HydroGen: Automatically Generating Self-Assembly Code for Hydron Units

George Konidaris, Tim Taylor, and John Hallam

Institute of Perception, Action and Behaviour, University of Edinburgh.
{gkonidar, timt, jhallam}@inf.ed.ac.uk

Abstract. This paper introduces HydroGen, an object compiler system that produces self-assembly instructions for configurations of Hydron units. The Hydron is distinct from other self-reconfigurable robotic units in that it operates under water, and can thus move without being constrained by gravity of connectivity requirements. It is therefore well suited to self-assembly as opposed to self-reconfiguration, and faces similar control problems to those expected in nanotechnology applications.

We describe the first version of the Hydron Object Compiler and its supporting software. The object compiler uses a basic instruction set to produce instructions for the distributed self-assembly of any given connected configuration of Hydron units. We briefly outline the implementation of a preliminary interpreter for this instruction set for Hydron units in a reasonably realistic simulated environment, and demonstrate its operation on two example configurations.

1 Introduction

The HYDRA project aims to develop self-synthesising robot systems based on the use of simple robotic building blocks. One approach to this is the development of *object compilers*, which generate instructions for the distributed synthesis of objects from some initial configuration. Tomita et al. [9] have shown that such a process is possible in 2D using a recursive self-assembly process for some (but not all) configurations of Fractum units. More recent work by Støy [7] has shown that such a process is possible for 3D self-reconfiguration using proteo modules [10], which can be simulated using three of the HYDRA project's Atron units [5]. However, this work requires restrictions on the form of the target object in order to make local decision-making sufficient because Atron units must crawl over each other to move, and must remain connected to a configuration as it changes.

This paper therefore introduces a complementary system and accompanying object compiler for the Hydron, the HYDRA project's other primary robotic platform. Hydron units operate while suspended in water, and are thus free of the constraints of gravity. This system, called HydroGen, is therefore able to assemble objects from units dispersed in water, rather than transforming one robot configuration to another, and provides a self-assembly system complementary to Støy's self-reconfiguration system.

2 The Hydron Unit

A Hydron unit prototype is show in Figure 1. Each unit is roughly circular with a slightly narrowed equator, giving an approximate height of 12cm and width of 10cm. The Hydron is suspended in water, and actuated in the horizontal plane by four nozzles which expel water drawn through an impeller at the bottom of the unit when activated, and which are selected by a rotating collar. A syringe draws or expels water through the bottom of the unit to control unit buoyancy, and thereby actuate the unit along the vertical axis. Each unit's hull will also support a small set of switchable optical sensors and emitters capable of transmitting data over short ranges. Optical sensors and transmitters were chosen because they provide a simple and flexible underwater communication mechanism. Experiments to determine the range and effectiveness of this scheme are currently underway.

Fig. 1. A Hydron Unit Prototype

Although previous research has assumed an alternate optical sensor and transmitter placement [8], in this paper the optical sensors and transmitters are assumed to be located at each nozzle, and directly on top of and underneath the unit. Each transmitter unit is surrounded by four sensor units, forming a diamond. We assume that each sensor and emitter is active at a maximum angle of just less than $\frac{\pi}{4}$ to the normal, so that sensors at a docking site can receive only signals from emitters at a compatible docking site on another Hydron.

We term the the area around the transmitter a *docking site* or *binding site*, and it is at these sites that the Hydrons are to align themselves with each other during assembly. During the assembly process, units that wish to bind at a particular site switch on the relevant transmitter; free units move toward these lights, and two units are considered docked when their transmitter and sensor sites are sufficiently close together. The sensor and transmitter placing thus simplifies control because each quartet of sensors allow precise alignment against a particular transmitter.

At the time of writing, two prototypes with functional propulsion systems have been successfully designed and built at Edinburgh. The third prototype will include the optical communication system, and be batch produced.

Since the Hydron is still at the prototype stage, current research takes place in simulation. The simulator models the physical characteristics of the Hydron unit and its light sensors and emitters, as well as the drag properties of water, although it does not as of yet fully model its fluid dynamics. This should not present a major problem since the prototypes move fairly slowly (at an average of about 1.4cm per second), thus hopefully minimising turbulence.

One serious potential problem that will likely affect docking in the physical robots is the fact that they are unactuated about the vertical axis. They may thus obtain docks skewed by up to $\frac{\pi}{4}$ radians if they encounter turbulence that rotates them about the vertical axis. Although control code could be added to allow individual units to constantly shift their positions to compensate for the effects of turbulence and minor position fluctuation, rotation about the vertical axis may result in the deformation of the target object, and such a situation would be difficult to detect and rectify. We solve this problem in the simulator through the use of electromagnets that are switched on to affect alignment during docking, although the eventual physical solution may differ.

Because Hydron units are suspended in water and can move about freely within it, they avoid having to perform planning (e.g., [2]) or imposing structural requirements on the assembled configuration (e.g., [7]). During assembly, Hydron units do not have to move across the surface of an already constructed body – instead, they can float around until they notice an optical signal from a site searching for a binding unit and then follow this light until they reach their intended position. Although this may require sufficiently many units initially placed in useful positions to achieve rapid assembly, and thus more units than strictly necessary, it may bring a measure of redundancy to the system.

Because the units would start in a completely disconnected initial configuration, self-assembly with Hydron units would be true self-assembly, rather than self-reconfiguration. This process has more in common with morphogenesis than other models, which rely on robots which start in one (arbitrary) configuration and crawl over each other to reach a second one.

A self-assembly system using the Hydron units as a basis might then be able to draw more inspiration from and provide more insight into the developmental stages of an organism than reconfiguration approaches, thus providing a bridge between reconfigurable robotics and computational development [3], and would provide a proof-of-concept complementary to Støy's self-reconfiguration system [7] (also developed under the HYDRA project).

In addition, the knowledge gained from such a system may be useful in the future since at very small scales air is viscous, and robots employed in nanotechnology applications will likely encounter similar control problems.

3 The HydroGen Process

The HydroGen design process is depicted in Figure 2. First, either an already available CAD model of the object is converted to a "modularised" Hydron unit representation (following Støy [7]), or design takes place at the unit level.

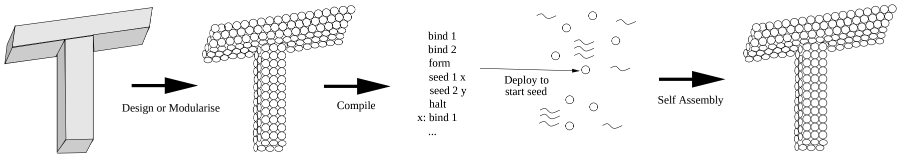


Fig. 2. The HydroGen Design Process

Given the unit representation, an object compiler then produces a list of instructions for its assembly, and these instructions are uploaded into a single initial *seed unit*, which floats in water with sufficiently many other units to complete the assembly process. This seed unit is responsible for initiating assembly and propagating the instruction list.

The code would operate under a simple seeded assembly scheme, where there are two types of Hydron units: free-floating units, and seeded units. Seeded units are already bound to the assembled structure, and can open their remaining binding sites. A free-floating unit then attaches to such a binding site, and is transformed to a seeded unit. No free-floating units execute any instructions until they dock and become seeded, whereupon the instruction list is transmitted along with an instruction label to begin execution with. Control and instruction propagation are thus both completely local and distributed, with the instructions specifying which binding sites each unit should open, and which instruction numbers the units bound to each should be seeded with. Thus, each unit carries an entire copy of the target representation, and the target is built using the progressive transfer of assembly position between units, as in Tomita et al. [9], resulting in a recursive assembly process.

A basic instruction set capable of accomplishing this is given in Table 1, along with a brief description of each instruction's function. In addition to the list given in Table 1, instructions may be given labels which are used as the second parameter to the `seed` instruction.

These instructions are clearly sufficient to reach any connected object configuration, provided that some Hydron unit can reach each binding site where necessary. In natural systems, cells and structures can be grown where they

Table 1. The Basic Hydron Construction Set

`bind` x	Bind another Hydron unit to site x.
`form`	Wait for all sites to complete binding.
`seed` x y	Transform the Hydron bound at site x to a seed, starting execution there at the instruction labelled y.
`halt`	Stop processing at this unit.

are required, but in a system consisting of seeded and free-floating Hydron units, a bind point may become unreachable because the path to it from all points outside of the seeded region is blocked.

One way to solve to this would be to enforce a breadth-first ordering on binding and seeding; however, for some combinations of structure, seed choice and controllers, blocking may still occur. The `form` instruction has been included as a point where some form of synchronisation behaviour can take place if necessary. This could take the form of signal propagation (where units that have not completed forming broadcast a growth suppression signal), or a simple timed seeding pulse.

In later work, we intend to expand the instruction set to potentially include explicit signal and gradient propagation, variable access instructions, etc., outlined in section 6.

4 Generating Self-Assembly Code

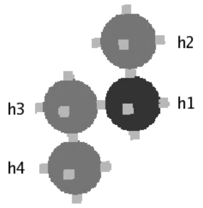
The object compiler is the heart of the HydroGen process. This section describes the first HydroGen compiler and its supporting programs.

The Hydron Object Designer provides an intuitive interface allowing users to construct objects out of Hydron molecules, by manipulating the user's viewpoint and adding and removing individual units. Although object descriptions will eventually likely be discretisations of CAD object models (as in Støy [7]), an object designer that works at unit level is initially a more useful prototyping tool. The Object Designer produces a description of the object as a simple list of Hydron coordinate triplets, suitable as input to the object compiler.

The Hydron Object Compiler then generates a set of self-assembly instructions that can be used to build the required structure in a distributed fashion. These instructions are required to be interpreted either by a control program running on an individual Hydron unit, or by the Hydron Object Interpreter, described below.

The compiler generates the instruction list as follows. First, the Hydron coordinate list is read into memory, and sorted on x coordinate, breaking ties first on y and then on z coordinate values. The first unit in the unit list is then placed in a unit queue, and marked as considered. The algorithm enters a loop that removes the first unit in the queue, and performs a binary search for each of the coordinate tuples that would be occupied by units docked at each site to determine which ones are present. Each unit present at a binding site generates a `bind` instruction; those that have not been marked as considered generate `seed` instructions and are added to the end of the unit queue. A `form` instruction is placed between the unit's list of `bind` and `seed` instructions and a `halt` instruction is placed at the end of the unit's instruction list. This list is then given the label hp and output, where p is the unit's position in the input list, and the loop repeats. The entire compilation process takes $\Theta(n \log n)$ time, where n is the number of Hydrons in the configuration, and $\Theta(n)$ space.

The use of a queue rather than simple recursive method for code generation results in an instruction list that seeds units in breadth-first order, with the aim of increasing the amount of binding that can occur in parallel without synchronisation. We consider this a good compromise between allowing for unordered growth, which maximises the amount of parallel docking possible but potentially blocks many docking sites, and a serial docking schedule, which removes the potential for parallel construction but allows for a completely predictable development process. The controller implementation could either rely on a breadth-first growth order happening naturally because of this, or employ a synchronisation method to ensure it, as in Tomita et al. [9].




```
h1:  bind 1         h3:  bind 0
     bind 5              bind 4
     form                form
     seed 1 h2           seed 0 h4
     seed 5 h3           halt
     halt           h4:  bind 1
h2:  bind 0              form
     form                halt
     halt
```

Fig. 3. A Sample Configuration and its Self-Assembly Code.

A simple object configuration (with the seed unit (h1) in darker gray than the others) along with the resulting compiler output is given in Figure 3. At present the compiler generates instructions for each unit, and since no code optimisation is performed, the instruction list length is proportional to the number of units in the target configuration.

The Hydron Object Interpreter is used to verify the code produced by the the Object Compiler, or to rapidly evaluate the object configurations produced by some other process (e.g., a genetic algorithm) without requiring a physically realistic Hydron unit simulation.

The interpreter starts with the seed unit and (if necessary) creates a Hydron with the required coordinates whenever a `bind` instruction is executed. Figure 4 shows four frames of the development of a simple Hydron structure in the Object Interpreter.

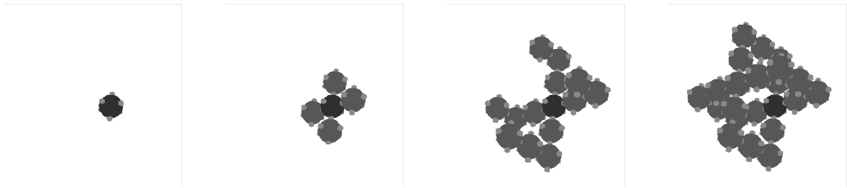


Fig. 4. The Object Interpreter Assembling a Hydron Structure

5 Self-Assembly in a Physically Realistic Simulator

This section outlines a preliminary implementation of the HydroGen instruction set in a reasonably realistic physical simulator. The implementation does not tackle the problem of breadth-first seeding co-ordination, or the problem of multiple light sources confusing a free-floating unit. In addition, it assumes that seeded modules bound to the existing structure are sufficiently rigidly attached as to not be knocked away by docking modules.

Here, free-floating modules pick the quartet of optical sensors with the highest overall reading, and use a proportional control scheme (with water resistance providing a differential damping component) to align the appropriate docking port with the light source by attempting to equalise the readings across the sensor quartet. Experience with the simulator indicates that this control method reliably brings the floating unit sufficiently close to the signalling unit for it to activate the appropriate electromagnet and complete the dock. The `form` instruction does not complete until all of the required binding sites have docked. This implementation is sufficient for constructing some structures but will fail when the structure has a hole in it because some docking sites will be blocked.

Figure 5 shows two assembly sequences. In the first, six free-floating Hydrons bind to all the sites on a seed Hydron, demonstrating docking and simple structural formation. This sequence required approximately 1030 simulated seconds.

The second sequence shows a slightly more complex configuration in development. Although in this case the Hydrons had to be placed so that they did not physically interfere with each other, the instructions are clearly being propagated, and the structure was assembled correctly in 3600 seconds.

6 Future Work

The first version of the Hydron Object Compiler system presented here is intended to form the basis for further work that will further develop the system using ideas from compiler optimisation and cell biology. This section outlines the major directions that future research is expected to take.

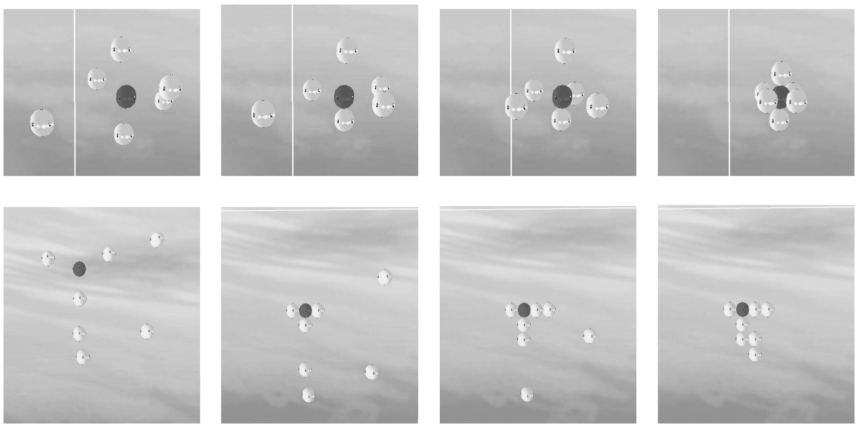


Fig. 5. Simulated Self-Assembly Sequences

6.1 Hardware and Simulation Implementation

Although the implementation given in section 5 can perform rudimentary assembly, it is not yet complete. It does not implement any form of breadth-first signal propagation, nor does it tackle the problems that could occur when two adjacent Hydrons switch on transmitters pointing in the same direction (although a form of local signal suppression would probably be sufficient here).

More importantly, it cannot reach all connected object configurations, because of the simplistic method used for Hydron docking and the narrow range of the Hydron's sensors. Other controller implementations (e.g., where a binding site notices that its way is blocked, lights up an orthogonal docking site, and instructs an arriving Hydron to move around) may solve this problem, or it may require preemptive blockage handling by the compiler; we expect it to require some combination of the two.

Further work is thus required to develop methods that can assemble all HydroGen instruction sequences in real and realistically simulated robots. The development of these methods and the further research detailed below (which is concerned exclusively with the compiler) will proceed concurrently, since the two processes are mutually informing.

6.2 Growing Disconnected Components using Cell Death

Another immediate limitation of the basic instruction set is that it cannot be used to assemble objects that consist of two or more disconnected pieces. One way to resolve this would be to construct the object with extra scaffolding units connecting the separate components of the object. A form of timed (or programmed) cell death, which occurs during development in natural systems for fine feature formation [4], could then be used to remove the scaffolding units once assembly is complete. The system could then reach any Hydron unit construction, provided the scaffolding units could escape from the configuration or perhaps find somewhere else to bind. This could also be used to solve blocking conflicts by initially construction portions of the object as solid and then removing Hydrons to obtain the intended structure.

6.3 Variables and Common Code Segments

The code generated by this version of the object compiler is linear in the size of the target configuration. One way to reduce code size and increase modularity would be to reuse of common code segments [1] and use variables to express repeated structures. This would require the addition of simple branching and arithmetic operators to the instruction set and extra functionality to the compiler, but would result in more concise code.

6.4 Obtaining Symmetry through Cellular Gradients

Another way to increase the expressiveness of the instruction set would be the use of a cellular gradient to specify binding site numbering. For example, if the start seed sent out a gradient and the other units numbered their binding sites starting at the site receiving it, then common code would produce radial symmetry. This would not require major changes to the compiler but there may be other interesting symmetry generating mechanisms (e.g., having multiple symmetric origins) that would require more instructions and further compiler functionality.

6.5 Evolving Hydron Configurations

Finally, it would be useful to attempt to bridge the gap between explicitly designed structures and those developed by evolutionary techniques, especially since the instruction set augmented with the additions described above represents an expressive genomic language. The use of an Object Interpreter may also allow for extremely fast genome evaluation, while preserving the ability of evolved solutions to be expressed in realistic environments.

Another possibility might be the translation of either the basic instruction set or a later variant of it to a Genetic Regulatory Network [6, 8]. The HydroGen system could then be used to seed such systems, rather than requiring evolution to start from scratch.

7 Summary

This paper has introduced the HydroGen object compiler, a system that produces instructions for the self-assembly of Hydron unit configurations, and briefly described the Hydron unit. It has also presented the first implementation of the Hydron Object Compiler, supporting a basic instruction set capable of expressing any connected unit configuration, and a controller that demonstrates that the system is capable of assembly in a reasonably realistic simulator. This system is intended as a platform for the further development of self-assembly methods, through the future research areas given in section 6. The aim of this research programme is to shed some light on the characteristics and functional requirements of natural and synthetic cellular self-assembly systems.

Acknowledgments

This work forms part of the HYDRA project (EU grant IST 2001 33060, `http://www.hydra-robot.com`). Facilities for this research were provided by the Institute of Perception, Action and Behaviour in the School of Informatics, University of Edinburgh. We would also like to thank the anonymous referees for their helpful comments and suggestions.

References

1. A.V. Aho, R. Sethi, and J.D. Ullman. *Compilers: Principles, techniques and tools.* Addison-Wesley, 1986.
2. K.D. Kotay and D.L. Rus. Algorithms for self-reconfiguring molecule motion planning. In *Proceedings of the International Conference on Intelligent Robots and Systems*, 2000.
3. S. Kumar and P.J. Bentley. An introduction to computational development. In S. Kumar and P.J. Bentley, editors, *On Growth, Form and Computers*, chapter 1, pages 1–44. Elsevier, 2003.
4. H. Lodish, A. Berk, S.L. Zipursky, P. Matsudaira, D. Baltimore, and J. Darnell. *Molecular Cell Biology.* W.H. Freeman & Co., New York, NY, 4th edition, 1999.
5. E. Østergaard and H. H. Lund. Evolving control for modular robotic units. In *Proceedings of the 2003 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, Kobe, Japan, 2003.
6. T. Reil. Dynamics of gene expression in an artificial genome – implications for biology and artificial ontogeny. In D. Floreano, J.-D. Nicoud, and F. Mondada, editors, *Proceedings of the Fifth European Conference on Artificial Life (ECAL99)*, 1999.
7. K. Støy. Controlling self-reconfiguration using cellular automata and gradients. In *Proceedings of the 8th International Conference on Intelligent Autonomous Systems (IAS-8)*, March 2004.
8. T. Taylor. A genetic regulatory network-inspired real-time controller for a group of underwater robots. In *Proceedings of the 8th International Conference on Intelligent Autonomous Systems (IAS-8)*, March 2004.
9. K. Tomita, S. Murata, H. Kurokawa, E. Yoshida, and S. Kokaji. Self-assembly and self-repair method for a distributed mechanical system. *IEEE Transactions on Robotics and Automation*, 15(6):1035–1045, December 1999.
10. M. Yim, Y. Zhang, J. Lamping, and E. Mao. Distributed control for 3D metamorphosis. *Autonomous Robots*, 10(1):45–56, 2001.