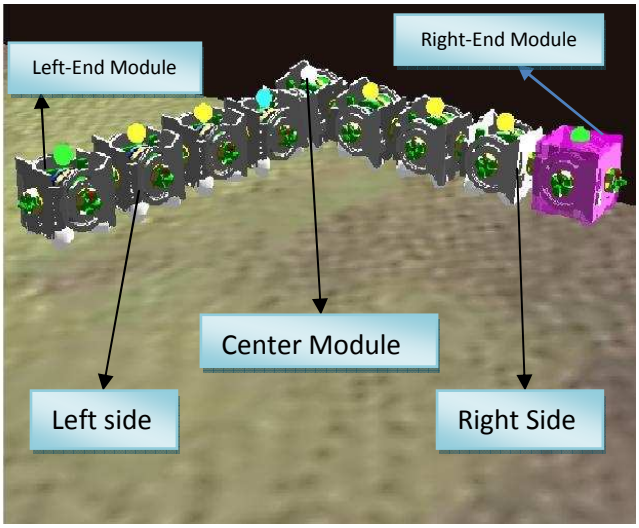


Figure 12b: 'V' organism configuration

By generating a sine wave which travels from the two ends of the 'V' towards the center of the 'V' creates a force vector that moves the organism in the direction diagonal to the two sides of the organism, towards the region the center module is pointing to [Towards the black arrow as shown in figure 12a]. By generating a sine wave that starts at the center of the 'V' and moves towards the ends of the 'V', makes the organism move in the opposite direction [Towards the red arrow in figure 12a]. Unlike in the caterpillar organism, it is possible to make the 'V' organism turn around without having to pitch any module towards the side. While moving forward, making one side of the 'V' organism stop reacting to the locomotion hormone would make the organism rotate on its axis on towards that side, i.e. if, while moving forward, the left side of the organism were to stop generating the sine wave and all the modules on this side stay at a constant

angle, with only module on the right side creating the sine wave, then this would make the organism rotate on its own axis towards left. Same way, stopping the right side while the left side pushes forward, makes the organism rotate towards the right side.

The two end modules in this organism are of module type ‘8’, the center module is of type ‘1026’ and the rest of the modules are of type ‘10’ and ‘9’, similar to that of the non-end modules in the caterpillar configuration. So the locomotion hormone generation, diffusion and reaction is very similar to that of the caterpillar organism. For the organism to move forward, the two end modules initiate the locomotion by independently generating the locomotion hormone and diffusing it. The rest of the modules, excluding the centre module, react to the locomotion hormone by rotating their hinge motor accordingly [i.e. between +45 and -45 degrees], and by diffusing the locomotion hormone to their child module. For the organism to move in reverse direction, the center module initiates the locomotion by generating and diffusing the locomotion hormone, which creates two sine waves starting from the center and moving away towards the ends of the ‘V’ organism.

Unlike forward and reverse motion mechanism, which is common in caterpillar and ‘V’ organisms, the mechanism for rotating the ‘V’ organism is different from that of the caterpillar organism. While moving forward, the centre module scans for obstacles on its two sides and on finding an obstacle through its right side sensor [which indicates that the obstacle is on the right side of the organism], it generates and diffuses a ‘direction change’ hormone, without changing its own direction variable, through its left connector [the side which holds together the left segment of the organism]. This propagates till the end of that side, on the way changing the direction variable of all the modules. This result in the left section of the organism becoming inactive, while the right side still producing the sine wave, which results in the organism rotating towards left on its own axis. Similarly on finding an obstacle on its front side [which indicates that the obstacle is on the left side of the organism], the center module, sends a direction change hormone through its back connector [the side which holds together the right segment of the organism], and this results in the right section becoming inactive which rotates the organism towards the right side.

When the organism is moving in reverse direction, i.e. when the hormone is generated and diffused by the center module, the two end modules actively scan for obstacles. When either of

the end modules picks up an obstacle, they generate and diffuse an ‘Obstacle Found’ hormone, which gets diffused from module to module until it reaches the center module. If the center module receives two obstacle found hormones, one each from the two sides, within a short fixed duration of time, then the center module makes a change to its internal direction variable, stops generating locomotion hormone, generates and diffuses a direction change hormone through both its connectors, which gives the two end modules the privilege to generate locomotion hormone, and the organism starts to move in the opposite direction. When the center module receives an obstacle found hormone from one of its side, like through its left connector, indicating that there is an obstacle on the left side, it generates and diffuses a direction change hormone through its back connector only. Also it does not make a change to its own internal direction variable, but it stops diffusing the locomotion hormone through the back side. On receiving the direction change hormone, the right end connector starts to generate and diffuse locomotion hormone. Now with the left segment of the organism, producing a sine wave which propagates from the center, outwards towards the left-end module, and the right segment of the organism, producing a sine wave that starts from the right end module, propagating inwards towards the center of the module, the organism starts to rotate towards left, avoiding the obstacle. Then, after a fixed short duration, since the center module generated and diffused the direction change hormone, it generates and diffuses another direction change hormone, through its right connector, which makes the organism go back to the reverse motion. This continues until the obstacle has been completely avoided. Similarly, to rotate right, on finding an obstacle on the right, while moving in reverse direction, the center module generates and diffuses the direction change hormone through its left connector, making the left section generate the forward moving sine wave for rotating the organism towards right.

2.4.3 Scorpio

The scorpion organism is made up of fixed number of modules, seven in all. A head module, two tail modules and four arm modules, as explained in the figure 13. Following table gives the module type, based on level-0 mapping, of each of the seven modules of the scorpion organism,

Module Name	Module Type
Head	73
Right Inner Arm	10
Right Outer Arm	2
Left Inner Arm	9
Left Outer Arm	2
Inner Tail	11
Outer Tail	2

Table 9: Module type of scorpion organism

Modules inner left and right arm are pitched 90 degrees, which makes them push the two connected modules forward, when the hinge motors are rotated by a positive angle, and push the connected modules backward when the motors are rotated by a negative angle.

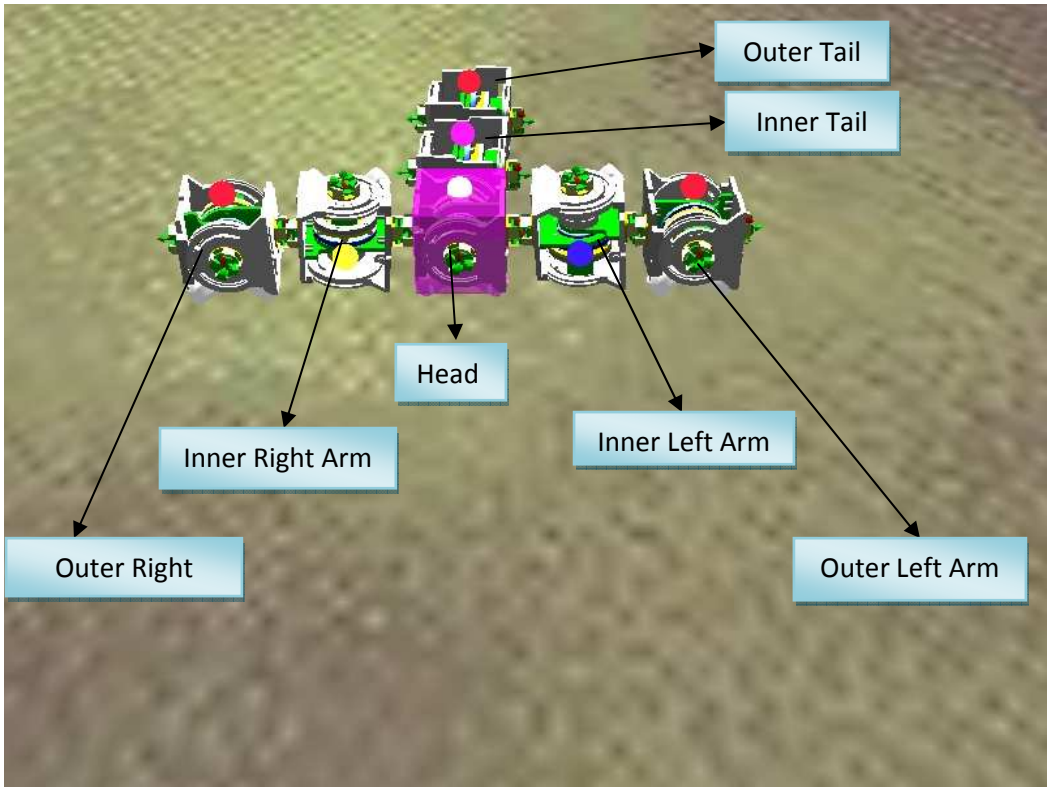


Figure 13: Scorpion configuration and modules.

The outer left and right arm modules rotate their hinge motor, alternating between a positive and negative angles, which make the arm lift up and push down into the ground, which is required for performing the butterfly gait. The locomotion gait performed by the scorpion organism is

similar to that of butterfly stroke in swimming. The forward motion in scorpion organism is attained by performing the following steps,

i. Push both the arms up by rotating hinge motor of the two outer arm modules by a positive angle of ‘0.75’ radians.
ii. Move both the arms forward by rotating the hinge motor of the two inner arm modules by ‘0.75’ radians.
iii. Push down both the arms by rotating the hinge motor of the two outer arm modules by a negative angle of ‘-0.75’ radians.
iv. Now move both the arms backwards by rotating the hinge motor of the two inner modules by ‘-0.75’ radians. This would pull the entire organism forward.

Reverse motion in scorpion organism can be attained by swapping step (ii) and step (iv) above. That is, the arm gets lifted up, then the arm swings backwards, in the third step, the arm gets pushed down, making contact with the ground and in the fourth step, the arm swings forward pushing the organism backwards. Rotation of the scorpion organism can be achieved by performing the forward motion action by one arm and reverse motion action by the other. The locomotion of the organism is achieved by coordinated action of the four arm modules, and the remaining three modules, head and two tail modules, do not contribute to locomotion. Also, the propulsion of the entire organism is the result of the actuation of the two inner arm modules.

Locomotion hormone, in the scorpion organism, is generated and diffused by the head module. The hormone message contains the hinge motor angle [which is always ‘0’ for the head module] and the time step, which alternates between ‘1’ and ‘2’. This message is diffused to both the inner arm modules and inner tail module. The tail module is a dead node and so ignores this hormone. The inner arm modules do not react to this hormone, but only diffuse the same to their respective outer arm modules. The outer arm modules, depending on the time step either rotates the hinge motor to ‘+0.75’ radians or to ‘-0.75’ radians, and then generates a locomotion hormone, containing their current hinge motor angel, which gets diffused to their respective inner arm modules. On receiving the locomotion hormone from the outer arm modules, the inner arm modules would either rotate their hinge motors to the same angle as their respective outer arm modules, or to (received angle * (-1)) radians [i.e. towards the opposite direction], based on what the internal direction variable reads. If the direction variable reads ‘forward’ then the inner

arm module will rotate to the same angle as its outer arm module, else if the direction is reverse, then it rotates in the opposite direction, as explained in the four step butterfly stroke gait, above. Then the inner arm modules diffuse the hormone to the head module. The head module would wait until it receives a hormone message from each of the inner arm modules before it generates the next set of locomotion hormone. This way, locomotion hormone, in the scorpion configuration, flows to and fro starting from the head module, out towards the outer arm modules and back to the head module.

To make this organism move in reverse direction, the direction variable of the inner arm modules should be changed to 'reverse', which makes the arms swing forward, while the outer arm modules are pressed against the ground, which pushes the organism backwards.

To make the scorpion organism rotate right, a direction change hormone, changing the direction variable from 'forward' to 'reverse', should be generated targeted at the inner right arm module, on receiving which, the right arm would start to push backwards while the left arm continues to pull forward, this would make the organism rotate towards its right. Similarly to make the organism rotate left, a direction change hormone should be generated, which would change the direction variable of the inner left arm from 'forward' to 'reverse', which would make the left arm push backwards and the right arm pull forward, making the organism rotate left.

2.5 Obstacle Avoidance:

Each module is attached with an IR distance sensor on each of the four sides. Internally, all modules scan all the distance sensors, which are on the side of an open connector, once every execution cycle. Distance sensors only on those sides with an open connector are scanned because the sensors on the side which has another module connected to it, gets masked by the connected module, and will always read very low value, which indicates that there is an obstacle on that side. Also, the left and right side sensors of a module that is that is pitched 90 degrees are not scanned even if the connectors on that side are open because one of the sides lie on the ground and the other face upwards towards the sky, making the sensor reading from these two sides useless for obstacle avoidance. This way of scanning only the potentially useful sensors, by leaving out the rest, reduces a considerable amount of load on the system, since the reading are taken once ever execution cycle. Furthermore, not all scanned sensor readings of all the modules are used in determining if there is an obstacle. For instance, in a caterpillar configuration, only

the sensor reading from the three open sides of a non-locomotion initiating module is used to determine if there is an obstacle in front.

Determining if there is an obstacle or not by reading sensor values, in these organisms, is not as straight forward as it is on a wheeled platform where in you set a lower threshold, and scan the distance sensor constantly until the value drops down to this threshold value, which indicates that there is an obstacle. Consider the caterpillar organism moving from right to left, the sensor on the front side of the module on the extreme left [Which is the non-locomotion initiating end module] reads out distance value ranging between 0 and 128 [‘0’ indicating that there is an obstacle right in front of the sensor, and ‘128’ indicating that there is no obstacle in the vicinity], as this side touches the ground and faces upwards towards the sky once every complete sine wave, the following figure shows both the scenario.

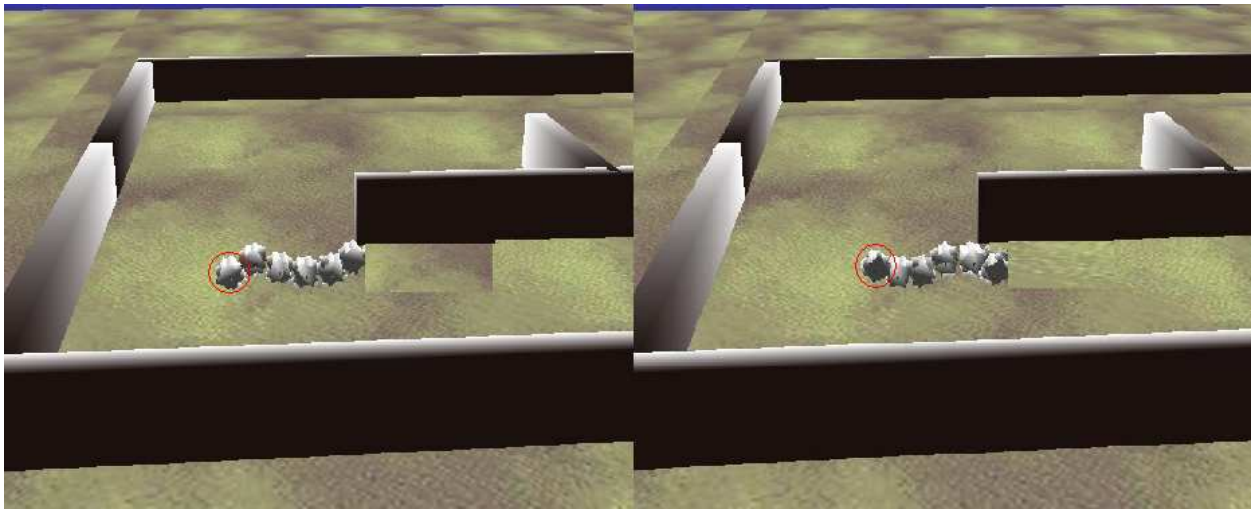


Figure 14: The front side of the extreme left module [The one circled] on the left frame is almost parallel to the ground. The same side of the same module is facing upwards on the right frame.

The same is the case with the two end modules in the ‘V’ organism. Due to this, determining if there is an obstacle or not by directly reading off sensor values will not be possible. So you need to look at the last few sensor readings to determine if there is an obstacle or not. I have a buffer for each of the four sensors, for storing the last ‘n’ sensor readings. The size of the buffer is set to ‘10’ after much experimentation. Taking an average of the buffer gives a better estimate as to whether there is an obstacle or not. The average needs to be compared with a lower threshold

limit, and this value differs for each side and between different organisms, due to the difference in locomotion gait and difference in local action of each kind of module type.

2.5.1 Caterpillar

In this organism the non-locomotion initiating module, i.e. the left-end module when the organism is moving from right to left, and the right-end module when the organism is moving from left to right, is responsible to obstacle avoidance. This module will scans for obstacles on all the three sides and generates an ‘obstacle found’ hormone when the sensor readings [Average of the sensor buffer] go below a set threshold. The ‘obstacle found’ hormone contains the information as to on which side the obstacle was found. The obstacle scanning module does the following,

i. If it finds an obstacle in front of the organism, then it checks if there are any obstacles on the right, if not then it generates a ‘obstacle found on left’ hormone, which makes the center pitched module rotate its hinge motors such that the organism turns towards the right side.
ii. If it finds obstacle both on the front and the right side, then it checks if there is a obstacle on the left side, if not then it generates a ‘obstacle found on right’ hormone, which makes the pitched module turn the organism towards left.
iii. If it finds obstacles on all three sides, then it generates a ‘obstacles found in front’ hormone, which gets diffused until it reaches the other end of the configuration, on receiving which the current locomotion initiating module will flip its direction variable [from forward to reverse if it is currently forward else reverse to forward], stops generating locomotion hormone and generates and diffuses a direction change hormone.

The above is for the left-end module when the organism is moving from right to left. When the organism is moving in the opposite direction, the right-end module does the obstacle avoidance task and refers to its back side distance sensor to check if there is an obstacle in front of the organism. Also, when it finds an obstacle through its right side sensor [which indicates that the obstacle is on the left side of the organism], it generates a ‘obstacle found on right’ hormone, which makes the pitched module perform the same action it does when it receives a ‘obstacle found on right’ hormone, irrespective of which side the hormone was generated from, which eventually ends up in the organism turning towards right side, avoiding the obstacle on the left.

Similarly, it generates a ‘obstacle found on left’ hormone, when it finds an obstacle through its [the module’s] left side sensor, which makes the organism turn left, avoiding the obstacle on the right.

The pitched module, on receiving an ‘obstacle found on right’ hormone, will rotate its hinge motors to an angel of ‘+45 degrees’ and rotates it back to ‘0’ degrees. This makes the upper half segment of the organism rotate towards anti-clock wise while the bottom half segment rotate clock wise, as shown in the figure 15.

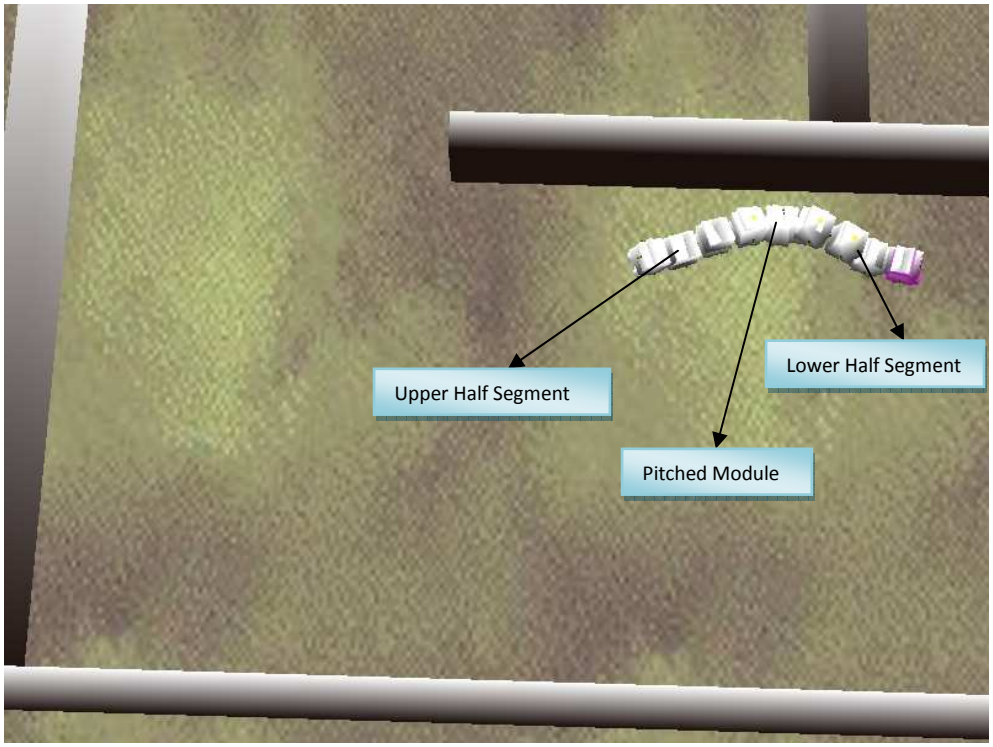


Figure 15: Caterpillar organism in turning motion.

On receiving an ‘obstacle found on left’ hormone, the pitched module will rotate its hinge motor to an angel of ‘-45’ degrees and back to ‘0’ degrees, which reverses the clock wise and anti-clock wise rotation of the upper and lower segments of the organism. Even though the two half’s of the caterpillar organism rotate in opposite directions, the organism follows the direction of its upper half segment, making the organism turn towards the left side when the upper half rotates anti-clock wise, and turn right when it rotates clock wise direction. On receiving an ‘obstacle found on front’ or ‘obstacle found on back’ hormone, the pitched module just diffuses it to its child

node without reacting to it, since in those situations the organism need to start moving in the opposite direction.

2.5.2 'V'

Obstacle avoidance in this configuration is done by the two end modules when the organism is moving in reverse direction, and by the center module when the organism is moving forward. When the organism is moving forward, with the two end modules generating the sine wave, which moves from the ends towards the center, the center module scans for obstacles on the left side of the organism with the sensor on its front, and scans for obstacles on the right side of the organism with the sensor on its right. The sensor value is again an average of the last 10 readings, but the threshold value for obstacle is different. The threshold values are hand coded based on observations made during experiments. When the center module finds an obstacle through its front sensor, which indicates that there is an obstacle on the left side of the organism, it generates and diffuses a direction change hormone with 'stop' as the direction data, through its back connector. This hormone propagates until it reaches the right-end module, on receiving which the right-end module changes its internal direction variable to 'stop' and it stops generating locomotion hormone. This result in the right segment of the organism becoming inactive while the left segment is still pushing forward, which makes the organism rotate right. The organism would rotate towards right until it avoids the obstacle on the left, and once the obstacle has been avoided, the center module generates a direction change hormone with 'forward' as the direction variable, and diffuses the same through its back connector, which results in the right-end module starting to regenerate the locomotion hormone, and the organism starts to more forward again. On finding an obstacle through its right connector, the center module, does the same direction change hormone generation but this time the hormone gets diffused through its left connector, which affects the left segment of the organism, and the organism rotates left avoiding the obstacle on its right.

When the organism is moving is the reverse direction, i.e. when the center module is generating the locomotion hormone, the obstacle avoidance task is done by the two end modules, using the sensor data from the three sensors [back, left and right]. The two end modules are of module type '8', the same as the right-end module in the caterpillar organism, so these two modules have the same control code as the right-end module in the caterpillar organism, which means that on

finding an obstacle through the back sensor, it looks if there is an obstacle on its right and then on its left and according generates an appropriate ‘obstacle found’ hormone, just as explained in the caterpillar obstacle avoidance section. But the center module does not consider the actual content of an ‘obstacle found’ hormones, it only reacts based on which of its two connectors the obstacle hormone was received from. When the center module receives an ‘obstacle found’ hormone through its left connector, indicating that there is an obstacle on the left side of the organism, then it generates and diffuses a direction change hormone with direction variable ‘Forward’ through its back connector, and stops generating locomotion hormone, making the organism rotate left. And when the center module receives an ‘obstacle found’ hormone through its back connector, which indicates that there is an obstacle on the right side, it generates and diffuses a direction change hormone through its left connector, making the organism rotate right, until it avoids the obstacle. If both the end connectors generates obstacle found hormone within a short fixed period of time, then the center module changes its internal directional variable, stops generating locomotion hormone and generates and diffuses a direction change hormone through both its back and left connectors.

2.5.3 Scorpio

While moving forward, the head module and the two outer arm modules scan for obstacles where as the outer tail module scans for obstacles when the organism is moving in reverse direction.

When the organism is moving forward, the head module scans for obstacles in front with its left distance sensor, and on finding an obstacle it changes its internal direction variable to ‘Reverse’ and generates and diffuses a direction change hormone with the same direction value, to all the modules connected to it. On receiving this, both the inner arm modules change their internal direction variable accordingly, which makes the arm push backwards, making the organism move in reverse direction.

While moving forward, the two outer arm modules scan for obstacles using all the three available sensors [front, right and left], as shown in figure 16. The right sensor of the right outer arm module and the left sensor of the left outer arm module seldom come it to picture, as they both are behind the arm and the organism is moving forward. The remaining two sensors on each outer arm module scan for obstacles as the arm swings forward and backward. When either of these modules find an obstacle through any of the three sensor, it generates and diffuses a

‘direction flip’ hormone. On receiving a ‘direction flip’ hormone, a module, if it is of the correct type, is suppose to multiply its direction variable by ‘-1’ [‘1’ represents Forward, ‘-1’ represents Reverse, and ‘0’ represents Stop], which flips the direction variable from forward to reverse or from reverse to forward. ‘Direction flip’ hormones are targeted only at inner arm modules, in this configuration, so if any other module type receives this hormone, then it just increments the hop count and diffuses it. When an inner arm module receives this hormone, it checks the hop count and reacts to this hormone if and only if the hop count is greater than or equal to 2, else it only diffuses the hormone after incrementing the hop count.

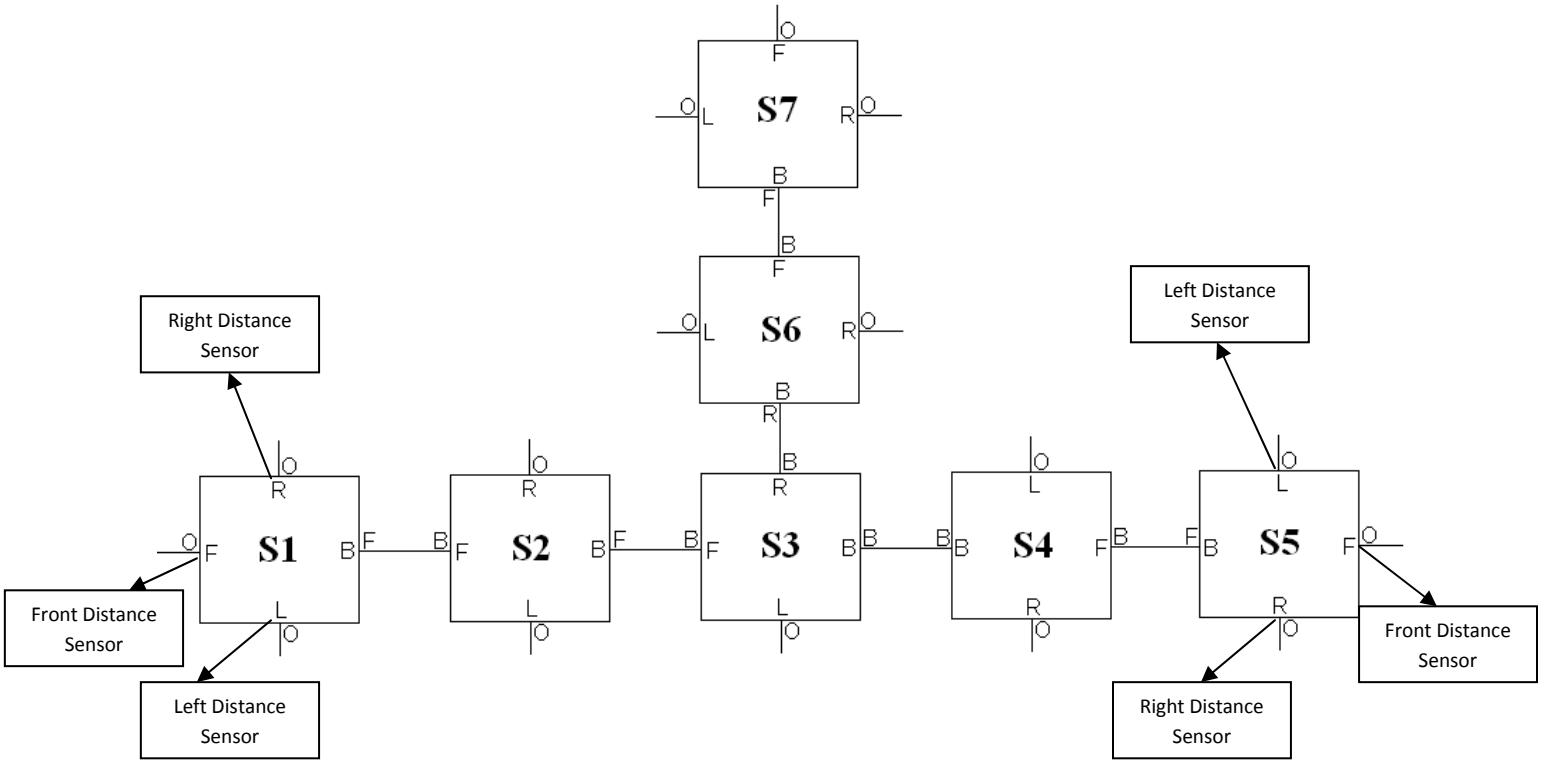


Figure 16: Scorpion configuration.

In the above figure, modules ‘S1’ and ‘S5’ are the outer arm modules, and modules ‘S2’ and ‘S4 are the inner arm modules. When module ‘S1’ finds an obstacle through one of its sensors, it generates a ‘Direction Flip’ hormone with hop count equal to zero and diffuses it to module ‘S2’. Module ‘S2’ being a inner arm module, checks the hop count of this hormone, since it is less than ‘2’, it increments the hop count to ‘1’ and diffuses it to module ‘S3’. On receiving this, module ‘S3’ increments the hop count to ‘2’ and diffuses it to its two child modules. When this hormone reaches module ‘S4’, which being an inner arm module, checks for the hop count, and

since the hop count is now ‘2’ it reacts to this hormone by flipping its direction variable. Flipping of the direction variable results in that arm now pushing backwards. With the left arm pushing backwards and the right arm pushing forward, the organism starts to rotate towards left, avoiding the obstacle on the right. Now, module ‘S1’ waits until it has completely avoided the obstacle and then generates another ‘Direction Flip’ hormone, which once again flips the direction variable of the module ‘S4’, making the left arm to pull forward, which makes the scorpion organism move forward again. Similarly, on encountering an obstacle, the left outer arm module ‘S5’ generates and diffuse a ‘Direction Flip’ hormone, which gets reacted by the right inner arm module ‘S2’, which makes the right arm push backwards while the left arm pulls forward, resulting in the organism rotating towards it right, avoiding the obstacle on its left.

The outer tail module scans for obstacles through its front sensor, when the organism is moving in reverse direction. When it finds an obstacle, it generates and diffuses a direction change hormone, with direction value ‘Forward’, which propagates throughout the network changing the direction variable of all the modules, which results in both the arms pulling forward, and resulting in the organism move forward.

3 Evaluation and Discussion

3.1 Topology Mapping

Topology mapping, both at level-0 and multi level, works fine for all the three organisms, updating the topology map as modules gets added or removed from the network. I was able to successfully test a level-8 topology map on a caterpillar organism made up of 10 modules, although I have used no more than level-2 topology map for implementing the three organisms discussed in this report. I bisected the caterpillar organism into smaller segments and observed all the modules correctly updating virtual topology map accordingly. I was able to confirm the results by printing out the content and structure of the linked list of each module and analyzing it.

3.2 Organism Length

The locomotion and obstacle avoidance algorithm written for the caterpillar organism is completely distributes and works fine irrespective of the length of the organism, although the speed of the organism decreases as the length of the organism becomes smaller, which is due to, longer organisms take bigger steps compared to smaller organisms. I was able to test a caterpillar organism with up to 16 modules [That's the maximum number of connected modules allowed by the current simulation design] with a pitched module, for rotation, at the 8^{th} position. The turning is not as sharp on a 16 module caterpillar, as it is with a 10 modules length caterpillar, but the organism still manages to turn. As the size of the caterpillar organism grows, it would probably be a better idea to have multiple pitched modules at different locations of the configuration to make the organism turn successfully. Similarly, a 'V' organism can be of any length, but the two segments have to be of equal length for the organism to move in a straight direction. If one side is smaller than the other, then the organism will tilt towards that side, as the bigger side will be faster and have a greater force vector, than the smaller side. The scorpion organism is of a fixed length and cannot grow is size.

3.3 Locomotion

The locomotion gait for all the three organisms works with varying degree of success. The fastest and the smoothest motion is produced by the caterpillar organism without a pitched module in the center. Although the organism moves very straight in this configuration, it cannot turn around without the pitched module in the center. The caterpillar configuration with the pitched module is slightly slower, less smooth, and the organism tends to very slightly tilt towards one side. It is

slower and less smooth because the pitched module does not contribute to the sine wave and its weight is carried by the other modules around it. And the tilt is due to uneven weight distribution. As you can see in figure 17, the weight of the screw driver wheels on the right makes the organism tilt slightly towards right, as it pushed forward. The turning mechanism, in this configuration, is not as sharp and accurate compared to the other two organisms. Trying to make it turn sharply, by increasing the angle of rotation of the pitched module make the organism very unstable and it becomes very vulnerable to topple towards the side. The speed and smoothness of the locomotion in caterpillar gait seems to suffer heavily when the pitched module is placed anywhere else than around about the mid section of the organism. Also, having more than one pitched module makes the motion very unstable, tilts heavily towards one side randomly, and the speed reduces drastically.

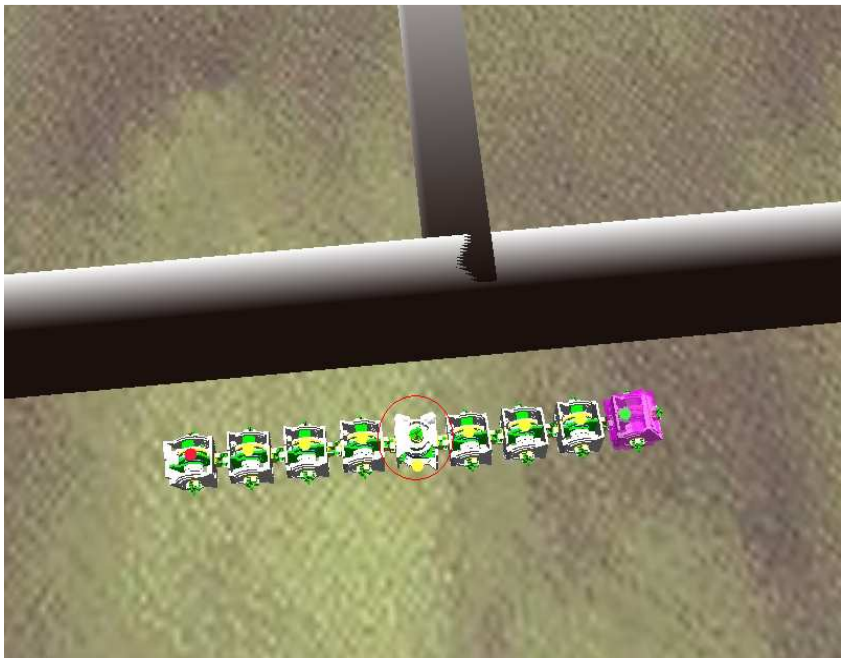


Figure 17: Top view of a caterpillar organism with the center pitched module.

The locomotion speed of the ‘V’ organism is not as fast as the caterpillar’s, the motion is fairly smooth and it moves fairly on a straight path, both when moving forward and reverse. The rotation in the ‘V’ organism is faster, sharper and more accurate compared to that of the caterpillar organism.

Scorpion organism has the slowest locomotion speed compared to the other two organisms. Although the gait is quit smooth, while moving forward and reverse, there is some deviation some times. The reason for slow speed is that there are seven modules in this configuration but there are only two modules [The inner arm modules] that pull the weight of the entire organism. I have tested the same locomotion gait by dethatching the two tail modules, and there is a significant increase in the speed and it also moves on a fairly straight path. Initially when I started experimenting with the scorpion configuration, I wanted to have a pitched module on the inner tail module, such that the turning mechanism of the organism could be controlled by its tail, and so I stuck with this configuration even though it has a slow speed. The turning in this organism is very sharp and accurate as it makes the organism rotate on its own axis, even though it is quit slowly in speed.

3.4 Obstacle Avoidance

Obstacle avoidance works best in the caterpillar organisms. Caterpillar organism can perform wall following as well as avoid any obstacle on its path while moving forward/reverse and continue moving in the same direction. Due to the sine wave gait and not very sharp turning, obstacle avoidance using distance sensors almost seems like obstacle avoidance using bump sensors, since the organism need to go very close to the obstacle before it realizes it's a obstacle.

Obstacle avoidance works fairly well in 'V' organism. It cannot perform wall following in all situations. Successful wall following depends on the angle which the organism approaches the wall. While moving forward, if it hits the wall head on perpendicularly, it pushes towards the wall until it turns towards one of the sides activating the distance sensor on that side, which makes the organism move away from the wall. Similarly if it approaches the wall perpendicularly while moving in reverse direction, it simultaneously activates distance sensor on both the end modules, making the organism move in the opposite direction. Otherwise, it performs the wall following, but as it approaches the other wall perpendicular to the current wall it is following, and hits that wall head on, it deviates away from the wall.

Obstacle avoidance in scorpion organism in much better compared to the 'V' organism but it is not as efficient as the caterpillar organism. When moving forward it can avoid obstacles and continue to move forward if the obstacle is picked up by either of the outer arm modules. When an obstacle is picked up by the head module, it is reacted to by making the organism move in

reverse direction. Just like in ‘V’ organism, the angle of approach towards the wall decides if the scorpion organism follows the wall or move away from it. When moving in reverse direction, the only way an obstacle, picked up by the outer tail module, is dealt with is by making the organism move in forward direction.

3.5 Fault Tolerant

With active topology mapping, a degree of fault tolerance is developed in to the system. When any a module in caterpillar or ‘V’ organism fails, the organism gets bisected at that point and continues to navigate. A caterpillar organism when bisected into segments of varying lengths, will resume operating as individual organism. In the ‘V’ configuration, if the two segments were to be separated by disconnecting the centre module, then the two segments become separate caterpillar organisms and start to perform locomotion and obstacle avoidance tasks just like a caterpillar organism. And before starting to operate as an individual caterpillar organism, either when bisected from a single caterpillar or when separated into two caterpillars from an existing ‘V’ organism, the locomotion initiating hormone, which is decided based on the interval direction variable, will generate and diffuse a ‘Reset’ hormone, on receiving which each module will reset its hinge motor to zero degrees, returning the organism to its initial configuration. Reconnecting two caterpillar organisms to form one big caterpillar and operating as a single organism, or connecting two caterpillars to form a ‘V’ organism should be possible, since topology mapping takes care of both active addition and active deletion of module into the network, while updating the connector status and changing the module types of the module, but I am unable to experiment these scenario since the simulation does not have the feature for connecting two modules during runtime, it only allows disconnecting modules during runtime. Disconnecting the scorpion organism’s tail from the main body converts the tail into a two module long caterpillar organism. Even though the tail recognized the disconnection, changes it module type accordingly and tries to move forward, it fails to do so since it is too small to produce a wave.

3.6 Discussion

Some improvements that can be made to the current system are,

- All the open distance sensors of all the modules are read every execution cycle, irrespective of whether it is used or not, which utilizes computational resource

unwontedly. This is programmed in the lower level, and so there is no control at the higher level to only choose those sensors to be read and stored that are used for obstacle avoidance in a given configuration. This can be improved by developing an atom for scanning the sensor values and storing it, and utilizing this atom appropriately and efficiently in the MDL2e code.

- Currently obstacle avoidance is performed based on the sensor value of only a couple of modules at a time. To extend the obstacle avoidance feature and to make it more efficient, an organism should make use of all the available sensors on all the modules. Consider for example, a moving object [like a rock or another organism] moves towards the center of a caterpillar organism. For the caterpillar organism to correctly sense this object and move away from its path, it would need to actively scan for obstacles with all its open sensors, and take action accordingly.
- In the scorpion organism, currently only the sensors on the outer tail module are used for scanning for obstacles, when the organism is moving is reverse direction. This could be extended to make use of the sensors on the fore side of the outer arm modules, using which the organism could know on which side the obstacle is and according rotate away avoiding the obstacle, instead of just moving forward when an obstacle is found while moving in reverse direction.

3.7 Summary

In this project I was able to develop a framework, based on Digital Hormone Method [DHM], for developing distributed controller for performing locomotion and obstacle avoidance task in modular robots. The main idea behind DHM is that in a multi robotic platform, each robot decides it action based on its topological position, its internal variables and sensor values, and local communication. As part of the framework, I have implemented modules for topology mapping, local communication [hormone messages], local sensing and actuation. I was able to test the same by implementing locomotion and obstacle avoidance tasks on three different modular robotic configurations, caterpillar, ‘V’ and scorpion organisms. I tested the system by simulating the configurations on a Delta 3D based physic engine simulator. The development of the framework part of the project is done in C++, where as the distributed controllers for testing the framework was implemented in a high level behavior coordination language, called MDL2e. The results of the experiment, on all the three organisms, proved that it possible to created

distributed controllers using the DHM, which leads to further investigation on using this method to create more complex behaviors in modular robotic organisms.

4 Bibliography

[1] Kernbach, S., Meister, E., Schlachter, F., Jebens, K., Szymanski, M., Liedke, J., Laneri, D., Winkler, L., Schmickl, T., Thenius, R., Corradi, P., Ricotti, L.: Symbiotic Robot Organisms: Replicator and Symbrion Projects. In: PerMIS 08, Gaithersburg, MD, USA. (2008)

[2] Schlachter, F., Meister, E., Kernbach, S., Levi, P.: Evolve-ability of the Robot Platform in the Symbrion Project. In: Workshop on Pervasive Adaptive Systems, Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems, Venice. (2008)

[3] Kernbach, S., Ricotti, L., Liedke, J., Corradi, P., Rothermel, M.: Study of Macroscopic Morphological Features of Symbiotic Robotic Organisms. In: IROS08, workshop on selfreconfigurable robots, Nice. (2008)

[4] Wei-Min Shen, Peter Will, Aram Galstyan, and Cheng-Ming Chuong. Hormone-Inspired Self-Organization and Distributed Control of Robotic Swarms. Autonomous Robots, 17(1):93–105, July 2004.

[5] Marc Szymanski and Heinz Wörn, JaMOS - A MDL2e based Operating System for Swarm Micro Robotics, IEEE Swarm Intelligence Symposium, Yuhui Shi and Marco Dorigo (Ed.), 324-331, 2007