28

Evolutionary Distributed Control of a Biologically Inspired Modular Robot

Sunil Pranit Lal and Koji Yamada
University of the Ryukyus
Japan

1. Introduction

Arguably the innovative problem solving abilities is one of the cornerstones for ensuring the survival of the homo sapien species in the game of evolution. Throughout history, when faced with challenges it was not uncommon for mankind to turn to nature for answers. In the modern day, problem solving utilizing techniques harnessed from nature has become a niche of the computational intelligence field. There are quite a number of classical contributions in this respect, which include artificial neural networks (ANN), genetic algorithm (GA), ant colony optimization (ACO), cellular automata (CA) and artificial immune system (AIS).

In the spirit of drawing inspiration from nature, our laboratory developed a modular robot modelled after a marine dwelling organism called the brittle star. The robot consists of independent modules with each module incorporating an onboard microcontroller for governing the behaviour of the module, actuator for inducing motion, and touch sensors for feeling the environment. Robot of such nature can be useful in search and rescue operations; for instance during earthquake the robot can be deployed to seek for survivors trapped under collapsed buildings which would otherwise be hazardous for human rescuers to reach.

Before novel applications of the robot can be envisioned the fundamental issues of motion control needs to be addressed. While the notion of studying the motion characteristics of the brittle star and incorporating it into the robot is intuitive and insightful, it is nonetheless quite impractical. The reason for this stems from the fact that the range of motion of the highly agile arms of brittle star in an aqueous environment overwhelmingly surpasses the two-degree of freedom legs of the landlocked robot. Hence we turn to nature once again, and attempt to draw from the evolutionary phenomena as a driving force to evolve the robot to move in its environment pretty much the same way the biological organisms have been shaped over billions of years by adapting to their environments.

Evolutionary algorithms are computational models, which capture the essence of evolution. Developed by John Holland in early 1970s (Holland, 1975), genetic algorithm (GA) is one of the widely adopted evolutionary algorithms. Inspired by biological adaptations, genetic algorithm is essentially a search technique used extensively to solve optimization related problems (Goldberg, 1989). The algorithm involves representation of candidate solutions to a problem using chromosomes also known as individuals. The initial randomly generated population of individuals is successively transformed based on their fitness by applying genetic operators

such as selection, crossover and mutation. The selection process screens the individuals such that the fitter individuals have higher probability of making it through to the next generation. The crossover operation essentially emulates the intraspecies mating by exchanging and combining genes from selected parent chromosomes to produce offsprings, with the hope that they may have better fitness than the parents. Mutation is a way to introduce diversity within the population, thus enabling better exploration of the search space. Based on the survival of the fittest, it is anticipated that with each passing generation the fitness of the individuals improves thus providing near optimum solution to the problem at hand.

In nature, organisms evolve to better adapt to their environment, which also consists of organisms from other species. Thus the interaction between the species also plays a role in shaping the evolution of organism. Co-evolutionary algorithm (Koza, 1991) is a computational model where two or more populations simultaneously evolve in a manner such that the fitness of an individual in a population is influenced by individuals from other populations. A classical example of this is the interaction between host and parasite species, whereby the host develops mechanisms to fend off the parasite, meanwhile the parasite finds ways to counter the defence mechanism of the host, thus setting off an arms race. This kind of external pressure, forces the populations to adaptively evolve thus avoiding suboptimal solutions to a large extent.

Using evolutional approach to derive motion characteristics of the robot requires a control model. In this chapter we will discuss two streams of control model namely cellular automata based control and neural network based control model.

In the cellular automata based control (Lal et al., 2006) the individual modules of the robot are modeled as cells in the cellular automata lattice. Based on this we incrementally developed control models for the brittle star robot leveraging off genetic algorithm and co-evolutionary algorithm to evolve suitable rules for each of the models.

A shortcoming of the cellular automata based approach was that the evolved rules where tightly coupled with the initial configuration of the cell lattice. To address this, the neural network inspired control model (Lal et al., 2007) was developed. In this framework each leg of the robot is modeled as a neural network with the modules representative of neurons interconnected via synaptic weights. Motion is achieved by propagating phase information from the modules closest to the main body to the remainder of the modules in the leg via the synaptic weights. Similar to the cellular automata based model, near optimal control parameters were evolved using genetic algorithm.

2. The Brittle Star: From Biological to Binary

Brittle stars [*Ophiurida*] (Hyman, 1955) are echinoderms found in most of the marine ecosystems around the world. The physical structure (Fig. 1) of the brittle star consists of five slender and flexible arms in a radial symmetry attached the central disc shaped body, which houses all of the internal organs. An internal skeleton of calcium carbonate plates referred to as *vertebral ossicles* supports the arms. These are linked by ball and socket joints, and moved by the surrounding muscle.

The brittle star moves by wriggling its agile arms to produce snake-like or rowing movement. Being able to crawling through small cracks, or being able to move even when missing an arm or two, lost perhaps when evading a predator, or being able to regenerate the lost arm or segments of it, makes the brittle star a true star in the arena of adaptive systems. This was indeed the inspiration behind developing the brittle star robot.

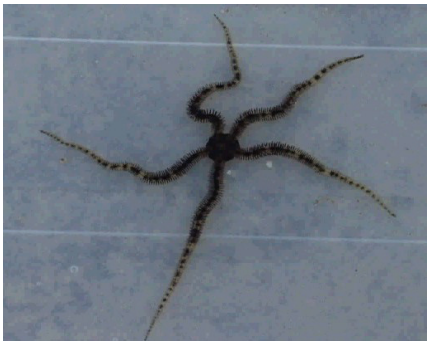


Figure 1. The brittle star

2.1 The Brittle Star Robot

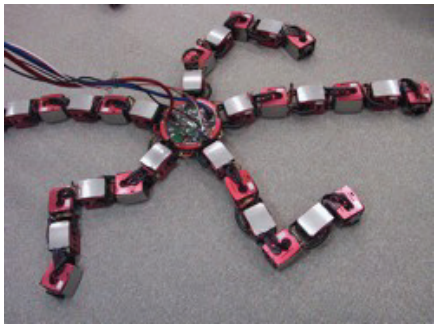
The brittle star robot (Fig. 2a) considered herein has a modular architecture consisting of modules as shown in Fig. 2b. Each module incorporates an onboard micro controller (BASIC Stamp 2sx), actuator (RC Servo Futaba S5301) and two touch sensors. In its current setup the actual robot hardware has five legs with six modules per leg.

The robot design, inspired by the brittle star has the following characteristics:

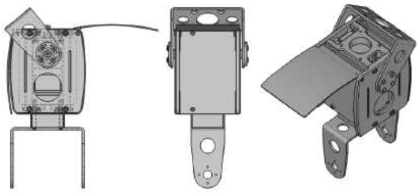
The robot has capability to move in all directions with a body shape consisting of five arms radiating from the centre.

Bone tissue of unit structure is built by connecting homogeneous modules. This allows redundancy, decentralization of control program and simplification of repair to the robot.

Modules of one degree of freedom are connected alternately in horizontal and vertical orientation (Fig. 3). In this way, a set of adjacent modules has two degree of freedom joint.



a)



b)

Figure 2. a) The brittle star robot b) Individual module connected to make up the leg

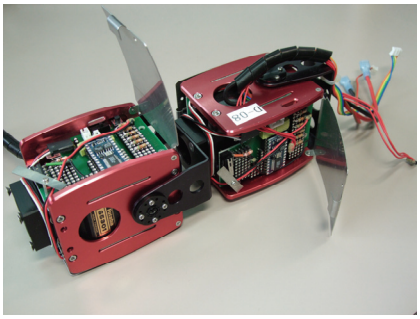


Figure 3. Exposed view of adjacent connected modules

2.2 Motion Control Problem Formulation

Moving the robot requires coordinated movement of the individual modules. Initially this was achieved through trial and error process (Takashi, 2005). Later genetic algorithm was used to better the process of finding near optimal motion pattern (Futenma et al., 2005) Though the approaches adopted in the past produced desired motion characteristics in the robot, the underlying concern was that should there be a failure of some modules the overall mobility of the robot would get compromised. To this effect, we have embarked on exploring strategies for evolving emergent global behaviour of the robot, such as walking in a straight line, from the combined action of all the functioning modules in the a given configuration.

One way to visualize the module is to consider it as a state machine. Since the motion of any given module is basically rotation between an allowable range, the state of j^{th} module in the i^{th} leg at time, t is given by (1)

$$S_{ij}^{t} \in \{\theta : \theta_{min} \le \theta \le \theta_{max}\} \tag{1}$$

This makes the robot a collection of state machines, or to put it differently, a state machine of state machines. The state of the robot with n legs and m modules per leg at time, t is thus given by (2)

$$R^{t} = \{S_{ij}^{t} : 0 \le i < n, 0 \le j < m\} \tag{2}$$

Therefore the problem of motion control of the robot becomes the task of finding optimal sequence (O_T) of state transitions, which would transform the robot producing desired locomotion.

$$O_T = R^0, R^1, R^2, \ \ldots \ , R^T \tag{3}$$

It should not take much imagination to realize the shear magnitude of the search space that needs to be explored to find near optimal solution, and thus the need for evolutionary computational approach.

2.3 Robot Simulation

While it is plausible to evolve motion controller on the physical robot itself, it is nonetheless highly impractical as doing so might damage the robot in evaluating the fitness of unknown

controllers, not to mention such an approach would be time consuming and unnecessarily cause wear and tear to the robot.
Simulated model (Fig. 4) of the brittle star robot was developed using Open Dynamics Engine (ODE version 0.9), which is an open source, high performance library for simulating rigid body dynamics (Smith, 2006). In developing the robotic simulation it is important to realize that the simulated robot is an approximation of the real robot, and as such choice of simulation parameters play a crucial role in determining if and how well the evolved control parameters can be implemented on the real robot.

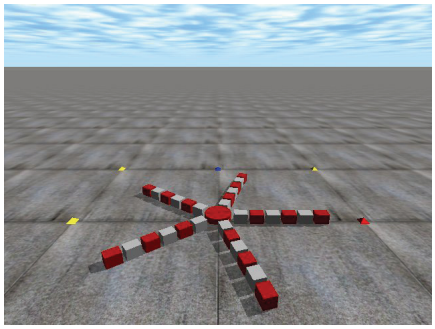


Figure 4. Simulated brittle star robot using Open Dynamics Engine

In our previous studies (Lal et al., 2006; Lal et al., 2007) we used torque as a parameter to control the rotation of the modules, which was simple to implement in ODE simulation. Later we realized that torque, as a parameter had no grounding on the physical robot because the control of the servomotor through the microcontroller involves varying pulse width to control the angle. Thus we have revamped our simulation model to centre on phase angle as the primary control parameter for the modules. It is worth mentioning that the while the control models presented in the following sections have largely remained similar to our previous work, informed choice of parameters have made significant difference in the results.
One of the parameters which influences all the control models presented in the following sections deals with the choice of number of states per module. The angular range $[-\pi/3,\pi/3]$ of the module was discretized into 16 states (0, 1, 2, ..., 15) which provided fairly granular control for the control models.

3. Cellular Automata Based Motion Control

3.1 Overview

Historically cellular automata (CA) emerged around 1940's (Wolfram, 2002) as a quest to develop computational mechanisms to resemble information processing systems in nature. John von Neumann's pioneering work in developing models for self-replicating automata (Neumann, 1966) set the stage for the future growth of this field.
Cellular automata consist of a lattice of identical finite state machines (cells) whose state changes is governed by some common transition rule. The next state of a cell at time, t+1 is determined from the current state of the cell and its neighboring cells at time t. Even the simplest rules can result in emergence of interesting patterns over a period of time, as in the case of Conway's game of life which is perhaps the best known two-dimensional cellular

automaton used to model basic process in living systems. Crutchfield et al. (2003) treatment of evolving CA using GA makes for interesting reading.
In the field of robotics cellular automata has been used to control self-reconfigurable robots, which can autonomously change their shape to adapt to their environment (Bultler et al., 2001; Stoy, 2006). Furthermore, Behring et al. (2000) has demonstrated promising use of cellular automata as a means to perform path planning using real robot in an environment clustered with obstacles.
In our approach, we modelled the individual modules of the robot as cells in the cellular automata lattice. Based on this we incrementally developed three control models for the brittle star robot. Genetic algorithm was used to evolve suitable rules for each of the models.

3.2 Singular Transition Rule for Disjoint Leg Set

In our initial attempt, the robot (with n = 5 legs and m = 6 modules per leg) was modelled as a set of disconnected one-dimensional CA lattice representing the legs of the robot and with each cell representing a module. Furthermore we decided to derive a singular CA transition rule for all the modules in the robot irrespective of their connection topology. The rational behind this was that since the rule would operate on a small neighborhood of modules, should any of the modules or legs fail, the overall mobility of the robot should not get compromised to a large extent. Furthermore, a single unifying transition rule would make the modules truly modular from both hardware and software point of view thus maintenance would be a ease in terms of replacement of modules.
Each cell in the CA framework is representative of a module and it has k = 16 possible states to represent the angle of rotation of the module. The neighborhood, η of a cell can encompass any number cells in its vicinity. The smaller the neighborhood, lesser the communication overhead per cell update. Moreover this also translates into fewer entries in the rule table. For this reason we choose the neighborhood of a cell to include only its adjacent cells, that is radius, r = 1 (Fig. 5).

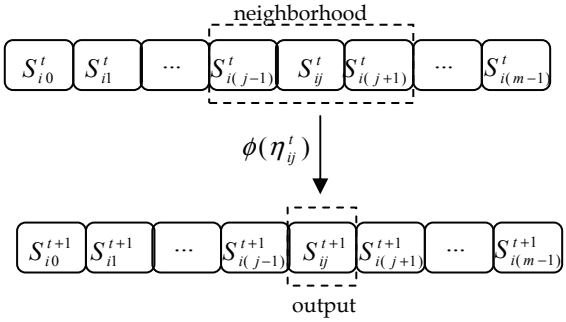


Figure 5. Lattice configuration showing states of the modules in the i^{th} leg of the robot. Radius r =1 groups 2 cells (modules) adjacent to the current cell to form a neighborhood, η which is applied as an input to the transition rule, φ(η). The resulting output is used to update the state of the current cell at time, t+1

To apply the same rule consistently on all the cells, we handle the cells on the edge of the lattice with the following boundary conditions for all u ∈{1, 2, … r}

$$S^t_{i(j-u)} = \begin{cases} S^t_{i(j-u+m)} & if\ j-u<0 \\ S^t_{i(j-u)} & otherwise \end{cases} \quad (4)$$

$$S^t_{i(j+u)} = \begin{cases} S^t_{i(j+u-m)} & if\ j+u \geq m \\ S^t_{i(j+u)} & otherwise \end{cases} \quad (5)$$

As an illustration, for 0th leg with 6 modules and neighborhood radius of 1, the neighbors for the cell S^t_{00} are S^t_{05} and S^t_{01}. Likewise the neighbors for cell S^t_{05} are S^t_{04} and S^t_{00}.

3.2.1 Genetic Encoding

There are two key pieces of information, which needs to be encoded in the chromosome; the initial states of all the modules and the rule table for working out the state transitions. With 16 states per cell (4 bits), the first 120 bits of the chromosome is allocated for the initial state of the modules. For a neighborhood with a radius r = 1 the number of entries in the rule table (Table 1) is $16^{2r+1} = 4096$, thus 16384 bits for encoding the rule table.

The rule table is encoded in a chromosome as shown in Fig. 6. The output bits are listed in lexicographical order of neighborhood. Interestingly, even though the radius is minimum possible for cell-cell interaction to take place, the rather large number of states per cell has made the search space of possible transition rules phenomenal (2^{16504}). For this reason we did not consider increasing the radius.

Index	Neighborhood			Output
0	0000	0000	0000	0010
1	0000	0000	0001	0100
2	0000	0000	0010	0000
⋮				
4096	1111	1111	1111	1101

Table 1. A sample rule table for (k,r) = (16,1)

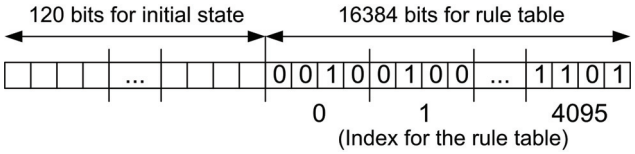


Figure 6. Representation of initial states and rule table in a chromosome

3.2.2 Fitness Function

The fitness of each chromosome is evaluated by first decoding the rule table it represents. The rule is then applied to the simulated model of the brittle star robot for successive state transitions (MAX_TRANS) thereby transforming it from initial position, (x_i,y_i) to the final position, (x_f,y_f). Our primary focus is on forward locomotion of the robot, thus the fitness of the chromosome is proportional to the Euclidean distance covered by the robot.

$$F = \sqrt{(x_f - x_i)^2 + (y_f - y_i)^2} \tag{6}$$

3.2.3 Genetic Operators

Based on the fitness of the chromosomes in the population, GA operations; namely selection, crossover and mutation are applied to whole population. Firstly selection was performed using roulette wheel selection method. In addition the best individual in a generation is automatically carried over to the next generation as per the elite selection scheme. The selected pairs of chromosomes are crossed over using one-point crossover at randomly chosen locus with a crossover probability of P_C. Finally, mutation operation involving bit flips is applied with a probability of P_M to individual genes in chromosomes after the selection and crossover operations.

3.2.4 Simulation Results

The simulation was carried out over numerous trials using parameters shown in Table 2. In each trial, initial population of chromosomes of size (POP_SIZE) was randomly generated and GA was executed for a number of generations (MAX_GEN). For each generation, the fitness of all the chromosomes in the population is evaluated after which genetic operators are applied to the population to create the next generation.

In evaluating the fitness of a chromosome, first the initial state information encoded in the chromosome is used to initialize the simulated robot, and then the encoded transition rule is applied for a number of successive transitions (MAX_TRANS) thereby transforming the cell lattice. In each transition the object model in the ODE environment is updated accordingly. The final position of the robot at the end of MAX_TRANS transitions is used in fitness calculation above (6).

The evolution of transition rules across generations is captured in Fig. 7. As can be expected the average fitness of the population increased with passing generation.

Parameter	Value
PC	0.85
PM	0.15
POP_SIZE	25
MAX_TRANS	30
MAX_GEN	750

Table 2. Summary of simulation parameters

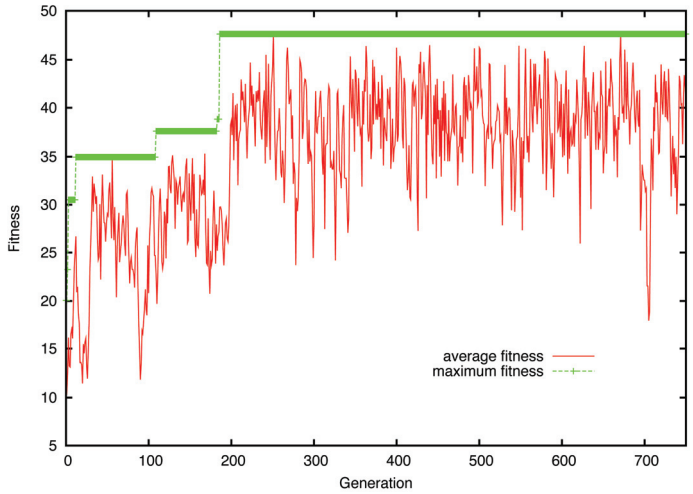


Figure 7. Graph of average and maximum fitness of population against generation for the disconnected one-dimensional CA model of the robot

3.3 Singular Transition Rule for Connected Leg Set

In this section, we extend the one-dimensional disconnected CA model by connecting modules closest to the central disc of the robot (Fig. 8) thus forming a two-dimensional structure. The notion of single transition rule for all the modules is adopted for reasons described in §3.2. Furthermore the radius describing the neighborhood of a cell is set to 1 (same as § 3.2).

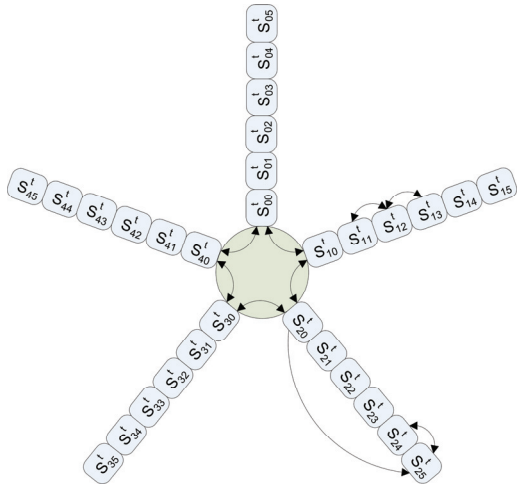


Figure 8. The two-dimensional CA lattice. The arrows represent the interaction between a cell (possibly located somewhere in the middle or on the edge of the lattice) and its neighboring cells. Essentially the interaction is similar to the model in §3.2 except modules closest to central disc interact with modules in similar position located on the neighboring leg

The two neighbors ("right", R and "left", L) for a cell at position (*i,j*) in the *n* x *m* lattice is given as follows:

$$R = \begin{cases} S^t_{(i+1)j} & if\ j = 0\ and\ i+1 < n \\ S^t_{(i+1-n)j} & if\ j = 0\ and\ i+1 \geq n \\ S^t_{i(j+1-m)} & if\ j > 0\ and\ j+1 \geq m \\ S^t_{i(j+1)} & otherwise \end{cases} \tag{7}$$

$$L = \begin{cases} S^t_{(i-1)j} & if\ j = 0\ and\ i-1 \geq 0 \\ S^t_{(i-1+n)j} & if\ j = 0\ and\ i-1 < 0 \\ S^t_{i(j-1)} & otherwise \end{cases} \tag{8}$$

Using the same procedures and parameters described in §3.2, the simulations were carried out using the revised model, the results of which is shown in Fig. 9. Notably the fitness of the best chromosome discovered was comparable to that of the previous model.

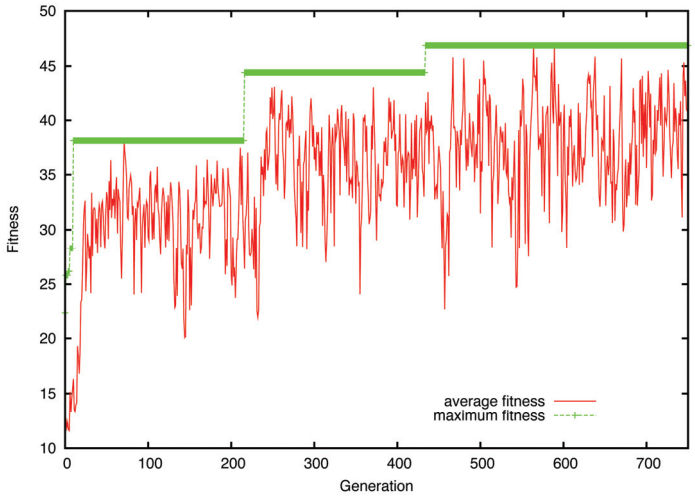


Figure 9. Graph of average and maximum fitness of population against generation for the two-dimensional CA model of the robot

3.4 Differential Transition Rule for Connected Leg Set

Building on the earlier models, in this section we present the final CA model, which represents the robot using a two-dimensional CA lattice as in the previous section, however there are two distinct transition rules for updating the cell states in the lattice. These rules are termed as control rule, *CR* and leg rule, *LR*. The position of a cell (*i,j*) in the lattice determines which of the two rules would be used to update its state and is given as follows.

$$Rule = \begin{cases} CR & if\ j = 0 \\ LR & otherwise \end{cases} \tag{9}$$

Essentially modules closest to the central disc (lead modules) are covered by control rule and modules located on other parts of the leg are taken care of by the leg rule.

3.4.1 Co-evolving the Rules

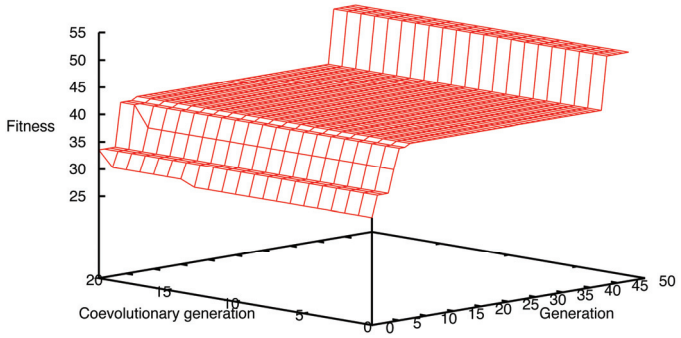
Given the differential nature of the rules we decided to employ co-evolutionary algorithm to discover the optimal rule set. Co-evolution can be competitive in a nature such as in (Rosin & Belew, 1995) where the population includes opponents trying to beat each other in a game of Tic-Tac-Toe, or it could be cooperative in nature such as in (Potter & Jong, 2000) where populations of match strings collaboratively contribute individual string to form a set of binary vectors which is to closely match a target set of binary vectors.

In our case a population of control rules and another population of leg rules is co-evolved cooperatively such that rules from both population need to be combined to control the robot. The genotype of individuals in the two population vary slightly in that the individuals in the population of control rule encode the initial state of lead modules whereas the leg rule individuals encode the initial state of the remaining (non-lead) modules. The encoding of state transition rules remain essentially the same as previous CA models. The following describes the pseudo code for the co-evolutionary algorithm.

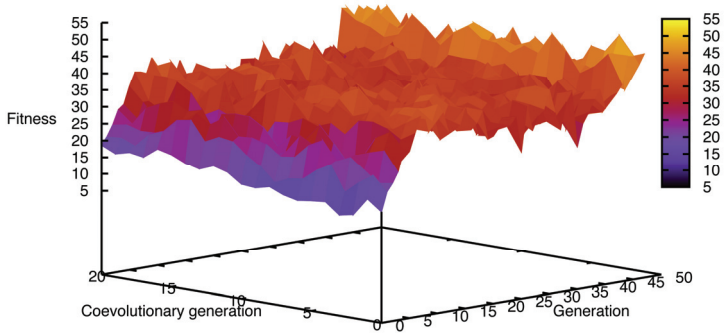
Create random population of control rules $\mathbf{P}_{CR}$ and leg rules $\mathbf{P}_{LR}$
LOOP UNTIL MAX_GEN
    Get best control rule *//Rule with the highest fitness; for initial population*
        //best rule is randomly chosen
    LOOP UNTIL MAX_COEVOL_GENERATION
        //Evaluate fitness of individual leg rules
        LOOP FOR all individuals in $\mathbf{P}_{LR}$
            Transform lattice model using best control rule and current leg rule for MAX_TRANS
            Get final robot position
            Calculate fitness, $F = \sqrt{(x_f - x_i)^2 + (y_f - y_i)^2}$
        END LOOP
        Perform GA operations on $\mathbf{P}_{LR}$
    END LOOP

    Get best leg rule
    LOOP UNTIL MAX_COEVOL_GENERATION
        //Evaluate fitness of individual control rules
        LOOP FOR all individuals in $\mathbf{P}_{CR}$
            Transform lattice model using best leg rule and current control rule for MAX_TRANS
            Get final robot position
            Calculate fitness, $F = \sqrt{(x_f - x_i)^2 + (y_f - y_i)^2}$
        END LOOP
        Perform GA operations on $\mathbf{P}_{CR}$
    END LOOP
END LOOP

The simulations were carried out again using the same parameters as in §3.2.4, additionally MAX_COEVOL_GENERATION was set to 20. The results of search for best control and leg rule pair is presented in Fig. 10 & 11. As can be seen the fitness of the controller improved compared to the previous control models. This can be attributed to the fact that the interaction between the populations within the scope of the co-evolutionary algorithm creates greater diversity thus requiring greater adaptation on the part of the individuals in the populations. Consequently the stagnation of the search algorithm at local optima that was prevalent in the previous simulations was least encountered.

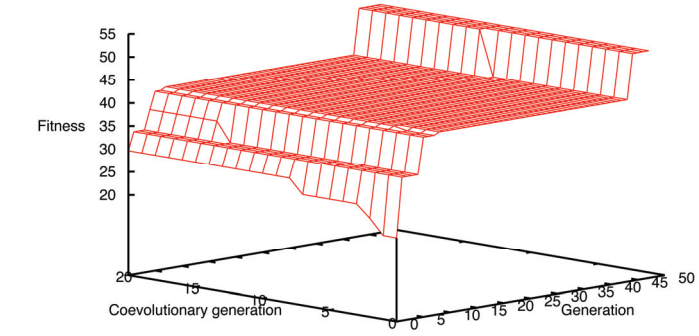


a)

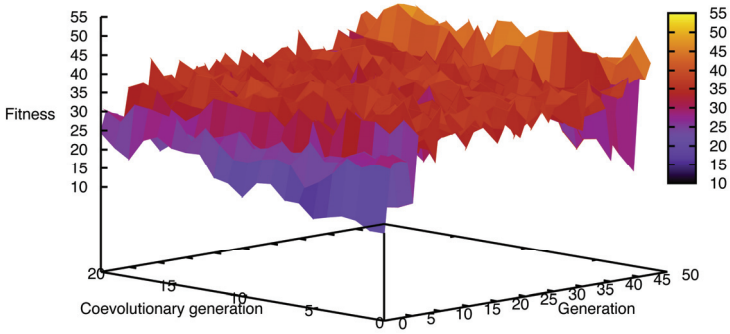


b)

Figure 10. Graph of a) maximum fitness, b) average fitness, against generation for the control rule



a)



b)

Figure 11. Graph of a) maximum fitness, b) average fitness, against generation for the leg rule

4. Neural network inspired Motion Control

While CA-based control architecture produced satisfactory motion some issues remained outstanding. In particular due to the inherent nature of cellular automata, the rules obtained were tightly coupled with the initial state of the lattice configuration. In other words if the initial phase angles of the modules is slightly changed then the resulting motion becomes incoherent. Moreover the shear size (2^{16504}) of the search space of possible transition rules made the task of learning computationally expensive. The model developed in the following sections tries to address these issues.

4.1 Overview

Inspired by the information processing ability of biological neurons, the modern day field of artificial neural networks, simply referred to as neural networks (NN) has its origins in 1943 with pioneering work by McCulloch and Pitts (1943) who developed computational model of a neuron. NN has been applied in multitude of areas namely pattern classification, function approximation, forecasting, optimization and control. While there are numerous neural network architectures in existence the fundamental computational model of the

neuron essentially remains the same. In a typical NN model the neurons are inter connected via synaptic weights. The neuron interacts by transmitting signals to other neurons connected to its output depending on the weighted sum of input stimulus to the neuron and the activation function. Learning takes place by adjusting the weights such that given an input, the output of the NN is as close as possible to the desired output. Interested readers are directed to (Jain et al., 1996) as good introductory reference material in neural networks.
In the literature many notable contributions have been made in the field of robotics and control leveraging on GA and NN. Reil and Husbands (2002) successfully simulated bipedal straight-line walking using recurrent neural network whose parameters were evolved by GA. Porting genetically evolved neural network controller for a hexapod robot from simulation model to actual hardware was demonstrated by Gallagher et al. (1996). One of the conclusions reached by them was that neural network controller performed extremely well in real world in spite of the fact that inertia, noise and delays were not taken into account in the simulation. Hickey et al. (2002) developed a system called *creeper* featuring neural network controller for producing realistic animations of walking figures. The weights for the neural network were evolved using GA wherein the fitness of a chromosome encoding the weights was related to its performance in controlling the simulated walking figure. Application of neural network is not confined to controlling just single robotic agents as demonstrated by Lee (2003) in controlling behaviour of multiagent system of simulated robots in a predator and prey type environment. Similar to other research work described above, GA was used to evolve weights for the neural network behaviour controller.

4.2 The Control Model

Neural networks are good at dealing with system parameters whose relationships are not easily deducible. Inspired by this, we decided to model the interaction between the modules using the principles of NN. It is worth mentioning that the functionality of the model we developed though similar to conventional NN has subtle differences that are explained below.
Each of the leg is modelled as a fully connected neural network (Fig. 12) with the modules represented as neurons. The modules maintain state information about its current phase angle. Furthermore the modules are interconnected via binary weights to model inhibitory and excitatory stimulus between them. Formally, $w_{ab} \in \{0,1\}$, where w_{ab} represents weight between the connection from module a to b.

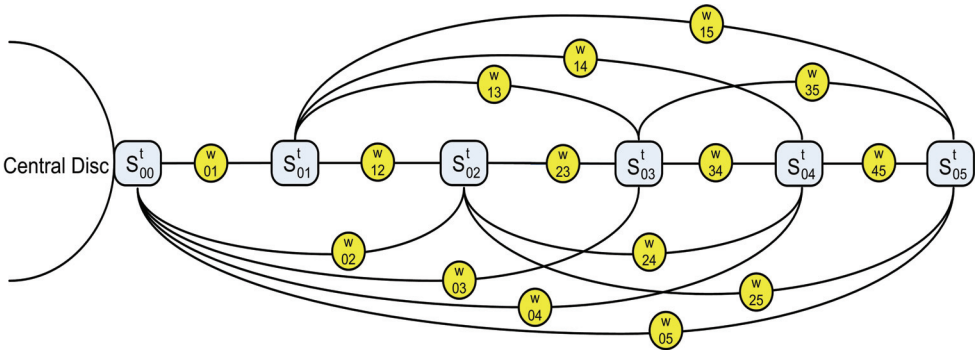


Figure 12. Conceptual framework for the neural inspired motion control architecture

Synchronized propagation of phase information through the network is used to update the current state of the modules. The modules closest to the central disc are termed as the lead modules. Starting from some initial state, the role of the lead module is to generate cyclic pattern in accordance with equation (10). Once the state of the lead module has been updated, the states of the remaining modules are then updated sequentially such that the module directly next to the lead module is updated first followed by the module next to it and so on.

$$S_{i0}^{t+1} = (S_{i0}^{t} + 1) \ \ mod \ \ k \tag{10}$$

where k is the number of states per module
The state of any given non-lead module, S_{ij} in the leg is updated as follows. First the input stimulus from the modules closest to the central disc before it is summed.

$$X_{ij}^{t+1} = \sum_{k=0}^{j-1} \phi_{ik}^{t+1} w_{kj} \tag{11}$$

where ϕ_{ik}^{t+1} is the phase angle (in radians) associated with state S_{ik}^{t+1}
Next an activation function f_a is applied to the summed input to yield the phase angle, or in other words the equivalent state information of the module in question.

$$S_{ij}^{t+1} \Leftrightarrow \phi_{ij}^{t+1} = f_a(X_{ij}^{t+1}) \tag{12}$$

To put it intuitively, equations (11) and (12) allow a module to undergo valid state transition using latest state information of modules which underwent state transition just prior to it. In summary the state transition of the modules occurs sequentially within a discrete time step.
Given the rotational nature of the modules we experimented using sinusoidal activation function (13) along with the traditional sigmoid activation function (14). It is worth mentioning that the parameters chosen for the activation function were done so as to keep the rotation of the modules in the allowable range [-π/3,π/3] as shown in Fig. 13.

$$f_a(x) = \frac{\pi}{3} sin(x) \tag{13}$$

$$f_a(x) = \frac{2\pi}{3(1+e^{-x})} - \frac{\pi}{3} \tag{14}$$

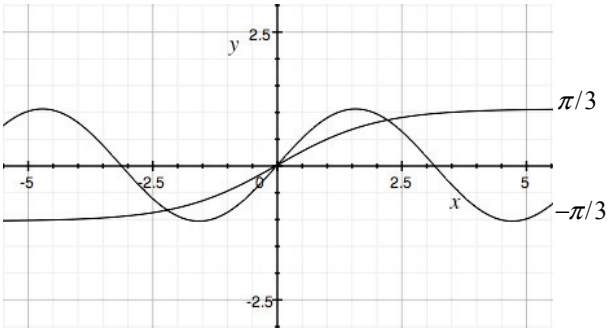


Figure 13. Graph of sinusoidal and sigmoid activation functions

4.3 Evolving Suitable Control Parameters

For the most part, GA framework was reused from §3 in determining near optimal initial states of the lead modules and the weight matrix for each of the legs. The only change to the GA framework required is encoding of the chromosome (Fig. 14). Notice unlike CA-based control model, the neural network control model requires only the initial state of the lead modules to function.

Compared to §3 the length of the chromosome has significantly decreased as it encodes 4 bits of initial state and 15 bits of weight matrix per leg. While real number could have been used for the weight matrix, in the interest of keeping search space manageable we decided just to use binary weights. Notably the search space (2^{95}) though significant is manageable in comparison with the search space for the CA-based model.

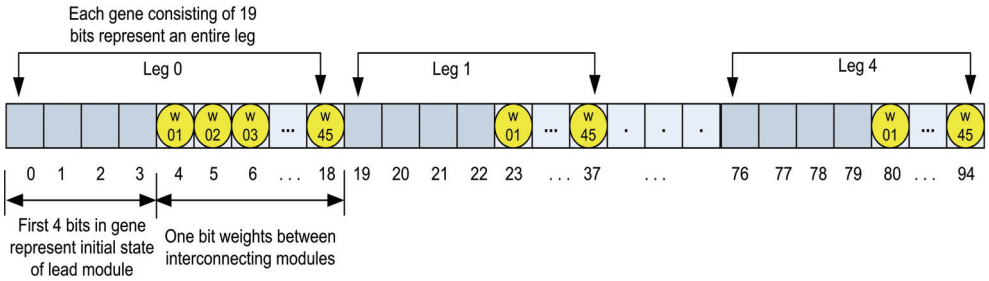


Figure 14. Genetic encoding of initial state and weight matrix

Using the similar procedures and parameters described in §3, the simulations were carried out using the revised model. The results obtained (Fig. 15, 16) indicated that the neural network model with sinusoidal activation function surpassed all the other models in terms of maximizing fitness.

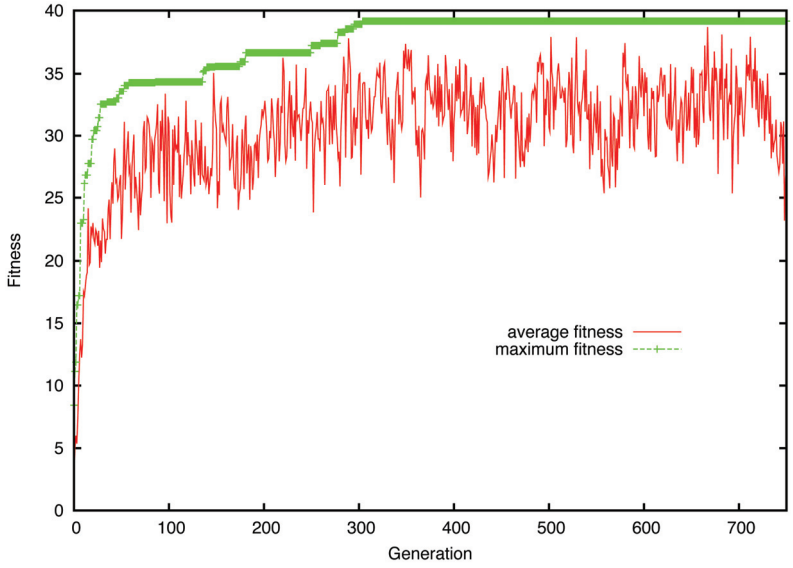


Figure 15. Graph of average fitness and maximum fitness of population against generation using sigmoid activation function

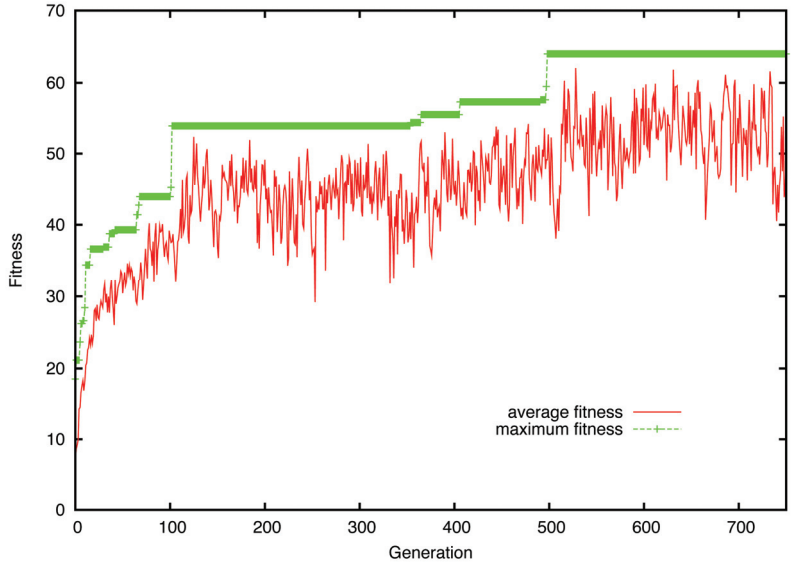


Figure 16. Graph of average fitness and maximum fitness of population against generation using sinusoidal activation function

5. Experimentation

In section we analyze the quality of the controllers evolved in terms of the motion characteristics produced and robustness of the control system in dealing with failure of modules. The following abbreviations have been adopted to refer to the controllers:

CA1d - One dimensional CA controller (§3.2)
CA2dSing – Two dimensional CA with singular transition rule (§3.3)
CA2dDiff - Two dimensional CA with differential transition rule (§3.4)
NNsig – Neural network controller with sigmoid activation function (§4)
NNsin - Neural network controller with sinusoidal activation function (§4)

5.1 Fault Tolerance

Evolving control parameters using GA is no doubt a time consuming process. Thus it would be highly desirable to have a robust control architecture, which can deal with module failures without having to be retrained.

In evaluating the robustness of the proposed models, module failure was simulated by configuring the module to be unresponsive to input stimulus thereby maintaining a fixed state. Simulation of the robot with a set of failed modules was carried out for 30 state transitions. The distance is then measured and used to calculate the degree of mobility (M_D) defined simply as:

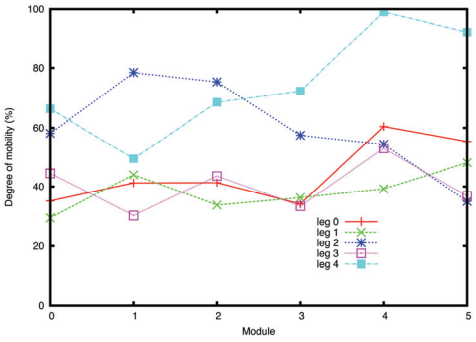
$$M_D = \frac{distance\ \ traveled\ \ by\ \ robot\ \ with\ \ set\ \ of\ \ failed\ \ modules}{distance\ \ traveled\ \ by\ \ robot\ \ without\ \ failed\ \ modules} \times 100\% \tag{15}$$

Failure was induced one by one in all the modules and the corresponding degree of mobility of the robot is depicted in Fig. 17. From the results it is apparent that the degree of mobility is greatly influenced by the position of the failed module, and the two streams of control model seem to handle failure differently. Overall the NN models performed better than CA models in the presence of module failure.

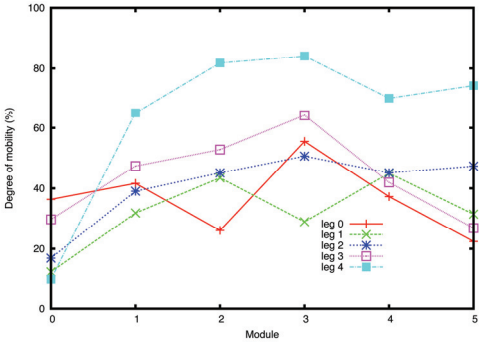
Within the CA models performance degradation was fairly distributed across the spectrum of individually failed modules. The two-dimensional models (CA2dSing & CA2dDiff) were more affected by the failure of the lead modules as it happens to be the point of connection between the legs. Overall CA2dDiff showed greater resilience compared to the other CA models.

Within the NN models the effects of failure of lead modules was noticeable. Needless to say the lead module is an important part of the control system as it provides the initial state information, and also the state transition of other modules is synchronized with the propagation of phase information from the lead module. Thus we can expect the failure of lead module would be the major contributor in hindering the overall mobility of the robot.

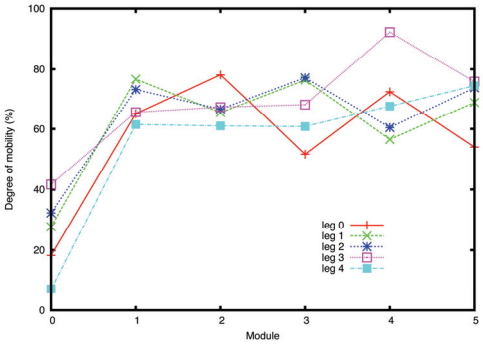
Finally, in the single module failure scenario, out of the 30 modules, 15 modules in NNsig and 8 modules in NNsin can fail with the robot still retaining 80% of its original mobility. Interestingly, for NNsin 6/30 (20%) of the modules can fail without compromising the mobility of the robot at all.



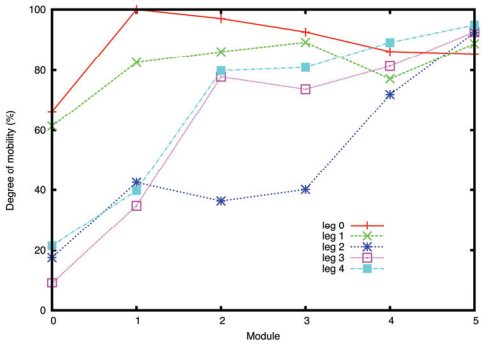
CA1d



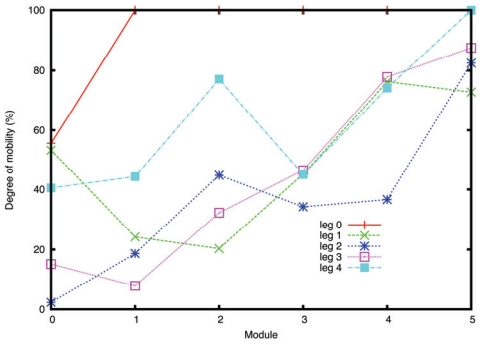
CA2dSing



CA2dDiff



NNsig



NNsin

Figure 17. Graph of degree of mobility against single module failure for all the models

5.2 Motion characteristics

Each of the controllers with best evolved parameters were run on the simulated robot for 30 state transitions, which is the same as the number of state transitions used by GA in evolving the controllers. The displacement of the central disc body of the robot from start to finish is depicted in Fig. 18. As can be seen the disc body of the robot did sway from side to side which is to be expected from robot of such nature. NNsig had the least amount of deviation on the straight-line path from start to finish. Though NNsig was the slowest of the lot, it nonetheless produced the smoothest motion.

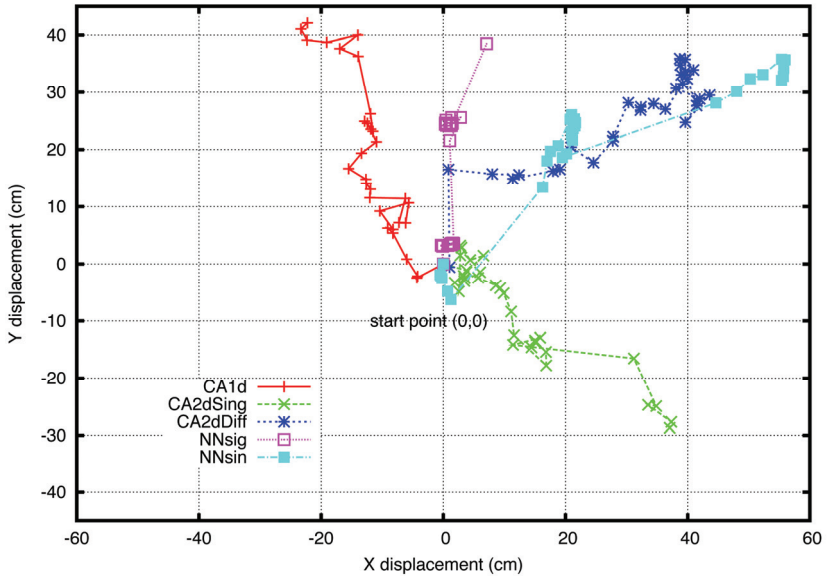


Figure 18. Displacement (cm) of the robot using each of the evolved strategies

An obvious observation is that the different controllers drove the robot in different directions, which would lead one to ponder as to how to get the robot to move in a desired direction. Well once the initial heading of the robot using a given controller has been determined, the robot can rotate about the centre of its disc body to the desired heading. An alternative to this can exploit the modular nature of the robot by swapping the control parameters between the legs and hence changing direction of motion.

In evaluating the fitness of a chromosome, the control parameters represented by the chromosome was applied to the simulated robot for MAX_TRANS = 30 state transitions. We are interested in knowing whether the results obtained can be extrapolated beyond this duration. Thus the motion of the robot using best control parameters discovered by GA was simulated beyond MAX_TRANS.

A clear demarcation in the behaviours of the CA based control models and NN based control models is captured in Fig. 19. The CA models peaked off around 30 state transitions after which their performance gradually degraded. Observations of the simulated robot using CA controller generally showed cyclic behaviour after exceeding MAX_TRANS. The NN controllers on the other hand made progress even after exceeding MAX_TRANS,

though it must be pointed out that this rate became slower in the interval after MAX_TRANS.

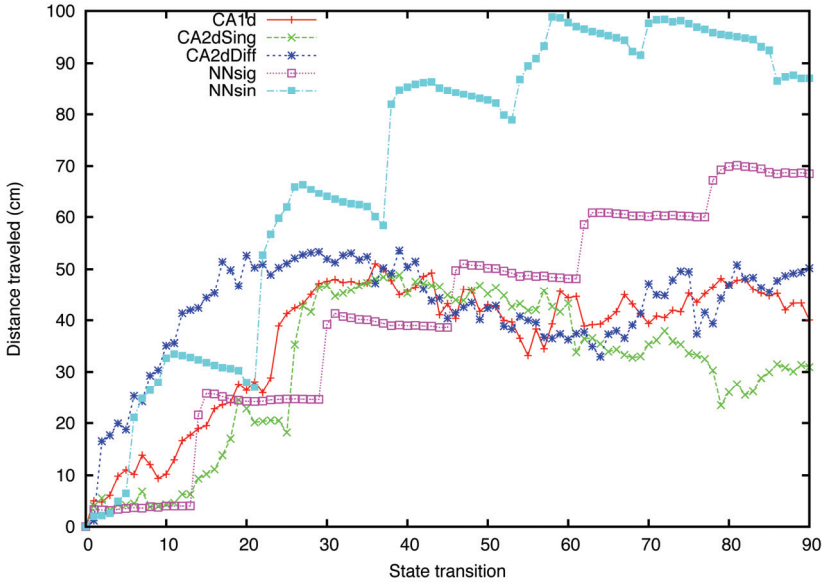


Figure 19. Graph of distance travelled against state transition using the best-evolved control parameters for each of the control models

The manner in which the controllers behave is intricately linked with the nature of evolutionary algorithm, which produced them. Genetic algorithm or for that matter any other evolutionary algorithm provide solution which is highly suited to a specific problem. In our case asking GA to evolve controllers to cover a maximum distance within 30 state transitions, will do exactly that. If we had used much larger number of state transitions, the evolved controllers except for covering greater distance, would display similar characteristics as the current controllers. One way to deal with this limitation is to let the robot move up till MAX_TRANS, reinitialize the state of the modules, and restart the state transitions from the beginning.

It is remarkable to see how the choice of activation function has influenced the strategy evolved by the neural network controller. Observations of state transition of individual module under the control of NNsig showed clustering of states, in other words given the current state of a module the probability that the next state will be the current state or the neighboring state was high. In contrast no such ascertains could be made about NNsin. The strategy employed by NNsin was quite unique compared to other controllers whose strategies resembled crawling motion. It involved coiling the legs, followed by lifting the disc body and then edging forward while uncoiling the legs and finally easing the disc body on to the ground as shown in Fig. 20.

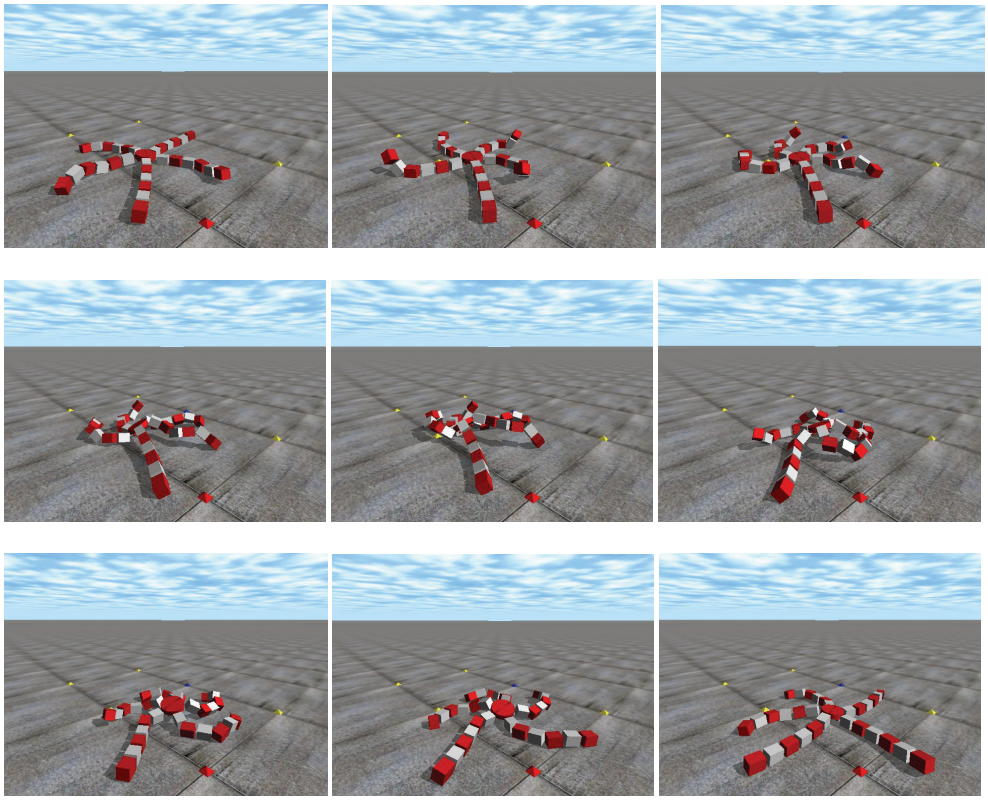


Figure 20. Snapshots of the robot's motion under the control of NNsin

6. Conclusion

The process of evolution by natural selection enables search for best solutions by adapting to the problem domain as time passes by. While we have utilized GA to find suitable parameters for our control models it is worth mentioning that from a global viewpoint the models in turn have evolved from trial and error means of motion control, to CA-based control architecture, to the neural inspired motion control model.

In light of the results, the CA2dDiff control model, whose parameters were derived using co-evolutionary approach, came out on top amongst the CA-based control models. However, the neural network based models showed better performance in terms of motion characteristics as well as exhibiting greater degree of resilience in overcoming scenarios involving failure of modules. The NN model with the sinusoidal activation function maximized the fitness function by covering the greatest distance, on the other hand the NN model with sigmoid activation function exhibited smoother motion characteristics.

Though the results are promising, this however does not imply the end of the road in the evolution of suitable controllers for the robot. There is still a lot that can be done to improve the robustness of the robot. One of the ideas is to leverage off the host-parasite analogy in coevolving a population of controllers with a population of test failure scenarios.

Evolutionary algorithms are goal driven. This makes them wonderful problem solving tools in that one has to describe the general framework of what the solution looks like and it does the rest by adaptively piecing the details. Being focused only on the end goal in terms of maximizing a fitness function is not always the best approach, as sometimes the route taken to achieving the goal matters more the end goal. In our case rather than focusing on maximizing the distance covered in a given number of state transitions, it is perhaps worthwhile exploring the nature of state transitions and seeking for specific states, which have the most influence in reaching the end goal. To this effect, we are considering markov model and reinforcement learning as future research options.

7. References

Behring, C.; Bracho, M.; Castro, M. & Moreno, J. A. (2000). An Algorithm for Robot Path Planning with Cellular Automata, In : *ACRI-2000: Theoretical and Practical Issues on Cellular Automata*, pp. 11-19, Springer-Verlag, London

Butler, Z.; Kotay, K.; Rus, D. & Tomita, K. (2001). Cellular Automata for Decentralized Control of Self-Reconfigurable Robots, *Proceedings of ICRA Workshop on Modular Self-Reconfigurable Robots*

Crutchfield, J. P.; Mitchell, M. & Das, R. (2003). The Evolutionary Design of Collective Computation in Cellular Automata, In: *Evolutionary Dynamics-Exploring the Interplay of Selection, Neutrality, Accident, and Function*, Crutchfield J. P. & Schuster P.K. (Ed.), pp. 361-411, Oxford University Press, New York

Futenma, N.; Yamada, K.; Endo, S. & Miyamoto, T. (2005). Acquisition of Forward Locomotion in Modular Robot, *Proceeding of the Artificial Neural Network in Engineering Conference*, pp. 91-95, St. Louis, Nov 2005, ASME Press, New York

Gallagher, J. C.; Beer, R. D.; Espenschied, K. S. & Quinn, R.D. (1996). Application of evolved locomotion controllers to a hexapod robot, *Robotics and Autonomous Systems*, Vol. 19, pp. 95-103

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, Massachusetts

Hickey, C.; Jacob, C. & Wyvill, B. (2002). Evolution of a Neural Network for Gait Animation. *Proceedings of Artificial Intelligence and Soft Computing*, Leung, H. (Ed.), ACTA Press, Calgary

Holland, J. H. (1975). *Adaptation in natural and artificial systems*, University of Michigan Press, Ann Arbor

Hyman, L. H. (1955). *The Invertebrates Volume IV: Echinodermata*, McGraw-Hill, New York

Jain, A. K.; Mao, J. & Mohiuddin, K. (1996). Artificial Neural Networks: A Tutorial, *IEEE Computer*, Vol. 29, No. 3, pp. 31-44

Koza, J. R. (1991). Genetic Evolution and Co-evolution of Computer Programs. In: *Artificial Life II, SFI Studies in the Science of Complexity Vol. X*, Langton, C. G.; Taylor, C.; Farmer, J. D. & Rasmussen, S. (Ed.), pp. 603-629, Addison-Wesley, Redwood City, CA

Lal, S. P.; Yamada, K. & Endo S. (2006). Studies on motion control of a modular robot using cellular automata, In: *AI 2006: Advances in Artificial Intelligence - LNAI 4304*, Sattar, A. & Kang, B. H. (Ed.), pp. 689-698, Springer-Verlag, Berlin Heidelberg

Lal, S. P.; Yamada, K. & Endo S. (2007). Evolving Motion Control for a Modular Robot, In : *Applications and Innovations in Intelligent Systems XV*, Ellis, R.; Allen, T. & Petridis, M. (Ed.), pp. 245-258, Springer-Verlag, London

Lee, M. (2003). Evolution of behaviors in autonomous robot using artificial neural network and genetic algorithm, *Information Sciences*, Vol. 155, pp. 43-60

McCulloch, W. S. & Pitts, W. (1943). A logical calculus of ideas immanent in nervous activity, *Bulletin of Mathematical Biophysics*, Vol. 5, pp. 115-133

Neumann, J. (1966). *Theory of Self-Reproducing Automata*, Burks, A. W. (Ed.), University of Illinois Press, Urbana, IL

Potter, M. A. & Jong, K. A. D. (2000). Cooperative Coevolution: An Architecture for Evolving Coadapted Subcomponents, *Evolutionary Computation*, Vol. 8, No. 1, pp. 1-29, MIT Press

Reil, T. & Husbands, P. (2002). Evolution of central pattern generators for bipedal walking in a real-time physics environment, *IEEE Transactions on Evolutionary Computation*, Vol. 6, No. 2, pp. 159-168

Rosin, C. D. & Belew, R. K. (1995). Methods for Competitive Co-evolution: Finding Opponents Worth Beating, *Proceedings of 6th International conference on Genetic Algorithms*, San Francisco, pp. 373-380

Smith, R. (2006). *Open Dynamics Engine User Guide*. [Online]. http://www.ode.org/

Stoy, K. (2006). Using Cellular Automata and Gradients to Control Self-reconfiguration, *Robotics and Autonomous Systems*, Vol. 54, pp. 135-141

Takashi, M. (2005). *Studies on Forward Motion of Modular Robot*. MSc Dissertation, University of Ryukyus, Japan

Wolfram, S. (2002). *A New Kind of Science*, Wolfram Media, Champaign, IL