A Model for Virtual Reconfigurable Modular Robots

Thomas Breton, Yves Duthen

A Model for Virtual Reconfigurable Modular Robots

Thomas Breton[1] and Yves Duthen[1]

[1]VORTEX Research Team, IRIT UMR 5505
University of Toulouse, France
{thomas.breton@irit.fr, yves.duthen@irit.fr}

Abstract

This paper presents a model for virtual reconfigurable modular robots in order to evolve artificial creatures, able of self-adaptation to the environment as well as good adjustment to various given tasks. For this purpose, a simulator has been entirely developed with the assistance of a physics engine to represent force activities. One of the most crucial points in modular robot construction is the choice of module type, complexity and diversity. We took interest on existing elements to obtain realistic results but assume simplifications to focus on our main goal that is algorithmic.

Introduction

Self-reconfiguring modular robots are one of the grand challenges of robotics (Yim et al., 2007). That points out the difficulty of such process creation at each step of development, as well in hardware as in software. First and foremost, a simulation is essential to test feasibility and evaluate performance of such modular robots, all the more so since only a few of virtual creatures have been brought to the real world.

To evolve these modular robots, rules have been defined to represent and reconfigure shapes as well as internal controller. This last one must be able to accomplish various different tasks and not only motion planning. As our main goal is to transform external structures, this objective has to be divided in a sequence of simpler tasks. Neural networks suit well these requirements and are relatively insensitive to noise; one will be used in our model. A genetic algorithm then trains the controller to emerge an adapted behavior. Afterwards, an example of a moving creature is given to show the learning ability of a modular robot with only basic oscillatory displacements, evolving together shape and motion.

As our goal is to evolve virtual modular robots, we focus on the algorithmic standpoint. That allows us to abstract from technical constraints and provides more liberty in conception. In fact, not only purely mechanical problems have to be solved but also energy management and communication methods. We propose here a model to accomplish different tasks such as motion planning, object displacement, and structure reconfiguration; evolving a controller (a neural network) by the mean of a genetic algorithm.

It is true that any consideration taken in modular robotic representation determines fundamentally the later possible description and evolution strategy. So we first present a brief modular robotics review, showing also the wide extension of this domain. Then we describe our objectives, the corresponding needs and technical concerns before detailing the whole model and its implementation. Afterwards a simplified experience is conduct and the results discussed.

Related Work

Historically, the first modular robots appear in the last 1980s with CEBOT (Fukuda et al., 1988) and several models have been built since then. CEBOT units are the very first independent modules equipped with a processor able to communicate with other ones, approach, attach and separate each other. Almost all modular robots are composed of several identical elementary components, fixed dynamically or statically. Conventional robots have often few degrees of freedom and can success to a very few specific tasks, whereas modular ones are more suited to adapt themselves to their environment and may achieve numerous problems by the means of evolutionary algorithms. Modules that compose these creatures have also a low degree of freedom but their combination exhibits complex behaviors and allows adaptation in various situations. They can be classified in several different classes: mobile (CEBOT), stochastic, lattice, chain, and hybrid. Stochastic modular robots are placed in a specified 2D or 3D environment (with other modules or/and in a substrate) and move randomly. They create structure by bonding to the substrate and/or to another element, evolving for instance in an oscillatory flow (White et al., 2005).

Lattice systems arrange modules in a two or three dimensional grid structure, simplifying greatly reconfiguration problems as the scene is a discrete world. ATRON modules

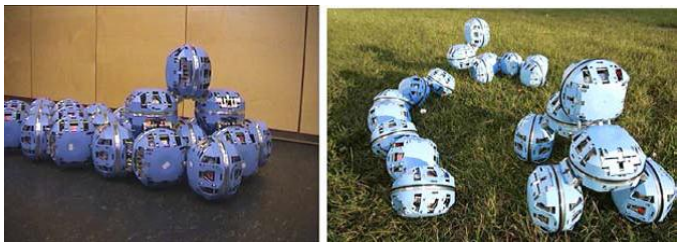

Figure 1: Left: ATRON reconfiguring. Right: three types of seven modules arrangement.

(Figure 1), developed by Stoy et al. (Ostergaard et al., 2006) are some of the most popular ones. Each module can rotate some multiple of 90 degrees in the lattice structure. On the right picture, we can see a car-like structure; this ability is an improvement of lattice based system, the system allowing continuous rotation of a module (rubber bands have been placed on the outmost of the hemisphere wheels). Furthermore, the connection mechanism is particularly robust that makes a reliable lattice system with some of the chain system flexibility.

Figure 2: A chain type modular robot composed of molecubes and grippers.

In a chain system, robots are composed of groups of connected serial chains (with possible internal loops). The modules move in a continuous space that makes many tasks such as motion planning, reconfiguration, and collision detection complex to achieve but it seems promising to future research. An example is presented above (Figure 2) from (Zykov et al., 2008). Other recent works have focused on self reconfiguration of modular robots Roombots (molecube-like with one degree of freedom more by a combination of two elementary modules) by the mean of local communication and virtual force field (Spröwitz et al., 2010). Experiments were based on handmade configuration goals in a random starting position. Several strategies have been tested and some managed to achieve the reconfiguration without any collision.

Figure 3: A six legs hybrid M-TRAN 9 modules robot.

Hybrid systems can be presented as a combination of previous ones. A great example is given by M-TRAN (Yoshida et al., 2003) modular robots (Figure 3). These modules can behave as lattice as well as chain type, allowing combining advantages of each other. Easy reconfiguration and large possibility of freedom for many tasks like locomotion. Nevertheless, motion planning for chain or hybrid modular robots is still a challenge (Teo, 2004).

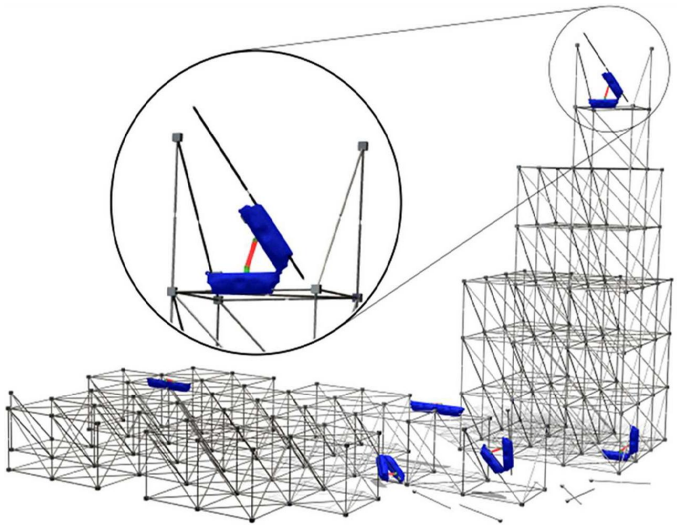

Figure 4: Model of robots reconfiguring a truss.

Another challenging domain relative to our main goal is the external structure reconfiguration. Some recent works

focused on this particular problem on truss structure reconfiguration (Figure 4); on one hand in a theoretical (algorithmic) sense (Yun and Rus, 2008; Lobo et al., 2009), on the other hand in a practical (constructible robots and truss elements) meaning (Hjelle and Lipson, 2009). The truss is represented by a tree structure and structure modification rules are defined. From a given truss configuration, the goal is to find a realizable elementary reconfiguration operation sequence to tend to a given functional goal. A genetic algorithm evolves this sequence and results have already proven physical feasibility.

Simulation

In order to develop virtual reconfigurable modular robots, our simulator has to achieve several problems. First, we attach a lot of importance to the fact that the resulting simulation must be as realistic as possible. As for, a physic model is used, taking into account gravity, static and kinetic friction, and force activities. Another important point is that evolutionary algorithms are large computer-time consumers and, at least in a first time, we will try to evolve creatures on a single computer, so the application has to be optimized in this way while preserving reliability. Furthermore, the simulator has to accomplish these tasks:

- Robot description: at each evolution step of each robot, a descriptor should define the creature shape as well as his internal controller.
- Creature evolution: a modular robot must be able to change his shape and his internal controller by the mean of different rules.
- Simulation: each robot have to be robustly simulate in a 3D space in a given trial period, no rendering is needed at this stage.
- Creature evaluation: to permit evolution, each creature is evaluated after simulation. A fitness function has to be created for each given task.
- Visual rendering: once a creature has been evolved the results have to be displayed in a three dimensional environment.

Finally, the simulator must be able to evolve a large population of creatures and allow diversity in modules. Relatively to existing open source simulators, if some are available, none fit completely our needs, even if a current development looks interesting in this way: SYMBRION and REPLICATOR (Winkler and Wörn, 2009). Some projects already successfully used them (Schmickl et al., 2009; Hamann et al., 2010). Another one, named USSR (Unified Simulator for Self-Reconfigurable Robots) and written in JAVA permits to simulate different kinds of modular robots: ATRON (Lund et al., 2006), Odin (Garcia et al., 2007; Lyder et al., 2009), and M-TRAN (Kamimura et al., 2005). It was used to simulate the combination between an ATRON and an Odin modular robots in virtual then in real world (Bordignon et al., 2008). However, developing our own platform gives us a lot of possibilities in evolution and optimization.

Technical considerations

Regarding to physics engine, its a crucial element for the simulation in terms of stability, reliability, and robustness. Some are available but they greatly differ in possibilities and performances (Boeing and Bräunl, 2007). Most popular are Open Dynamics Engine (ODE), Bullet, Nvidia PhysX, and Havok. For instance, ODE doesnt propose dynamic convex mesh (useful for collision detection), Bullet doesnt manage fixed joints (this can be done by object combination) nor kinetic friction and Havok is a commercial solution and so not attractive for our case. Tests have been realized with the first three engines and results show best performance in time computation for Nvidia Physx, just ahead Bullet.

Additionally, PhysX includes all essential functionalities and is currently the first used engine in game development. By now, the rigid bodies calculation is done by the CPUs but Nvidia announced its GPU computation in the next SDK (Software Development Kit) version (to appear this year). It should accelerate significantly performances with a dedicated adapted graphic card and let the CPUs free of charge for other tasks. However, new features are only available for Windows (XP, Vista, Seven) and no more support is given to Linux older versions from nearly two years.

Like Nvidia PhysX physics engine, our simulator is written in C++, which allows an excellent global performance and many possibilities of optimization. To keep physics engine version up to date we decide to write the code under Windows Seven rather than Linux. Nevertheless, the code is relatively easy to transfer to another operating system like Linux; main changes rely on multithreading functions. So if complex evolution strategy requires long time computation, we will transfer the code to Linux and test algorithms on a grid computing.

Model Implementation

From our main virtual modular robot goal, we will detail our implementation selection about module, and controller choices, then representation, evolution, and evaluation methods.

Global objective

The overall goal of this project is to transform external structures by virtual autonomous reconfigurable modular robots. To realize this task many points have to be considered. First

the main representation of a robot is a hybrid representation able to solve complex tasks, as described as necessary in the literature (Duro et al., 2008). Elementary modules can be fixed together to form parts able to achieve different particularly tasks, like motion or object displacement. These parts can then be attached or detached dynamically to form a more complex entity, a robot with a more global goal, such as structure reconfiguration.

Module types

One of crucial point in modular robotics development is the choice of elementary module types. In most cases modular robots are composed of only one identical type with a few degrees of freedom, linked up according to a chain or a lattice structure. It is true that evolution complexity grows up with module diversity. Our model is inspired by existing elements called molecubes (Figure 5) which have been well experimented in virtual and real life (Zykov et al., 2008). But as our goal is to test virtual complex tasks we allow ourselves some liberty in their manipulation.

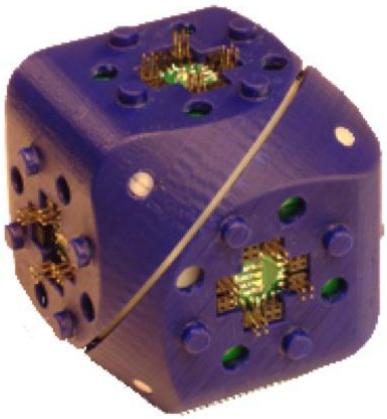

Figure 5: An elementary module: molecube.

Molecube is a cube divided in two parts, rotating around an axis defined by to opposite corners. It as then one degree of freedom and can be fixed to another module. Originally, the structure model was a chain type where a module can fix dynamically only two other elements (more if hand-connected), but we overcome this feature in our model, introducing a way to attach up four modules. To represent such an arrangement, we will use a tree structure, allowing complex configurations and avoiding loop structures. Other parts have also already been manufactured, such as wheels, grippers, digital cameras, batteries and passive modules. By now, our model only makes use of standard molecubes but is planed to accept more types, especially passive modules. Another simplification has been done relatively to module communication: as we dont consider technical specifications, we assume that all modules can communicate directly with no latency. In a same way, energy management isnt part of our preoccupations.

Internal controller

Evolving robot behavior is a real challenge in modular robotics. To carry out this achievement one can use different approaches: stimulus-response rules, neural networks or state machines (Sofge et al., 2003). For stimulus-response, continuous inputs have to be transformed in discrete outputs and it is really hard to determine transitional goals, even for simple tasks. State machines are more suited for discrete charge. Furthermore, both two approaches are very sensitive to noise, whereas neural networks offer a good alternative to each of these problems; all the more so since a large variety of input signals helps to evolve such a system, which is our case as we will see.

Computer modeling with a physics engine bring over many advantages and opportunities with few computational effort. In our creatures, a lot of internal sensors are directly provided by the physics engine and some external ones are easily computable. First, each molecube module, as describe above, is composed of two halves with a rotation axis. At each step of a simulation, Nvidia PhysX engine allows the application to get back various data comparable to internal sensors such as module position, orientation, its linear and angular velocity, torque forces and for the rotation axis the motor state, its velocity and angle of rotation. Other elements can be easily retrieved like global mass, global position, and external sensors can be thought, such as distance detector, infrared sensor, digital camera, etc. These data are then bringing in the neural network as inputs which will compute creature comportment by the means of its outputs, applying forces and angular velocity to module rotational axes. After all, the evolutionary algorithm will be ready to train the neural network.

Internal representation

At this stage, robot parts, as described above, can be fully considered as entire robots and defined by a set of elementary modules in a tree structure. They are encoded in a gene according to a simple representation with a very low memory cost but which exactly describes creature shape. First, tree structure is a depth-first one and each elementary element is encoded on two bytes (16 bits) as follow:

- Bits 12 – 15: elementary module type.
- Bits 8 – 11: orientation.
- Bits 6 – 7: currently unused.
- Bits 0 – 5: next elements.

Thus the model accepts up to sixteen elementary module types (molecube, passive module, etc.) and gives an orientation: for instance molecube can have four possible ones, according to its rotation axis direction. Next elements stand for "is linked to another module in a given direction" (from 0 to 5: front, back, up, down, left, and right). This representation permits short gene length (fixed in our model for practical and efficiency reasons). Many useful functions have been directly encoded in the depth-first tree structure, nearly all recursive but fast enough for relatively small trees. Then a robot can be fully describe and functional with a gene, a spatial position and one or more controller for a given task.

Shape evolution

In order to evolve different module configurations, several structure reconfigurations have been built, well suited to tree structures:

- Add branch: add a random branch of another creature to a random free place in the tree (cross-over).
- Remove branch: remove a random branch (mutation).
- Swap branch: swap two random branches from two creatures (cross-over) or swap two random branches of the same creature (mutation).
- Add leaf: add leaf at a random free place (mutation).
- Remove leaf: remove a random leaf of the creature (mutation).

These rules are not always applicable; the number of modules is limited (lower and upper), a sub-branch cannot swap with one of its parent, and some configurations cant be built (modules intersection), so achievability tests have to be executed for each transformation. Once mutations and cross-over have been done, the simulator can run simulations to evolve a controller.

Evaluation

Simulations are run (in parallel for multi-core computers) for a given period of time (without the need of a visual representation), after what an evaluation is given to each creature of the population, according to the given task. A fitness function takes charge to grade resulting robots, depending of its performance. Then, as commonly for genetic algorithms, creatures are classified and bests ones reused for mutation and cross-over for next generations. The simulator stops when a given number of generations and/or when a valuable fitness grade is reached. Then results are saved. After all, a visualizing tool has been built in OpenGL to ensure rendering of evolved creatures.

First Experiments

As the project is still in development and the neural network not yet implemented, very first tests have been conducted with some simplifications in order to validate other functionalities, physics engine use and in a larger domain, feasibility. In a first stage, we tried to evolve creatures with basic oscillatory movements, and without any sensor (only collisions are managed by the physics engine). The controller is then distributed on all modules (which are here independents) and some major modifications have then been realized in the model presented above:

- If the gene tree structure is preserved, a motion directive has been added to each module (encoded in gene).
- For all mutations and cross-over, the motion directives are copied out.
- Two move mutations have been added:
  - Change move: randomly modify motion directive in one or more module.
  - Smooth change move: randomly lightly modify motion directive in one or more module.

Experimentation

One of our goals was to validate this approach with very simple movements, no bias and realized on a single computer. Virtual modular robots are randomly built in a first stage, then simulated and evaluated by a fitness function. The simulation positions virtual creature on the ground and starts moving for fifteen (simulated) seconds after five (simulated) seconds of resting (to avoid falling displacements of unstable robots). Then shape and motion directives are co-evolved by mutations and cross-over of best individuals.

The motion directive is a given by the following formula:

$$v = \sum_{i=1}^{2} \alpha_i . \sin\left(\frac{\omega_i . \pi . t}{\tau} + \varphi_i\right) \quad (1)$$

Where:

- v is the velocity target applied to the rotational axis,
- α_i the amplitude,
- ω_i the frequency,
- t the time,
- τ a regulation factor,
- φ_i the phase.

And the fitness function rewards creatures that move forward, with the less side deviation:

$$\Phi = y - y_i + 0.5.|x_i - x| \quad (2)$$

Where:

- Φ is the fitness value (cube size is one unit),
- x, y the final x, y centre of mass robot position,
- x_i, y_i the initial x, y centre of mass robot position (after resting).

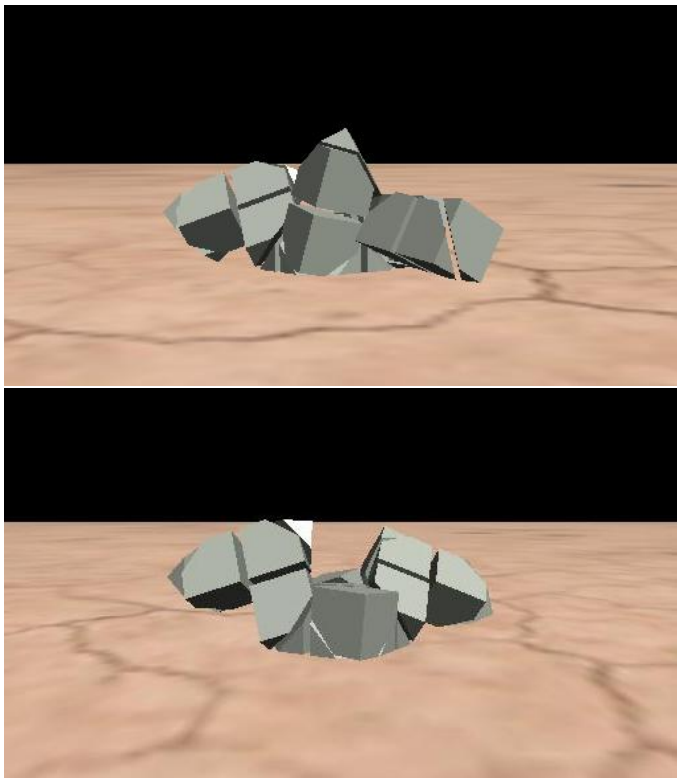

Figure 6: eight modules moving robots. Second picture shows a two legs-like motion strategy.

Results

Some other parameters have to be fixed, such as limits of forces, number of modular elements (here from eight to thirty), cross-over and mutation rates, etc. Tests have been realized on a population of 1 024 modular robots (keeping 256 bests individuals at each step) on a single computer with an Intel core i7 processor @ 3 GHz (four cores with hyper threading technology, i.e. eight virtual cores). From the 50th generation (more than half an hour of computation), bests creatures act as well and after the 70th one (about one hour of calculation) no significant improvement appears (Figure 8). A video of the best individual is available here: `http://www.youtube.com/watch?v=J_0nBlmSFHs` . However, purely random creature construction doesnt give any acceptable result in a same period of time.

We can first see that almost all best virtual creatures (Figure 6) have the lower number of elements (all in the lasts generations). It is due to two major reasons: on one hand, adding a module to a creature grows up motion directive search field significantly and then slows down convergence as same (large creatures easily deviate and are less graded than smallest ones). In the other hand, small creatures earn good graduation earlier than others, so big creatures are less and less retained for next generations.

We have also notice that the large number of parameters greatly influences convergence and its rate speed (Figure 7, Figure 9). They have here been hand-fixed with reasonably values, as describe below. For instance, a simplest sinus function gives some poorer results but keep some convergence and more sinus combination increases the search field too much and therefore decreases results quality. The resulting robot movement is in addition very sensitive to noise and light changes in a motion directive can greatly interfere on final progression. Incorporating it to mutation directives helps to obtain a swift convergence.

Technical implementation

These values are:

- The amplitude: from -4.0 to 4.0 (step 0.125)
- The frequency: form -1.5 to 1.5 (step 0.125)
- The time step is $1/60$ second
- The regulatory factor is 60
- The phase is inferior to π
- The resulting angular velocity in absolute value is then inferior or equal to $8.0 \approx 2.5.\pi\ rad/s$

At each generation, creatures are sorted by grade and 25 % of best individuals are kept with no modification. Then new genes are built from these ones with 30 % of cross-over and 45 % of mutation, dispatched as follow:

- Cross-over:
  - 25 % branch swapping (removes a branch and substitutes it by another creature one).
  - 5 % add a branch (from another creature).

- Mutation:
  - 15 % change move (randomly change from one to three motion directives).
  - 10 % smooth change move (randomly change lightly from one to three motion directives).
  - 5 % add leaf
  - 5 % remove leaf
  - 5 % move branch (internal branch displacement)
  - 5 % swap two branches (internally)

Notice that if an operation is not achievable, it is replaced by a single random move directive modification.

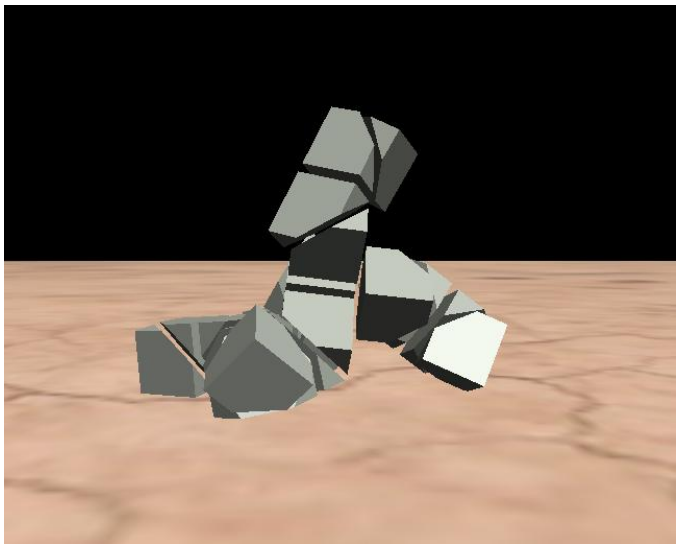

Figure 7: A ten modules moving robot, obtained after evolution, setting minimal module number to ten.

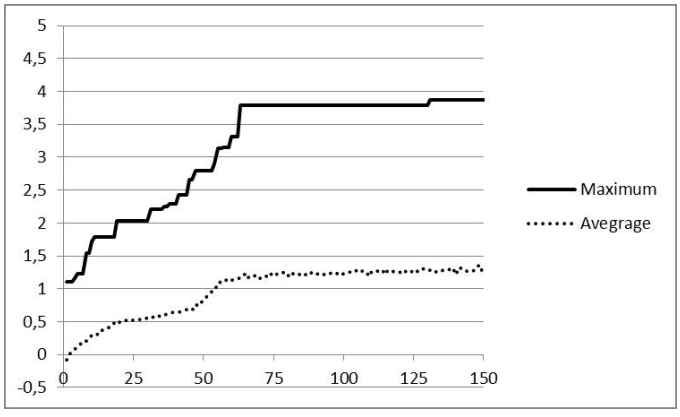

Figure 8: Fitness evolution towards generations, 1024 creature population of eight modules minimum; 5 seconds resting, 15 seconds moving.

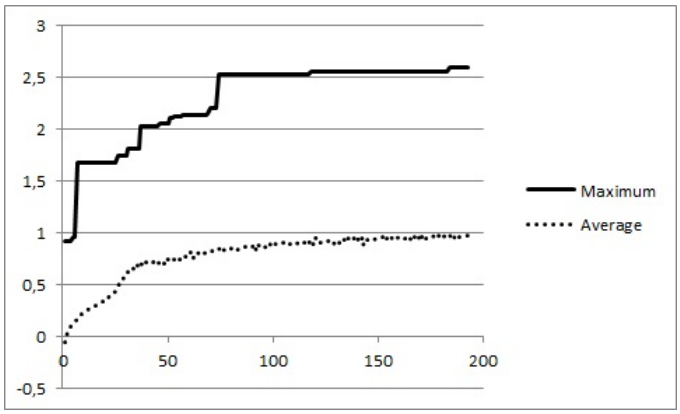

Figure 9: Fitness evolution towards generations, 1024 creature population of ten modules minimum; 5 seconds resting, 15 seconds moving.

Conclusion

In this paper we have presented a model for virtual reconfigurable modular robots, able to evolve virtual creatures in a modeled physical world. As the project is still in development we have implemented a simplified motion controller to test our virtual simulator. These very first results have shown a convergence with only basic motion directives and are then suitable enough to compare future ones with an evolved neural network controller, that greatly encourage us to pursue development in this way. Furthermore, directive movements are very sensitive to noise, a problem that neural network should reduce significantly.

Future Work

Current program expansion is a neural network encoding in order to achieve several tasks, from motion to structure reconfiguration. Virtual modular robot displacement could be placed in continuation of some older works with a new kind of evolution. In 1994, Karl Sims stepped across a new challenge, realizing virtual autonomous creatures able to learn dynamically motion planning (Sims, 1994). Some other works have then been achieve from with more sophisticated tasks, such as bloc pushing, stair climbing and skating (Lassabe et al., 2008). A review and a presentation can be found here (Duthen et al., 2011). Our project fits to these achievements but not in a similar approach: evolving virtual realistic modular robots by the mean of neural networks which should also consider collision control and be able to accomplish particular tasks like object displacement in a virtual world.

References

Boeing, A. and Bräunl, T. (2007). Evaluation of real-time physics simulation systems. In *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*, GRAPHITE '07, pages 281–288, New York, USA. ACM.

Bordignon, M., Stoy, K., Christensen, D. J., and Schultz, U. P. (2008). Towards Interactive Programming of Modular Robots. In *Proceedings of the IROS'08 Workshop on Self-Reconfigurable Robots & Systems and Applications*, Nice, France.

Duro, R. J., Graña, M., and Lope, J. (2008). On the need of hybrid intelligent systems in modular and multi robotics. In *Proceedings of the 3rd international workshop on Hybrid Artificial Intelligence Systems*, HAIS '08, pages 641–648, Berlin, Heidelberg. Springer-Verlag.

Duthen, Y., Luga, H., Lassabe, N., Cussat-Blanc, S., Breton, T., and Pascalie, J. (2011). An introduction to the Bio-Logic of Artificial Creatures. In Plemenos, D. and Miaoulis, G., editors, *Intelligent Computer Graphics*, chapter 1, pages 1–24. Springer, http://www.springerlink.com.

Fukuda, T., Nakagawa, S., Kawauchi, Y., and Buss, M. (1988). Self organizing robots based on cell structures—cebot. In *Proceedings of the IEEE/RSJ international conference on Intelligent robots and systems*, IROS, pages 145–150.

Garcia, R. F. M., Stoy, K., Christensen, D. J., and Lyder, A. (2007). A self-reconfigurable communication network for modular robots. In *Proceedings of the 1st international conference on Robot communication and coordination*, RoboComm '07, pages 23:1–23:8, Piscataway, NJ, USA. IEEE Press.

Hamann, H., Stradner, J., Schmick, T., and Crailsheim, K. (2010). Artificial hormone reaction networks: Towards higher evolvability in evolutionary multi-modular robotics. In *Proceedings of the 12th International Conference on the Sythesis and Simulation of Living Systems*, ALife XII, pages 773–780, Cambridge, USA. MIT-Press.

Hjelle, D. A. and Lipson, H. (2009). A robotically reconfigurable truss. In *Proceedings of ASME/IFToMM International Conference on Reconfigurable Mechanisms and Robots*, ReMAR 2009.

Kamimura, A., Kurokawa, H., Yoshida, E., Murata, S., Tomita, K., and Kokaji, S. (2005). Automatic locomotion design and experiments for a modular robotic system. In *IEEE/ASME Transactions on Mechatronics*, volume 10, pages 314–325.

Lassabe, N., Luga, H., and Duthen, Y. (2008). A new step for evolving creatures. In *Symposium of Artificial Life*, IEEE/ALife '07, pages 243–251.

Lobo, D., Hjelle, D. A., and Lipson, H. (2009). Reconfiguration algorithms for robotically manipulatable structures. In *ASME/IFToMM International Conference on Reconfigurable Mechanisms and Robots, 2009. ReMAR 2009*.

Lund, H. H., Beck, R., and Dalgaard, L. (2006). Self-reconfigurable robots with atron modules. In *Proceedings of the 3rd International Symposium on Autonomous Minirobots for Research and Edutainment*, volume 1 of *AMIRE 2005*, pages 5–18, Berlin, Heildelberg. Springer-Verlag.

Lyder, A., Petersen, H. G., and Stoy, K. (2009). Representation and shape estimation of odin, a parallel under-actuated modular robot. In *Proceedings of the 2009 IEEE/RSJ international conference on Intelligent robots and systems*, IROS'09, pages 5275–5280, Piscataway, NJ, USA. IEEE Press.

Ostergaard, E., Kassow, K., Beck, R., and Lund, H. (2006). Design of the atron lattice-based self-reconfigurable robot. *Autonomous Robots*, 21(2):165–183. 10.1007/s10514-006-8546-1.

Schmickl, T., Stradner, J., Hamann, H., and Crailsheim, K. (2009). Major feedbacks that support artificial evolution in multi-modular robotics. In *in Exploring New Horizons in Evolutionary Design of Robots (EvoDeRob) IROS09 workshop, ser. LNCS*, pages 63–72, Berlin, Heidelberg. Springer-Verlag.

Sims, K. (1994). Evolving virtual creatures. In *Proceedings of the 21st Annual Conference on Computer Graphics*, pages 15–22, New York, USA. ACM.

Sofge, D. A., Potter, M. A., and Schultz, A. C. (2003). Challenges and opportunities of evolutionary robotics. In *Proceedings of the 2nd International Conference on Computational Intelligence, Robotics and Autonomous Systems*, CIRAS.

Spröwitz, A., Laprade, P., Bonardi, S., Mayer, M., Möckel, R., Mudry, P.-A., and Ijspeert, A. (2010). Roombots-Towards Decentralized Reconfiguration with Self-Reconfiguring Modular Robotic Metamodules. In *Proceedings of IEEE IROS2010*, IROS, pages 1126–1132.

Teo, J. (2004). Darwin + robots = evolutionary robotics: Challenges in automatic robot synthesis. In *Proceedings of the 2nd International Conference on Artificial Intelligence in Engineering and Technology*, volume 1 of *ICAIET '04*, pages 7–13.

White, P., Zykov, V., Bongard, J., and Lipson, H. (2005). Three dimensional stochastic reconfiguration of modular robots. In *Proceedings of Robotics: Science and Systems*, Cambridge, USA.

Winkler, L. and Wörn, H. (2009). Symbricator3d — a distributed simulation environment for modular robots. In *Proceedings of the 2nd International Conference on Intelligent Robotics and Applications*, ICIRA '09, pages 1266–1277, Berlin, Heidelberg. Springer-Verlag.

Yim, M., Shen, W.-M., Salemi, B., Rus, D., Moll, M., Lipson, H., and Klavins, E. (2007). Modular self-reconfigurable robot systems: Challenges and opportunities for the future. *IEEE Robotics & Automation Magazine*, 14(1):43–52.

Yoshida, E., Murata, S., Kamimura, A., Tomita, K., Kurokawa, H., and Kokaji, S. (2003). Self-reconfigurable modular robots harware and software development in aist. In *Proceedings of IEEE International Conference on Robotics, Intellignet Systems and Signal Processing*, RISSP 2003.

Yun, S. and Rus, D. (2008). Optimal distributed planning for self assembly of modular manipulators. In *Proceedings of International Conference on Intelligent RObots and Systems*, IROS '08, pages 1346–1352.

Zykov, V., Phelps, W., Lassabe, N., and Lipson, H. (2008). Molecubes extended: Diversifying capabilities of open-source modular robotics. In *Self-Reconfigurable Robots Workshop*, IROS '08.