Elements of Autonomous Self-Reconfigurable Robots

Christensen, David Johan

Publication date:
2008

Document version
Final published version

Citation for published version (APA):

Terms of use
This work is brought to you by the University of Southern Denmark through the SDU Research Portal. Unless otherwise specified it has been shared according to the terms for self-archiving.
If no other license is stated, these terms apply:

• You may download this work for personal use only.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying this open access version

If you believe that this document breaches copyright please contact us providing details and we will investigate your claim.
Please direct all enquiries to puresupport@bib.sdu.dk

Download date: 25. Oct. 2019
Elements of Autonomous Self-Reconfigurable Robots

Author: David Johan Christensen

Supervisors: Prof. Henrik Hautop Lund
Associate Prof. Kasper Støy

The Maersk Mc-Kinney Moller Institute
University of Southern Denmark

June, 2008
Cover picture is a collage of an ATRON structure, an octopus arm and a fractal from the Mandelbrot set.
Abstract

In this thesis, we study several central elements of autonomous self-reconfigurable modular robots. Unlike conventional robots such robots are: i) Modular, since robots are assembled from numerous robotic modules. ii) Reconfigurable, since the modules can be combined in a variety of ways. iii) Self-reconfigurable, since the modules themselves are able to change how they are combined. iv) Autonomous, since robots control themselves without human guidance. Such robots are attractive to study since they in theory have several desirable characteristics, such as versatility, reliability and cheapness. In practice however, it is challenging to realize such characteristics since state-of-the-art systems and solutions suffer from several inherent technical and theoretical problems and limitations. In this thesis, we address these challenges by exploring four central elements of autonomous self-reconfigurable modular robots: design, scalability, self-reconfiguration and adaptation.

The first element we consider is the design of systems, modules, robots, and behaviors. We introduce a number of design principles that will guide our designs throughout the thesis. The design principles advocate simple, extendable, heterogeneous systems, where the robot’s behavior emerges from autonomous modules controlled in a distributed fashion. The second element considered is scalability in terms of size and number of modules. We study the interdependence between morphology, module size and behavior and observe how none of these aspects can be studied in isolation. To facilitate scalability we propose a module organization inspired by the anatomy of biological organisms, which allows reuse of module structures and control from one robot to the next. The third considered element is the process of self-reconfiguration. To fulfill the goals of scalability and fault-tolerance, we propose a distributed strategy based on meta-modules that emerge from the structure of other modules. The fourth and final element considered is adaptation, which we study in the context of locomotion. Our approach is distributed by having each module learn its own function in isolation from other modules. We study how adaptive, configuration independent and fault-tolerant collective behaviors emerge at the level of the robot.
Danish Resume


Acknowledgements

First, I would like to thanks to my main supervisor Henrik H. Lund for being a strong driving force. Then, I would like to thanks my project supervisor Kasper Støy for his creativity and for being a role model. I would also like to thanks my officemate David Brandt for being a living encyclopedia to whom I can always turn. In addition, I want to thanks my former officemates Richard Beck for being the voice of sanity and the most methodical person I know and Esben H. Østergaard for his brilliant madness. I would also like to thanks Jason Campbell for giving me a great and inspiring stay at Intel Research in Pittsburg. In addition, I want to thanks all the other people I worked with in Odense. Thanks to Mirko Bordignon, Nicolai Dvinge, Lars-Peter Ellekilde, Franco Garcia, Andreas Lyder, Ulrik Schultz, Danish Shaikh and many others. Thanks for previous and future collaborations – I consider all of you my friends. Finally, thanks to my dear wife, Helene H. Christensen for all her support – I love you more than research itself.
# Table of Contents

## I The Study of Self-Reconfigurable Robots

1 Introduction
   1.1 What is a Self-Reconfigurable Robot? 3
   1.2 Why Study Self-Reconfigurable Robots? 3
   1.3 A Brief History of Self-Reconfigurable Robots 4
   1.4 Elements of Autonomous Self-Reconfigurable Robots 5
   1.5 Taxonomy of Self-Reconfigurable Robots 6
   1.6 Thesis Structure 8

2 Survey of Related Systems 11
   2.1 Systems and Control for Self-Reconfiguration 11
      2.1.1 Chain Type 11
      2.1.2 2D Lattice Type 14
      2.1.3 3D Lattice Type 16
      2.1.4 3D Hybrid Type 19
      2.1.5 Abstract Proposed Types 20
   2.2 Alternative Ways to Self-Assemble 21
      2.2.1 Swarm Robots 21
      2.2.2 Stochastic Self-Assembly 21
      2.2.3 Zero Gravity Modules 22
      2.2.4 Biological Inspired Morphogenesis 22
      2.2.5 Physical Self-Reproduction 22
   2.3 Summary 23

## II Elements in the Design of Self-Reconfigurable Robots 25

3 Design of Reconfigurable Modular Robots 27
# TABLE OF CONTENTS

3.1 Design Scope .............................................. 27
3.2 Design Goals and Issues ................................. 28
3.3 Design Principles ........................................ 29
   3.3.1 Principles of System Design ....................... 30
   3.3.2 Principles of Module Design ....................... 31
   3.3.3 Principles of Robot Design ....................... 32
   3.3.4 Principles of Module Behavior Design .......... 33
3.4 Summary .................................................. 34

4 Modular Mechatronics ........................................ 35
   4.1 The ATRON Self-Reconfigurable Robot ............... 35
      4.1.1 Design of the ATRON Module ................. 35
      4.1.2 Application Versatility of ATRON ............. 37
      4.1.3 Related Work on ATRON Control ............... 39
      4.1.4 Applied Design Principles .................... 39
   4.2 The Odin Deformable Modular Robot ................. 40
      4.2.1 Design of the Odin Robot .................... 40
      4.2.2 Applied Design Principles .................... 41
   4.3 The Catom Ensemble for Programmable Matter ....... 41
      4.3.1 Design of the Catom System ................. 42
      4.3.2 Related Work on the Catom System ............ 44
      4.3.3 Applied Design Principles .................... 44
   4.4 Summary ................................................ 45

5 Reliable Information Exchange .............................. 47
   5.1 Introduction ........................................ 47
      5.1.1 Design Goals ................................ 48
      5.1.2 Related Work ................................ 49
   5.2 The ATRON Communication System .................... 50
      5.2.1 Infrared-based Communication Hardware .... 50
      5.2.2 Basic Communication Protocol ............... 51
   5.3 Crosstalk Elimination and Neighbor Detection .... 52
      5.3.1 Extended Communication Protocol .......... 52
      5.3.2 Experiments ................................ 57
   5.4 Discussion, Conclusion and Future Work .......... 60
      5.4.1 Applied Design Principles .................... 60
      5.4.2 Conclusion ................................ 60
# TABLE OF CONTENTS

5.4.3 Future Work ........................................... 61

III The Scalable Nature of Self-Reconfigurable Robots 63

6 Physics of Myriads of Tiny Modules 65
   6.1 Effects of Scaling Down Module Size ....................... 65
      6.1.1 Basic Scale Effects ................................ 65
      6.1.2 Miniature Modules .................................. 66
      6.1.3 Scale Effects on Miniature Module .................... 66
   6.2 Effects of Scaling Up Module Numbers ..................... 67
   6.3 Summary ................................................. 68

7 Control and Module Size Interdependence 71
   7.1 Introduction ............................................ 72
      7.1.1 Modular Embodiment ................................ 72
      7.1.2 Related Work ....................................... 72
   7.2 Scale Effects on Locomotion of Tiny Robots ............... 73
      7.2.1 Miniature Catom Chains .............................. 73
      7.2.2 Locomotion of Miniature Catom Chains ............... 74
      7.2.3 Experimental Setup ................................ 76
      7.2.4 Results ............................................ 78
   7.3 Discussion, Conclusion and Future Work ................... 83
      7.3.1 Applied Design Principles ........................... 83
      7.3.2 Conclusion and Future Work .......................... 84

8 Organizing a Myriad of Modules 85
   8.1 Introduction ........................................... 86
      8.1.1 Module Organization ................................ 86
      8.1.2 Related Work ....................................... 87
   8.2 Anatomy-Based Organization ............................... 88
      8.2.1 Biological Inspiration ............................... 88
      8.2.2 Engineering Perspective ............................. 89
   8.3 Exploring the Anatomy Concept with ATRON ................. 89
      8.3.1 Anatomical Parts ................................... 90
      8.3.2 Discussion .......................................... 97
   8.4 Anatomy-Based Organization of Catoms ..................... 97
      8.4.1 Anatomical Control Primitives ........................ 97
## TABLE OF CONTENTS

8.4.2 Anatomical Parts .............................................. 101
8.4.3 Experiments with Anatomy-Based Catom Robots ........ 103
8.5 Odin and Anatomy-Based Organization ......................... 109
8.6 Discussion, Conclusion and Future Work ...................... 110
  8.6.1 Applied Design Principles ............................... 110
  8.6.2 Conclusion and Future Work ............................ 110

IV The Self-Reconfiguration Process .............................. 113

9 Unconstraining Motion Constraints .............................. 115
  9.1 Introduction .................................................. 115
  9.1.1 Types of Motion Constraints ............................ 115
  9.1.2 Related Work ............................................. 116
  9.2 Emergent Meta-Modules Control Strategy ................... 117
    9.2.1 What is a Meta-Module? ............................... 118
    9.2.2 Meta-Module Life-Cycle ................................ 118
    9.2.3 Attraction-points as Task Specification ............. 119
    9.2.4 Reachable Space of Meta-Modules ..................... 119
  9.3 Selecting a Meta-Module for ATRON .......................... 120
    9.3.1 Small and Non Lattice-based Meta-Modules .......... 120
    9.3.2 Different Meta-Modules Types ......................... 120
    9.3.3 Characteristics of Meta-Module Types ............... 124
    9.3.4 Results and Conclusion ................................ 127
  9.4 The Selected ATRON Meta-Module Type ....................... 129
    9.4.1 Morphology ............................................. 129
    9.4.2 Meta-Actions ........................................... 130
    9.4.3 Control Strategy ....................................... 130
  9.5 Discussion, Conclusion and Future Work ..................... 133
    9.5.1 Applied Design Principles ............................ 133
    9.5.2 Conclusion ............................................. 133
    9.5.3 Future Work ............................................ 133

10 Automatic Control Generation ................................. 135
  10.1 Introduction ................................................ 135
    10.1.1 Design Scope .......................................... 135
    10.1.2 Related Work ......................................... 136
TABLE OF CONTENTS

10.2 Evolving Control for ATRON Meta-Modules 136
   10.2.1 Artificial Neural Network Controller 137
   10.2.2 Evolution of Artificial Neural Network Controller 139
   10.2.3 Experiments 141
10.3 Discussion, Conclusion and Future Work 144
   10.3.1 Applied Design Principles 144
   10.3.2 Conclusion 145
   10.3.3 Future Work 145

11 Robustness to Faults and Errors 147
   11.1 Introduction 147
      11.1.1 Biological Motivation 147
      11.1.2 Types of Errors 148
      11.1.3 Related Work 148
   11.2 Fault-Tolerance and Emergent Self-Repair 148
      11.2.1 Experimental Setup 148
      11.2.2 Experiments 149
   11.3 Discussion, Conclusion and Future Work 156
      11.3.1 Discussion 156
      11.3.2 Applied Design Principles 157
      11.3.3 Conclusions and Future Work 157

V Adaptive Collective Behavior 159

12 Learning with any Morphology 161
   12.1 Introduction 162
      12.1.1 Scenario 162
      12.1.2 Design Challenges 162
      12.1.3 Related Work 163
   12.2 A Strategy for Learning Actuation Patterns 164
   12.3 Learning with Simulated ATRON Robots 166
      12.3.1 Experimental Setup 166
      12.3.2 Experimental Results and Discussion 168
   12.4 Learning with Physical ATRON Robots 175
      12.4.1 Experimental Setup 175
      12.4.2 Experimental Results and Discussion 178
# TABLE OF CONTENTS

12.5 Discussion, Conclusion and Future Work ........................................ 182
  12.5.1 Applied Design Principles ................................................. 182
  12.5.2 Conclusion ......................................................................... 182
  12.5.3 Future Work ..................................................................... 183

13 Learning with any Module .................................................................. 185
  13.1 Introduction ............................................................................ 185
    13.1.1 Design Requirements ..................................................... 185
    13.1.2 Related Work ................................................................ 186
  13.2 Extended Learning Strategy ...................................................... 187
    13.2.1 Learning Pose and Actuation ........................................... 187
    13.2.2 Learning Gait Table ....................................................... 187
  13.3 Extended Learning with ATRON ................................................. 188
    13.3.1 Experimental Setup ......................................................... 188
    13.3.2 Results ........................................................................ 189
  13.4 Extended Learning with Joint Limited ATRON ........................... 191
    13.4.1 Experimental Setup ......................................................... 192
    13.4.2 Results ........................................................................ 192
  13.5 Extended Learning with M-TRAN .............................................. 194
    13.5.1 Experimental Setup ......................................................... 194
    13.5.2 Results ........................................................................ 195
  13.6 Discussion, Conclusion and Future Work .................................... 197
    13.6.1 Applied Design Principles .............................................. 197
    13.6.2 Conclusion ................................................................... 197
    13.6.3 Future Work ................................................................. 198

VI Concluding Remarks .......................................................................... 199

14 Conclusion ....................................................................................... 201
  14.1 Summary ................................................................................ 201
  14.2 Contributions ......................................................................... 203
  14.3 Publications ........................................................................... 204
  14.4 Future Research ..................................................................... 204
List of Figures

2.1 Polybot, Polypod and CKbot ............................. 12
2.2 CONRO and YaMoR .................................. 13
2.3 Fracta and Micro-unit ................................. 14
2.4 Crystalline ............................................ 15
2.5 Hexagonal and square metamorphic system .......... 16
2.6 Molecule and L-Cube ................................ 17
2.7 3D-Unit and Telecube ................................. 17
2.8 External actuation and Slimebot ...................... 18
2.9 M-TRAN and SuperBot ................................ 19
2.10 Digital Clay and Sliding Cube Model ................ 21
2.11 SwarmBot and Hydron ............................... 22

4.1 ATRON mechanics and lattice ......................... 36
4.2 Three ATRON robots able to move .................... 37
4.3 ATRON robot arm and conveyer surface .............. 38
4.4 Odin robot and Odin/ATRON hybrid ................... 40
4.5 Two scales of Catom modules .......................... 42
4.6 Electrostatic Catom model ............................ 43

5.1 Crosstalk between 4 ATRON modules .................. 51
5.2 Intuition behind crosstalk elimination .............. 57
5.3 Time to detect a neighbor module .................... 59

6.1 Scale effects on module strength .................... 66
6.2 Scale effects on module velocity .................... 67
6.3 Scale effects on self-reconfiguraiton ................. 68

7.1 GA optimization of locomotion of miniature Catoms .... 77
7.2 Average velocity versus chain length and module strength/radius 79
7.3 Degree to which gaits are periodic versus chain length and module strength/radius .............................. 80
7.4 Gait-type versus chain length and module strength/radius ............................................................ 81
7.5 Typical gaits at different lengths and scales ......................................................................................... 82
7.6 Overview of gait types and fastest gait found ....................................................................................... 83

8.1 How can functionality scale up with number of modules? ................................................................. 86
8.2 Concept of anatomy-based organization .............................................................................................. 88
8.3 Notation of ATRON configurations ..................................................................................................... 90
8.4 ATRON bone .................................................................................................................................... 91
8.5 ATRON hinge joint ............................................................................................................................. 92
8.6 ATRON socket-ball joint .................................................................................................................... 93
8.7 ATRON muscle in a helix shape .......................................................................................................... 94
8.8 ATRON muscle contraction strength .................................................................................................. 94
8.9 ATRON parallel contraction ................................................................................................................. 95
8.10 ATRON-skin patterns ......................................................................................................................... 96
8.11 Anatomy-based decomposition of organization and control ............................................................... 98
8.12 Primitives for organization and control ............................................................................................ 99
8.13 Locomotion robots assembled from cilia anatomical parts ................................................................. 104
8.14 Distributed manipulation surface from bones and cilia .................................................................... 105
8.15 Experiments with distributed manipulation surface ............................................................................ 105
8.16 Muscle actuated arm ......................................................................................................................... 106
8.17 Elbow angle as a function of time ...................................................................................................... 107
8.18 Grasper robot with whisker feedback for grasp reflex ....................................................................... 108

9.1 Lattice-based versus emergent meta-modules ...................................................................................... 121
9.2 Morphology of different meta-module types ....................................................................................... 122
9.3 Reachable space of six different ATRON meta-modules .................................................................... 123
9.4 Orientation constraint on meta-module type 6 .................................................................................... 124
9.5 The selected ATRON meta-module .................................................................................................... 129
9.6 Flow diagram of control strategy for ATRON meta-module ................................................................. 131

10.1 Fitness graph when evolving a controller for a meta-module ............................................................... 140
10.2 500 ATRON modules support an unstable roof ............................................................................... 141
10.3 1000 ATRON modules self-reconfigures across a gab .................................................................... 142
10.4 Effectiveness of as a function of number of modules ....................................................................... 143
10.5 Time to self-reconfigure as a function of number of modules .......................................................... 144
11.1 Experiments on basic meta-module behavior ................. 150
11.2 Experiments on tolerance of collisions with obstacle ....... 151
11.3 Experiments on tolerance of collisions with other modules .. 153
11.4 Experiments on tolerance of module failure ................. 154
11.5 Performance as a function of module failure rate ............ 155
11.6 3426 ATRON modules self-repairing a broken bone ......... 156

12.1 Seven learning ATRON robots .......................... 167
12.2 Examples of a quadrupedal crawler learning to move ....... 169
12.3 Convergence of a quadrupedal learning to move .......... 170
12.4 Velocity at the end of learning in simulation ............... 171
12.5 Learning after self-reconfiguration and module fault ....... 172
12.6 Millipede robot used for scalability experiments .......... 173
12.7 Convergence time and divergence versus number of modules 174
12.8 Example of learning divergence for 100 module robot ....... 174
12.9 The intuition behind divergence in large scale robots ....... 175
12.10 Experimental setup for online learning of ATRON robots ... 176
12.11 Locomotion experiments with physical ATRON robots ....... 177
12.12 Pictures from learning experiment with two-wheeler ...... 179
12.13 Average velocity of a two-wheeler versus time while learning 180
12.14 Pictures from learning experiment with quadrupedal ...... 181
12.15 Average velocity of a quadrupedal versus time while learning 181

13.1 Illustration of pose and actuation learning strategy ....... 187
13.2 Illustration of gait-table based learning strategy .......... 188
13.3 Average velocity at the end of learning with three learning 189
    strategies on seven ATRON robots ....................... 189
13.4 Convergence graphs of learning with ATRON ............... 190
13.5 Average convergence time of three learning strategies on seven 191
    different ATRON robots ............................ 191
13.6 Average velocity at the end of learning with joint limited 193
    ATRON ........................................ 193
13.7 Convergence of learning with joint limited ATRON .......... 193
13.8 Typical gaits found for joint limited ATRON ............... 194
13.9 Average velocity at the end of learning with M-TRAN robots 195
13.10 Convergence graphs of learning with M-TRAN robots ...... 196
13.11 Typical gaits found for M-TRAN robots .................. 197
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Characteristics of Catoms at different scales.</td>
<td>44</td>
</tr>
<tr>
<td>5.1</td>
<td>Crosstalk characteristics of extended communication protocol</td>
<td>60</td>
</tr>
<tr>
<td>7.1</td>
<td>The six CPG parameters defining a chain type gait.</td>
<td>76</td>
</tr>
<tr>
<td>7.2</td>
<td>Correlations between physics/Catom parameters and max velocity/rise time</td>
<td>81</td>
</tr>
<tr>
<td>8.1</td>
<td>Grasping performance</td>
<td>109</td>
</tr>
<tr>
<td>9.1</td>
<td>Basic properties of different meta-module types.</td>
<td>121</td>
</tr>
<tr>
<td>9.2</td>
<td>Efficiency of meta-module types.</td>
<td>127</td>
</tr>
<tr>
<td>9.3</td>
<td>Effectiveness of meta-module types.</td>
<td>127</td>
</tr>
<tr>
<td>9.4</td>
<td>System moveability of meta-module types.</td>
<td>128</td>
</tr>
<tr>
<td>9.5</td>
<td>Meta-module moveability of meta-module types.</td>
<td>128</td>
</tr>
<tr>
<td>9.6</td>
<td>Side effect balance of meta-module types.</td>
<td>128</td>
</tr>
<tr>
<td>11.1</td>
<td>Single meta-module following online-planned sequences of meta-actions.</td>
<td>150</td>
</tr>
<tr>
<td>11.2</td>
<td>Meta-module moving from a starting to a goal position around</td>
<td></td>
</tr>
<tr>
<td></td>
<td>an unknown obstacle.</td>
<td></td>
</tr>
<tr>
<td>11.3</td>
<td>Two and three meta-modules coexisting - no coordination.</td>
<td>152</td>
</tr>
<tr>
<td>12.1</td>
<td>Controller transitions for learning experiment</td>
<td>171</td>
</tr>
<tr>
<td>12.2</td>
<td>Velocity when transferring controllers to hardware</td>
<td>178</td>
</tr>
<tr>
<td>12.3</td>
<td>Results of physical learning experiments with two-wheeler</td>
<td>179</td>
</tr>
<tr>
<td>12.4</td>
<td>Results of physical learning experiments with quadrupedal</td>
<td>180</td>
</tr>
</tbody>
</table>
List of Algorithms

1. Neighbor detection - Receive a *ping* byte on channel $i$ . . . . . . 54
2. Neighbor detection - Update state, $C_i$, of channel $i$ . . . . . . . 55
3. Neighbor detection - Is neighbor module on channel $i$? . . . . . . 55
4. Cross-Talk Elimination - Send package, $P$, through channel $i$ . 56
5. Cross-Talk Elimination - Receive package, $P$, through channel $i$ 58
6. Learning Module Controller . . . . . . . . . . . . . . . . . . . . . . . . . 165
Part I

The Study of
Self-Reconfigurable Robots
Chapter 1

Introduction

1.1 What is a Self-Reconfigurable Robot?

The concept of self-reconfigurable robots was introduced by Fukuda and Nakagawa in the late 1980s\[55\], they define that a self-reconfigurable robot: Consists of several cells (or modules), each cell have some measure of intelligence and the cells can automatically be combined and detached. Today most self-reconfigurable robot systems, consist of a number of interconnected mechanical robot modules. Modules have their own computational power and are able to communicate with other modules and sense the environment. The modules have connectors and actuated degrees of freedom that allow them to connect to, disconnect from and move relative to other modules in order to change the configuration of modules without human intervention. The design of the modules provides the self-reconfigurable robot with abilities such as shape-change to optimal configuration in a given situation and failure tolerance from redundancy and ability to self-repair.

1.2 Why Study Self-Reconfigurable Robots?

The reasons for studying self-reconfigurable robots are at least two-fold. On one hand, the exploration of such systems may provide insight into the nature and principles of biological organisms: To understand biology we must understand its basic principles such as modularity, redundancy, emergence, self-organization and self-replication. On the other hand, a reason for studying self-reconfigurable robots are their possible applications. Potential short-term areas of applications include exploration of unknown and hazardous environments such as space or earthquake areas, entertainment such as art or toys and production such as flexible robot arms. On the long term, applications also include three dimensional physical screens and remote physical presence. The two reasons for studying self-reconfigurable robots are by no means exclusive. A strong focus on the basic research
of biological principles will uncover new possibilities and open new doors for applications. On the other hand, a focus on applications will force real life constraints onto the otherwise abstract theory and much experience can be transferred to theory by working with real world problems. The study conducted in this thesis is a compromise between the two extremes. The studies conducted are not far from possible applications, while at the same time contains a more fundamental scientific value.

1.3 A Brief History of Self-Reconfigurable Robots

The idea and fascination of building artificial creatures has been around for as long as humans. Drawings on the walls of caves and figures carved in wood or amber is amongst the earliest examples. Historically new technology and insight has served as inspiration for the creation of artificial creatures. In the mid-19th century, the newfound understanding of biological cells and genetics stimulated the development of new types of artificial creatures. John Von Neumann founded the field of Cellular Automata, when he in 1950 described self-reproducing automata[121]. Later mechanical prototypes for self-assembly and self-replication were developed, for example Penrose’s wooden bricks from 1957[135, 134]. With biological cells as a clear source of inspiration, Fukuda and Nakagawa in 1988 described the concept of “Dynamically Reconfigurable Robotic System”[55]. They proposed that this type of robot system should consist of “several cells”, each cells should have some “intelligence” and the cells should be able to “be combined and detached by one another automatically”. Compare to traditional robots they claimed that the advantages were amongst other: optimal shape under circumstances, fault tolerance and self-repair. Fukuda et al. also reported the first implementation of a self-reconfigurable robot, the cell structured robot (CEBOT)[57, 50, 58]. Previously, similar systems had been proposed but they never reached past the conceptual level. Since this first self-reconfigurable modular robotic system, more than twenty such systems have been constructed. Some of these systems are reviewed in Chapter 2. Today the field is still gathering momentum and new system are being constructed: The ATRON, SuperBot and Catom systems are examples of newer systems. Some systems, e.g. SuperBot, are designed with a focus on space or inspection, taking self-reconfigurable towards applications. While other systems, such as the Catom, focus on micro-scale modules in great numbers, which brings the system closer to the original source of inspiration: the biological cell.
1.4 Elements of Autonomous Self-Reconfigurable Robots

The development of autonomous self-reconfigurable robots requires a large, complex, interdisciplinary, and costly project, which consist of many separate, yet interdependent, elements. From an abstract perspective, we can divide these elements into four groups.

Physical Platform The development usually start with a system concept which may describe the desired features of the system as well as some intended use cases. At this point crucial decisions are taken regarding the abstract capabilities of the modules, the type of configurations that can be formed, etc. The following, time-consuming, phase include design, implementation and debugging of module prototypes before the final set of modules are produced. This includes implementation and integration of all the necessary electronics, mechanics and embedded software.

Development Tools To support the development of autonomous self-reconfigurable robots a number of tool are often developed. Such tools include simulators, which are used to prototype both robots and especially control software. Simulators that can simulate dynamics scales poorly with the number of modules. Therefore, it is restricted to study robots with relatively few modules, for tasks such as locomotion. Purely kinematic simulations scales better and can be used to study large-scale self-reconfiguration. Importantly, simulators also have the function of revealing any critical shortcomings in the system concept before they are build. A dedicated programming language is sometimes implemented to program the robots (either physical or in simulation), again to facilitate the development of control. The final tool we will mention is a GUI that enables a developer to remote control and monitor a physical module or robot.

Robot Morphology Given a task we can try to come with a robot that can solve it. At this time we may realize any system concept shortcomings and limitation of the hardware implementation. A real advantage of reconfigurable robots is that we can quickly explore the robot morphology and often it is not too difficult to come with solutions at least as long as it consist of relatively few modules.

Behavioral Control The final element is to control the robot. A straightforward solution, which we will not consider in this thesis, is remote control performed by a human controller. Instead, we equip the robot with a control program that allows it to perform autonomous control. Several aspects can be controlled. The first is control of self-
reconfiguration, which enables the robot to shift between morphologies. This problem involves a when, a what and a how, where often only the how is considered. Other aspects are control of manipulation or locomotion which are either developed on a robot-to-robot basis based on a general control framework, such as gait-tables, or it can be automatically developed, e.g., based on artificial evolution. The final aspect we will mention is online adaptation, which enables the robot to adapt its behavior to the physical constraints of its morphology or environment.

The elements considered in this thesis mainly fall within behavioral control, however, since elements are interdependent and therefore cannot be studied in isolation we also address issues related to system concept and robot morphology. We try to capture our experience with these elements in the form of design principles. Design principles have the advantage that they in a compact and concise form can communicate hard-learned experience from one robot designer to another.

1.5 Taxonomy of Self-Reconfigurable Robots

Self-reconfigurable modular robotic systems are classified by properties of the module type(s) comprising the system, by the characteristics of interconnected modules and by the control strategy applied to the system. This section presents some of the most commonly used distinctions used for classifying self-reconfigurable robots and their control.

Homogeneous vs. Heterogeneous A system may be homogeneous in hardware and/or in control. If only one module type exist in the system the hardware is homogeneous, if more than one it is heterogeneous. Most self-reconfigurable robots today consist of only one or two module types. The reason is probably that the system becomes simpler to design on the conceptual level and the modules cheaper to mass-produce. However, there might be other arguments in favor of a heterogeneous solution, such as lower module complexity. Chapter 3 discusses this further.

Lattice vs. Chain Based One way to categorize different self-reconfigurable robotic systems is by dividing them into lattice or chain based systems. In lattice-based systems, the modules are positioned and move in a lattice structure like the atoms of a crystal. The lattice facilitates self-reconfiguration, since it helps to align connectors on neighbor modules. In chain-based systems, the modules are positioned in a chain that may branch and contain loops. The chain-based approach can with few modules be applied to real world problem such
1.5. Taxonomy of Self-Reconfigurable Robots

as locomotion. Some system can be categorized as hybrids since they can function both as chain and in lattice configurations, the M-TRAN, ATRON and SuperBot are examples of such systems.

**Connector Mechanism** The connector system is crucial for self-reconfiguration and is one of the most challenging mechanical parts of a module design. Connector systems can be categorized into unisex or male/female types as well as rigid and flexible types. They can also be categorized by their technology: mechanical, magnetic, electro static, etc. Important aspects of connectors include their strength, speed, rigidity, alignment tolerances and whether or not they need energy to connect/disconnect and stay connected.

**Degree of Freedom** To do work on the environment and to self-reconfigure modules must be actuated. The degree of freedom of each module, affects the complexity of controlling and constructing the system. The actuated degrees of freedom are usually rotational or linear and in the order of one, two or three DOF per module.

**Energy** The means of energy supply affects the autonomy of the individual modules and the system as a whole. Some modular systems are powered externally through wire, while other systems gets their energy from onboard batteries. Power-sharing between modules is an important feature not yet implemented in many systems.

**Communication** Communication between modules are needed to coordinated their actions. Direct communication may be divided into global or local. Global, broad-cast types, communications has the problem that it does not scale well with the number of modules. The local alternative is neighbor-to-neighbor communication, which has problems with delays in large systems, since the message may have to pass through many modules to reach its target. A hybrid alternative has been developed for the Odin system (see Chapter 4). In some control work stigmergy, which is emergent communication through the environment, is as important as direct communication.

**Centralized vs. Distributed Control** Control can be centralized, where a central computer or human directly controls the actions of the individual modules. Alternatively, the control can be distributed, where the modules themselves select their actions based on local conditions, and let a collective behavior emerge. The pros of distributed control is that it may facilitate scalability and fault-tolerance while centralized control is provable, closer to optimal and easier to design. Static (hard-coded) control is useless in most applications; it is, however, the way many demonstrations of physical self-reconfigurable robots are made.
Complexity. There is a tradeoff between the functionality that a module provides and the complexity that it must encapsulate. We will refer to the complexity of something, e.g. a module, extensively throughout this thesis, why we here will explain the term a bit more carefully. In computer science, one way to measure the complexity of a string is by measuring the length of the shortest programs necessary to generate it. However, this is not a practical method for measuring the complexity of a self-reconfigurable robot. A simple and more practical method for measuring the complexity of both hardware and software were proposed by McShea for measuring the complexity of animals [110]. He proposed to measure complexity as four different types. The first type measures non-hierarchical complexity of an object as the number of different physical parts and the second type measures the non-hierarchical complexity of a process as the number of different interactions between the parts. These two metrics measure the complexity at a given spatial or temporal scale and can easily be applied to modules by counting the number of different mechanical parts and their interactions. The next two metrics are more applicable to measuring the complexity of a self-reconfigurable robot with several hierarchical layers since they measure hierarchical complexity. For an object this is measured as the number of nested levels of parts within wholes and for a process as the number of levels in a causal specification hierarchy.

Emergence. Emergence is related to distributed control and complexity. A behavior of some complexity (not too chaotic or simple) that arises from a large number of simple processes interacting in parallel is said to be emergent. E.g. the behavior of biological organisms is emergent from the interactions of its cells. The behavior of a modular robot can be emergent, if the modules are controlled in a distributed fashion and if their combined behavior give rise to a coherent robot behavior. The potential advantages of utilizing emergence as a control design principle includes resilience to disturbances, e.g. module failures, and scalability in the number of modules.

1.6 Thesis Structure

This thesis is divided into 6 parts and 14 chapters.

Part I introduces self-reconfigurable robots. Chapter 1 gives a brief overview of the central concepts and Chapter 2 survey related systems and control.

Part II consider elements in the design of self-reconfigurable robots. Chapter 3 defines the design scope, goals and principles, which are used throughout the thesis. Then, Chapter 4 describes the design of three hardware platforms, ATRON, Odin, and Catom which forms the conceptual and
experimental platforms of the thesis. Finally, Chapter 5 considers the design of communication systems and describes the development of a distributed communication protocol for the ATRON system.

Part III turns to the fundamental aspects of scalability. Chapter 6 study theoretical effects of scaling up the number of modules while scaling down the size of the individual modules and Chapter 7 experimentally explore the inherent interdependence between morphology, module size and behavior in the context of Catom locomotion. Last, Chapter 8 propose and study a method for module organization, which is scalable and encapsulate complexity to support robot design abstraction.

Part IV study issues in the self-reconfiguration process. First, Chapter 9 describes the different types of motion constraints and propose a control strategy based on emergent meta-modules for the ATRON system. Then, Chapter 10 use artificial evolution to automatically develop a distributed controller for the ATRON meta-modules. Finally, in Chapter 11 we study fault tolerance and emergent self-repair using the ATRON modules.

Part V study adaptive collective behavior. Chapter 12 describes a simple distributed learning strategy, which enables ATRON robots to learn to move independent on module configuration and module faults. Then, Chapter 13 extends this learning strategy, so that it also works for other modules such as M-TRAN.

Part VI gives conclusive remarks. Chapter 14 contains a summary of the thesis, summarizes contributions, peer-reviewed publications, and point out future research directions.
Chapter 2

Survey of Related Systems

This chapter first briefly describes a number of different self-reconfigurable robotic systems and then some of the control strategies developed to control them. We group the systems into Chain, 2D lattice, 3D lattice and Hybrid systems. Finally, we describe alternative ways to self-assemble, other than self-reconfiguration.

2.1 Systems and Control for Self-Reconfiguration

2.1.1 Chain Type

Polypod The first chain based self-reconfigurable robot, Polypod, was developed by Yim as part of his Ph.D. work in the early 90's[198, 199]. It consists of two module types: Segments and nodes. Segments have two degree of freedom (DOF) and are equipped with actuation, sensing, computation and communication. Node modules have batteries and six connectors to allow branches in the robots. Unisex connectors, that passively connect and disconnect using a shape memory alloy (SMA), allow the Polypod to self-reconfigure. The main ability of the Polypod is locomotion, such as snakes, walkers and rolling tracks[198, 200] and autonomous shifting between different locomotion styles. Figure 2.1(a) shows a Polypod segment. Different locomotion styles is controlled using gait control tables, which defines the action of each module in a given global time-step.

Polybot The Polybot is a second generation of Yim’s Polypod[203]. Polybot also consist of nodes and segments but the mechanical complexity of segments has been decreased. Polybot segments only have a single rotational degree of freedom, see Figure 2.1(b). Since the late 90’s three generations of the Polybot system have been developed[43]. The later versions (not the first) are able to self-reconfigure and most of the versions use external power. Since there is no mechanical alignment of the Polybot modules, as in lattice
based systems, IR emitting and sensing are used to guide the alignment of the unisex connectors[142, 212]. As for the Polypod, the Polybot is able to perform many different types of locomotion[205, 202] and also surface-based distributed manipulation has been shown[207]. To control locomotion phase automata patterns have been developed, which are state machine, defining the actions of a module in a given state[222]. The shift from one state to another is event (time or sensor) driven. Proposed applications for the Polybot include manipulation and locomotion in space[208], urban search and rescue[201] and land warfare[46].

**CKbot**  CKbot is the latest addition to Yim’s Polypod family. The kinematics is similar to Polybot, although the nodes have disappeared and several segment types have been introduced. The three types of segments are ‘L’ and ‘U’ modules with variations of the Polybot kinematics and ‘Leg’ modules that can be used as wheels. Compared to other modular robots, the speed of the ‘L’ and ‘U’ segments are at impressive 0.1 sec for a 60-degree rotation. Each segment has four passive connector surfaces, which can be attached to another connector using either magnets or screws dependent on the version. In addition, the system includes a number of end-effectors, such as digger, syringe and probe. A global CAN bus provides communication between modules and local IR channels provide topology detection. Batteries and ZigBee communication are added externally to the modules, power is shared through a high voltage (22V) and a low voltage (6V) power bus. As Polybot, the CKbot is able to perform various effective locomotion styles[148]. Further, CKbot has been used as the platform to perform amphibious locomotion and other tasks such as digging and liquid sampling in Lake Tyrrell, Australia[209] (see Figure 2.1(c)). Using the CKbot automatic recognition of configuration has been studied [132] and self-assembly after disassembly/explosion has been demonstrated[210].

**CONRO**  The CONRO system consist of homogenous modules that have two rotational DOF (yaw and pitch) [27, 25, 26], see Figure 2.2(a). A
module is self-contained including batteries and has one female and three male connectors, which allows snake, tree or single loop configurations. In its latest version the female connector, is actively able to release the male connector[80, 81]. Automatic docking has been demonstrated, as for the Polybot, connectors are aligned using IR feedback for guidance[155, 143]. The CONRO system is well suited for different styles of locomotion. Role-based control was developed to control locomotion of CONRO robots[170, 172, 169]. Each module, dependent on its position in the configuration tree, automatically selects a role. This allows a CONRO robot to shift from one locomotion style to another when it is manually reconfigured[171]. Hormone-based control is a generic strategy, proposed by Shen et al.[152, 153], which allow information to be distributed between the modules. Digital hormones represent information that propagates through the configuration of modules. Using digital hormones modules can detect topological changes, coordinate and synchronize actions for locomotion or self-reconfiguration[154, 146].

YaMoR Yet another Modular Robot (YaMoR) is a chain type system developed at EPFL, Switzerland[113]. Its kinematic is similar to the Polybot modules, with a single rotational degree of freedom. Each module has on-board Bluetooth communication and a FPGA is used as the controller[114]. Robots are manually assembled from modules. Using the YaMoR system co-evolution of behavior and morphology has been explored in simulation[105] and online learning in both simulation as well as on the physical hardware[106, 162]. Figure 2.2(b) shows a YaMoR module.

Self-Reconfiguration of Chain-Based Systems A natural representation of a chain-based robots configuration is as a graph[28]. A distributed algorithm for discovery of topology changes were given by Salemi et al.[147]. Casal and Yim divides reconfiguration into three different classes[24]: i) Mobile (each module is mobile in the environment), ii) Substrate (lattice based systems) and iii) Closed chain (chain based systems). For closed
2.1.2 2D Lattice Type

**CEBOT** The first self-reconfigurable robot was the cell structured robot (CEBOT), proposed and partly implemented from the late 80’s by Fukuda et al. [57, 50, 58]. The proposed system is heterogeneous with three levels of module/cell types: 1) actuation and mobile cells, 2) branching, length adjustable, orientation changing and power cells, 3) Work cells, such as end-effectors. Different prototypes of different modules types have been developed. In [57] a mobile (wheeled) cell, with some sensing capabilities were shown to dock to a passive cell using infrared light for guidance. Communication between cells is described in [54], and in [51] cells are assembled using robot arms. A number of studies have been conducted on distributed control of the CEBOT, including how to organize distributed knowledge[56], how to organize groups behavior[52] and how to make distributed decisions[53, 79].

**Fracta** In 1994, Murata et al. presented the Fracta unit[117], see Figure 2.3(a) and 2.3(b). Its simple design allowed it to rotate around any of its six neighbor units using tree electro-magnets. The hardware implementation includes computation and optical neighbor-to-neighbor communication, power is external. 20 modules were made. A simple stochastic self-reconfiguration/self-assembly strategy were developed to control the system[117]. Based on local conditions a unit can estimate a distance to its goal, if the distance is not zero it will with some probability perform a random move. In [221] the self-reconfiguration strategy is improved to avoid deadlocks and speedup by moving in the direction more probable to

Figure 2.3: (a) Workings of a Fracta unit. (b) Group of Fracta units. (c) Two micro-units.
2.1. Systems and Control for Self-Reconfiguration

Figure 2.4: (a) A single Crystalline Atom. (b) A group of Crystalline Atoms.

be correct instead of just random. If a unit is removed from the system a process of self-repair begins. The system self-reconfigures so that a spare unit takes over the holes in the structure. This method works well for systems with few units (< 30). For large scale systems a hierarchical method for self-reconfiguration and self-repair has been developed[179].

Micro-unit The micro-unit[214, 215, 219] is an attempt to miniaturize the units, required for a self-reconfigurable system. The units measures $2 \times 2 \times 2$ cm and weighs 15 g. Each module has four neighbors, which it can autonomously rotate around using its SMA actuators. The connector system is male/female with female connectors able to release the male connector also using a SMA. Two micro-units in the process of reconfiguring are shown in Figure 2.3(c).

Crystalline The shape of a Crystalline Atom is square, see Figure 2.4(a). The faces of the Crystalline square can actively be contracted and extended by a factor of two. In the newest implementation the Crystalline Atom has 2-DOF, since it can actuate its extension/contraction in the x and y axes separately. Two of the four connecting surfaces are active males; the other two are passive female. Relative motion of an Atom is accomplished by extension/contraction, which makes it slide past other Atoms. This allows the Atom to move through the structure, which is unlike most other systems that only allow movement on the surface. For self-reconfiguring of a Crystalline structure a distributed planning strategy, called PACMAN, has been developed[20]. Each Atom individually finds a path that it then follows towards a goal position. A group of 12 Crystalline atoms can be seen in Figure 2.4(b).

Metamorphic Metamorphic modules come in hexagonal or square shapes[29, 130], see Figure 2.5(a) and 2.5(b). Upper and lower bounds on module moves for self-reconfiguration are found in [30] and some distance metrics are given in [131]. The modules move from one lattice position to another, sliding along or climbing over neighbor modules. Distributed control from a chain
to a shape, filling holes, forming bridges or enveloping obstacles has been extensively studied [189, 193, 191, 192, 190, 101]. Evolutionary techniques have been used to generate control to move small groups of modules through a maze-like world[3] and sorting membranes[4].

2.1.3 3D Lattice Type

**Molecule** The Molecule system was developed from the late 90’s. A Molecule module has 4-DOF and is able to self-reconfigure in 3D[90, 89]. Molecules are bipartite and consist of two atoms connected by a rigid bond. Each atom can rotate relative to the bond and relative to one of its five connectors. An molecule is shown in Figure 2.6(a). The third and latest generation of Molecule features a male/female connector system, such that every other module is equipped only with male or female connectors. Four Molecules have been constructed. For controlling 3D structures of Molecules a hieratical planning approach is taken[86, 84]. Planning is divided into trajectory, configuration and task-level planning. Trajectory planning finds a path from one position to a goal position. Configuration planning finds a plan to move a group of Molecules from one configuration to another, using trajectory planning. Task-level planning selects which configuration fits the required task of the Molecules. Two approaches to locomotion has been explored[85, 88]. One approach is statically stable locomotion, which moves the structure of modules by self-reconfiguration while keeping the system stable. Another approach is dynamically stable locomotion, which moves the robot centre of mass to produce dynamic locomotion for example in the form of a rolling wheel.

**I-Cube** The I-Cube module is a two module system consisting of links and cubes[182, 184]. The links have three rotational degree of freedom and are capable of moving and (dis)connecting from the passive cubes, see Figure 2.6(b). In [139] a hieratical planner strategy is suggested for shape-changing I-Cube structures, on the top level planning is done for metacubes consisting of 8 cubes and 16 links.
2.1. Systems and Control for Self-Reconfiguration

Figure 2.6: (a) Molecule. (b) Two I-Cube links and a cube.

Figure 2.7: (a) Four 3D-Units. (b) Second generation Telecube.

**3D-Unit** A 3-D unit module has six degree rotational of freedom[97, 118]. Each unit has a cube at its centre and arms that can connect and rotate on each face of the cube. With these characteristics, a unit is unable to move itself, but it can move neighbor modules. The 6-DOF makes the modules mechanically quite complex, and with a total module weight of 7 kg a 90-degree rotation take about 1 minute. In [220] a method for distributed shape-change of 3-D unit structures was presented. The method applies the metaphor of simulated annealing. Each unit first calculates its reachable positions, one-step away. Then it with greater probability selects a reachable position closer to the target configurations. The proposed method works for structures of up to 20 units.

**Telecube** Telecube is a 3D generalization of the 2D Crystalline system[174]. Its six degree of freedom allows it to expand each of its six faces independently, such that the module can expand by a factor of two. Connection is achieved using switching permanent magnets on the connector faces, actuated by SMA. The modules have IR communication and computation onboard but rely on external power. In its contracted state a module is 5x5x5 cm. Figure 2.7(b) show a second-generation module. A membrane of Telecubes can manipulate objects passing through it. Control for membranes
that exhibited sorting behavior of different types of objects, were evolved in [95, 94]. In [188, 187] a complete (can reach all possible configurations) control strategy was presented. The strategy used meta-modules consisting of 2x2x2 Telecubes. Such meta-modules generalize to abstract cubic type of modules with has roll over and slide along type of actions. The running time of the algorithm was proven to be $O(N^2)$ where $N$ is the number of meta-modules. A distributed control strategy which were presented in [93], uses the concepts of seed modules which send out scents to guide the motion of other modules. The control strategy allows the groups of Telecube modules to move around obstacles and manipulate objects [92].

**External Actuation Modules** White’s self-reconfigurable modules does not contain any actuation for moving in the lattice structure [197]. Instead, it relies on external actuation in the form of synchronous motion. A module is square with a single rotational degree of freedom connector at each corner. Connectors passively bond using permanent magnets; the bond can be broken using shape memory alloy actuators. Modules can rotate about the corners of neighbor modules by disconnecting and connecting at the right time and then let their inertia move them from one lattice position to another. The goal is to simplify the modules, to allow them eventually to be scaled down in size. Figure 2.8(a) and 2.8(b) show an external actuation module and illustrates the concept.

**Slimebot** The Slimebot modules have six linear degree of freedom and one extra actuator to increase/reduce friction [75]. Unlike most other self-reconfigurable systems, it does not use rigid active connectors. Instead, each connector is made from passive unisex Velcro. The Slimebot utilized a simple distributed control mechanism where each module changes its oscillation frequency depending on the local sensor state. Self-reconfiguration is then an emergent process that arises from local module oscillation. This control strategy allows the Slimebot to exhibit photo taxis, while adapting to obstacles, see Figure 2.8(c). Heterogeneous connectors, implemented as modules
2.1. Systems and Control for Self-Reconfiguration

Figure 2.9: (a) M-TRAN modules in the form of a walker. (b) SuperBot Design. (c) Three SuperBot modules.

Other Systems The 2D Chobie modules[74, 83] are shaped like squares that can slide on the side of other modules. In [178] the design and prototype implementation of a 3D gear type module is described. In [138] the mechanical design of the M-cube, a 3D, 6-DOF cubic module is described. The module is able to rotate each of the faces in the cube. A cubic pneumatically actuated module is presented in [73]. A 16-DOF module called “robotic atom” is proposed in [44, 67].

2.1.4 3D Hybrid Type

M-TRAN The M-TRAN modules have been developed since the late 90’s[119, 116]. A module has two parallel rotational degrees of freedom. The modules are fully equipped with different types of sensors, batteries and communication and computation capabilities. The M-TRAN modules are both able to function as a lattice based and as a chain based system, why it is classified as a hybrid system. Three generations of modules have been constructed, the latest generation uses a mechanical connection mechanisms. In terms of self-reconfiguration planning for cluster walking[217, 216] and the shift between different locomotion styles has been explored[96, 99]. Evolution has been used to produce locomotion (including self-reconfiguration) for small groups of M-TRAN modules in [218]. Similar central pattern generators for locomotion of different robots (fixed configurations) were evolved in [78]. The latter approach allowed the robots to adapt its locomotion pattern to changes in the surface properties[77]. A M-TRAN walker is shown in Figure 2.9(a).

SuperBot The SuperBot[151], see Figure 2.9(c), is a new system currently under development. It has almost the same physical characteristics
as the M-TRAN module except for an extra degree of freedom. The two half of a modules are connected with an actuated joint allowing them to rotate relative to each other, see Figure 2.9(b). It is argued that it combines the advantages of the M-TRAN, CONRO and ATRON into one system. A SuperBot module can move on its own allowing the system, at least in principle, to self-assemble from unconnected loose modules. At the cost of added mechanical complexity, the SuperBot improves the locomotion capabilities of the M-TRAN modules, along with an improved ability to self-reconfigure (reduced motion constraints).

2.1.5 Abstract Proposed Types

Cubic module The sliding Cube is an often taken abstract model for self-reconfigurable modules. Such Cubes sit in a cubic lattice and are able to slide along and around the faces on neighbor Cubes. The main reason to study this system is to develop control strategies, which supposedly have a more generic nature. Also, some physical systems such as I-Cube[183] and Molecube[109] can construct large meta-modules that have the motion characteristics of cubes. This also apply to Telecube[187] but the 2x2x2 meta-modules have slightly different characteristics. The reason the system has not yet been implemented physically is probably due to the mechanical complexity, e.g. number of DOF with probably would be two per connector face, making it a total of 12 DOF. In a series of papers[168, 167, 164, 165], Stoy develops a distributed control strategy for such modules. The strategy starts with a CAD module of the desired shape and from a random configuration of modules reconfigures it into the shape, see Figure 2.10(b). The strategy relies on scaffolds and attracting gradients. The scaffolds allow modules to move freely through the centre of the structure, using the volume to self-reconfigure and to remove local minima. Cluster walk type locomotion, using cellular automata type rules, has been explored in [16, 17, 18, 19]. The cluster can move across obstacles or through tunnels and clusters may divide or merge. However, the world it simulated to fit the cubic lattice. Similar work on self-reconfiguration control for cube type modules include [47] and [87].

Proteo Yim et al. suggested utilizing rhombic dodecahedron shaped modules called Proteo[206, 211]. Unactuated prototypes are shown in Figure 2.10(a). Each module has 12 rhombus faces and is considered a 3D analogy to the 2D hexagons. They have the property of being homogenous and space filling. The argument for Proteo modules over cubic modules is scalability. Cubic modules slide along the sides of other modules, making friction an increasing problem as the system scales down. Proteo modules roll from face to face on neighbor modules, eliminating friction. However, the Proteo module's 12 faces make the system hard to implement, since this probably will
2.2. Alternative Ways to Self-Assemble

2.2.1 Swarm Robots

In contrast to self-reconfiguration, the swarm approach to self-assemble relies more on the capabilities of the individual modules. The individual modules have greater mobility in relation to the environment, e.g. using wheels. Swarm systems take inspiration from social insects such as ants and bees, whereas self-reconfigurable systems mainly take inspiration from cells. An example of swarm type robots is SwarmBot[65, 180, 66, 115, 42], which are mobile robots that can physical self-assemble using grippers to achieve collective behavior, see Figure 2.11(a).

2.2.2 Stochastic Self-Assembly

Most artificial systems rely on internal energy (actuators) for self-assembling; in contrast to this stochastic self-assembly is achieved using energy from
the environment. The individual building blocks are able to connect and
disconnect but are otherwise passive. By stochastic shaking the modules,
self-assembly can be achieved. An example of this in 2D on a air-hockey
table[195] and in 3D, where modules are submerged into oil[196]. Miyashita
et al. presented another example of stochastic self-assembly with simple
vibrating modules floating on a water surface[112].

2.2.3 Zero Gravity Modules

Gravity is perhaps the single most limiting factor on the mechanical design
of self-reconfigurable robots. A way to eliminate this factor is by construct-
ing modules for use in space[157] or by building underwater modules. The
Hydron modules are an example of the latter[129, 177, 163, 82, 124]. The
modules are equipped with sensors, batteries and four nozzles, which expel
water to control horizontal motion. The modules control buoyancy, for ver-
tical movement using a syringe, see Figure 2.11(b). Similar approaches are
taken in the AMOUR system[186].

2.2.4 Biological Inspired Morphogenesis

An alternative approach to self-assembly is to directly mimicking biological
cells. Such work has been presented in simulation by Hotz[69, 70, 71]. The
next natural step is to build robots (or artificial cells) that mimic the physics
of cell, this is being perused in the current Pace project: “Programmable
Artificial Cell Evolution”.

2.2.5 Physical Self-Reproduction

An important aspect of self-assembly is self-replication. Self-replication,
also biological, is always dependent on the right conditions and building
blocks in the environment. An important aspect is the ratio between the
complexity of the entity being replicated and the complexity of the building
blocks. Artificial physical self-replication is still in the lower end of this
ratio, because the building blocks are still much more complex than what they replicate. Examples of physical self-replication include a LEGO car that in a carefully designed environment can self-replicate\cite{175} and groups of robot modules which can recreate their structure from modules in their environment\cite{224, 173} and self-replication in a stochastic environment\cite{64}.

### 2.3 Summary

This chapter surveyed systems that have properties relevant for this thesis. These systems all have an adaptable morphology since they consist of reconfigurable interconnected units. Further, the systems are scalable since more units can be added open-ended. The functionality of these systems varies from simple unactuated structures, to self-contained robots, to robot swarms. The systems either can function in the plane (2D) or in space (3D) and their environments can vary from being on land, on a water surface, or submerged in water.

Most of the systems can physical self-organize by their ability to perform some form of self-assembly. One class of systems use self-reconfiguration, where the units stays interconnected while changing their configuration. In this class, units were robotic modules that could form lattice, chain or hybrid type morphologies. Most of the systems were self-contained with the necessary actuation to self-reconfigure, except one system that utilized external actuation. Another class of systems, self-assembled from separated units. The units of these systems were either complex mobile robots or simple modules. The simple modules self-assembled by relying on external propulsion or in one case the modules themselves vibrated to move.

In Chapter 3 we will define the scope of systems considered in this thesis. This scope is spanned by the three systems presented in Chapter 4.
Part II

Elements in the Design of Self-Reconfigurable Robots
Chapter 3

Design of Reconfigurable Modular Robots

In this chapter, we discuss the design of reconfigurable modular robots. We are concerned with design regarding all aspects of the system. First, we will define the design scope, i.e. the type of systems that we consider. Then, we describe the design goals of such systems. Finally, we describe a number of design principles for the design of systems, robots, modules and behaviors. We follow these principles throughout this thesis and explain how we applied them at the end of most chapters.

3.1 Design Scope

There is a tradeoff between engineering and science in our long-term overall objective. On the scientific side, we are interested in working toward autonomous, self-sustaining, self-assembling, self-replicating, evolvable robots as an instance of artificial life to increase our understanding of life in general. On the engineering side, we want to maintain the realism of practical applications, by working toward a versatile platform for robots able to move and manipulate autonomously in the context of real tasks, e.g., search and rescue.

To realize our overall objective we consider the use of reconfigurable modular robots. Such a robot is constructed from physically coupled modules that can be reconfigured manually or can reconfigure themselves. The same set of modules can be combined in different ways to construct many different robots. The system is open for an increased number of modules, which means that we can construct a robot from 10, 100 or 1000 modules without limit. The modules are self-contained with computation and communication abilities. Further, a robot is actuated and has sensors to enable it to interact with the environment.

To satisfy the scientific part of our objective, we are interested in be-
haviors, which are not tied to a particular robot with a fixed number of modules. The robots and behaviors considered must show progress toward the long-term objective. Consequently, we are not interested in few-module robots, which is just a modular implementation of a conventional robot. To satisfy the engineering part of our objective we do not consider systems that are unable to interact with the environment, e.g., unactuated modules used to study stochastic self-assembly in isolation. Likewise, we do not consider modular robots that are designed for a narrow application range or low versatility, e.g., modular serial manipulators.

### 3.2 Design Goals and Issues

This section describes design goals for reconfigurable modular robots. We explain why such robot may be more able to fulfill these goals than conventional robots. For a balanced discussion, we also point out the typical issues, which make these claims not hold in practice. In the next section, we propose design principles that can help us to avoid these issues and realize the design goals.

**Autonomous and Adaptive**  
*Claim:* Self-reconfigurable robots are more autonomous since they are able to adapt to a higher degree than traditional robots, not only can they adapt at a behavioral level but also at a morphological level.

*Issues:* Realizing morphological adaptation requires reliable self-reconfiguration which few systems are able to demonstrate. Further, the algorithms for controlling self-reconfiguration of existing physical systems are complex and generally assume large-scale systems and are therefore not applicable for the typical few-module systems. Also, the criterias for when to self-reconfigure and into what is a largely unexplored area.

**Versatile and Flexible**  
*Claim:* Compared to traditional (fixed-configuration) robots the versatility and flexibility of reconfigurable robots comes from the use of simple modules. The modules can be assembled in a variety of configurations, constructing various robots, which yields high versatility. The ability to quickly assemble or change a robot yields high flexibility. If the robot is self-reconfigurable, the versatility is further increased since the robot can autonomously change between configurations.

*Issues:* In practice it is difficult to solve real application with modular robots. This is due to the design requirements that modules must be somewhat general, as well as the specific overhead of redundant mechanics and electronics, e.g., connectors and processors that make a module too clumsy and inefficient to solve practical problems.
3.3. Design Principles

**Scalable Claim:** The number of modules in a reconfigurable modular robot can be scaled up, which indirectly can improve other design goals such as versatility and reliability.

*Issues:* It is time consuming and expensive to produce and maintain a high number of modules. From the point of real applications more modules tends to decrease, not increase, the reliability because the system as a whole is more complex with more possibilities of errors. Collective actuation and sensing are required for whole-body behaviors of robots comprised of many modules. Few, if any, strategies exists for collective actuation and most existing modules are not designed for it.

**Reliable and Robust Claim:** Reconfigurable robots are redundant, which allows one module to compensate for the fault of another module. Repair is made simple since a broken module with little effort can be replaced. If the robot is also self-reconfigurable, a spare module can physically replace a broken module, and thus achieve a level of self-repair.

*Issues:* A typical module design is complex with many interacting mechanical and electronics components, this decrease the reliability of the individual modules. Exploiting redundancy is not automatic and adds to the overall complexity of the system. Further, a typical robot only consist of few (perhaps up to a dozen) modules and in such system redundancy is too low to be useful.

**Cheap Claim:** When moving beyond research prototypes, modular robots may eventually become cheap compare to traditional robots. This is due to the cost optimization effect of mass production of relatively simple modules.

*Issues:* The high redundancy of modular robots, such as extra actuators, is basically a waste of resources which counter effects any effect from mass production. On the contrary, conventional robots can always go for a minimal, non-redundant, and therefore cheaper design.

3.3 Design Principles

Design principles can guide the design of robots. Their purpose is to be make the implicit explicit, and thereby to facilitate exchange of knowledge and experience between robot designers. Design principles constraints the design space to simplify the design task. Further, design principles are debatable. Therefore, they can be strengthen or weakened through experiments, to eventually arrive at improved design principles.

In this section, we present a number of design principle on which we base the designs of robots and control strategies described in this thesis. The scope of the design principles are limited to the type of robots described in Section 3.1. Our design principles are inspired by Pfeifer’s design principles
for autonomous agents[136, 137], and follow the original ideas of embodied intelligence proposed by Brooks[15, 14], as well as Materic’s work on mobile multi-robot systems[107]. Most importantly, the design principles are not new but have been the implicit of explicit working hypotheses of numerous researchers in the field of reconfigurable modular robots.

We divide the principles into categories of system design, robot design, module design and module behavior design. We do not see the design principles as proven truths, but as a set of working hypotheses for which there exists more or less validation. The following chapters will exemplify and evaluate the use of these principles in the context of various robots and tasks.

3.3.1 Principles of System Design

Already when we design the system’s concept, we face critical choices, which may eventually decide the success of the system. Here, we point out two design principles that are supposed to steer designers clear of two common pitfalls.

— Principle 1 —

A system cannot be universal, but must be versatile, scalable and extendible.

No general-purpose system exists - the design of a system is a trade-off between generality and optimization for a specific range of tasks and environments. Therefore, the design of a system must be based on clear constraints imposed by the task/environment. Further, as discussed in the previous section a system must be versatile and scalable. We also require that the system is extendible, so that new features can be added to the system gradually. Principle 2 states how to realize extendibility.

— Principle 2 —

A system must consist of a set of morphological differentiated module types.

Another principle found in several systems is that of heterogeneity. The purpose of this principle is to balance module complexity and system versatility. If the modules are homogeneous, either the system is closed for extensions of new features, e.g. sensors such as a GPS, or we must increase the complexity of the module design to extend it. Instead of insisting on homogeneous modules, we can simplify the system design by splitting the functionalities over several morphological differentiated modules types.

An simple example of this principle is the Polypod design that consist of node (battery, six connectors) and segment (actuator, two connectors)
modules. The result of this division is that both module types are relatively simple, compared to an alternative module design which would have to include six connectors, actuator and battery in one module.

### 3.3.2 Principles of Module Design

A system consist of a number of differentiated module types. Here, we consider principles for designing a module type for such a system.

— **Principle 3** —

> A module design should exploit gravity, friction and other physical effects.

This principle advocates that a module design must exploit rather than overcome physical properties. Most lattice-based self-reconfigurable systems violate this principle since they are designed in an attempt to make module behavior independent of physics, e.g. gravity, in the process of self-reconfiguration. It does so by designing the modules with connector systems that can be assumed completely rigid and actuator systems that are so strong and precise that they can align connectors ideally. This approach can easy leads to clumsy, wasteful and complex designs. Some alternative systems are able to exploit physics to make the task simpler, e.g. the Slimebot, White’s external actuation modules or systems that utilize stochastic self-assembly. Yet, still no system has demonstrated self-reconfiguration or self-assembly in a three dimensional gravity environment, while taking advantage of physics.

— **Principle 4** —

> A module design should exploit the physical presence of other modules.

When designing a module we are also designing the ecological niche that the modules are situated in. Lattice based designs are a good examples on how to exploit this. In this case, a special niche is created in the form of a lattice, and a module is designed to exploits the constraints of the lattice in order to self-reconfigure. The lattice simplifies the design because it specify at which points a module must have connectors and the relative orientation of the modules.

— **Principle 5** —

> A module design must be specialized to a small set of capabilities.
This principle states that a module’s complexity must be kept low by specializing it to a few actuator or sensory functions. Together with Principle 2, this principle specifies that a system should consist of several module types each with their own set of basic functionalities. Note, that this principle is not valid for all scenarios, for example, robots with few modules or for applications such as physical rendering, a complex module design or a homogeneous design may be preferable. Also, note that such scenarios are beyond the scope of this thesis.

3.3.3 Principles of Robot Design

Design principles for conventional robots, e.g. Pfeifer’s principles, still apply when assembling a reconfigurable modular robot: It must be situated, self-sufficient, designed for a particular niche, etc. In the following principles, we consider only the aspects that are special to reconfigurable modular robots.

— Principle 6 —

A robot’s behavior emerges from the collective behavior of its modules.

According to Brook’s behavior based paradigm a robot’s behavior should emerge from parallel, loosely coupled control processes. For reconfigurable modular robots, this is true not only in control, but also in hardware if we perceive a module as a parallel, loosely coupled process. Two potential advantages of emergent behavior are: i) that the behavior of the robot can be robust to disturbances at module level and ii) that complex behavior at the robot level can arise from simple behavior at the module level. Further, this principle highlights the collective aspect of a reconfigurable modular robot. Both for actuation and sensing any single module may be too week and too uninformed to do any difference alone, but by collective sensing and actuating the robot may be able to act as a coherent whole. An implication of this principle is that we must utilize distributed control, as we will address further in Principle 8.

— Principle 7 —

A robot’s function cannot critically depend on any of its modules.

In the worst-case, a robot has a single-point-of-failure per module and therefore a reliability problem. We have to take into consideration that as the number of modules are increased in a robot the probability that some module will fail increases. Therefore, to design a scalable robot we must design and control it so that it can tolerate that some modules are unable
3.3. Design Principles

to fulfill their function. For example, an often-used design pattern is that of
a specific seed module, which starts an initialization sequence in the robot.
This is a violation of this principle, but it is a minor violation since the
dependence is brief and only on a single module.

3.3.4 Principles of Module Behavior Design

Not only do we consider a self-reconfigurable robot as a complete agent, we
also consider a module as a complete agent. The role of a module in a robot
is as a cell, anatomical part (e.g. muscle), or limb (e.g. leg) in a biological
organism. Which metaphor applies is dependent on the number of module
in the robot. Here, we describe principles for the design of module behavior
based on these metaphors.

— Principle 8 —

A module acts independently and maintains its own integrity.

A module is responsible for its own control, is situated within a robot
and perceives the world from this perspective. A module is a physical entity
able to perceive and manipulate parts of its world, e.g., it may autonomously
decide to connect or disconnect from a neighbor module. Further, a module
is self-sufficient in the sense that it embeds its own actuation and processing.
This ensures that a module can perform local tasks and handle local prob-
lems. At the robot level, this leads to increased robustness. By following
this principle, we also implicitly follow Principle 6 and 7.

— Principle 9 —

A module is not alone, but exists in a society of peer modules.

A module must also often rely on the willingness of other modules to
collaborate or the ecological niche that they provide. For example, power
cannot be harvested by any individual module. Therefore, individual module
must depend on the society of modules, the robot, to maintain and distribute
the power, e.g., through power-sharing. Similar, some control parts, e.g.
behavior arbitration, affect the robot as a whole and must therefore be
handled collectively by the modules. This principle should help us avoid the
pitfall that a module must be able to do everything itself. This helps to
encapsulate complexity, since some tasks can be handled at the robot level,
while individual modules can handle other tasks.

— Principle 10 —

A module’s behavior is based on local context and local
interactions.
The context is a module’s neighborhood of modules and interactions can be with other modules or the environment. Both context and interactions should be local in both a spatial and temporal sense. The goal is to simplify the control by ensuring the situatedness of a module within a robot. Further, only by excluding centralized control can we study how a global robot behavior can emerge from simple, local, independently acting modules.

— Principle 11 —

A module is not unique, but can be dynamically replaced.

Self-reconfigurable robots are naturally redundant, due to their modular nature with many identical or similar modules. It is therefore important to maintain the redundancy and flexibility of the system by ensuring that a module at any time can be replaced by an equivalent module without compromising the integrity of the robot. For this reason, for example the use of global unique module ID’s is discouraged.

— Principle 12 —

A module’s genotype is identical for same type modules.

With genotype, we mean the behavioral specification of a module, typically a software controller. Therefore, the most straightforward way to implement this principle is to use the same controller for the same type modules. It limits the number of controller that needs to be maintained and facilitates scalability. Note that the use of one controller does not imply that the organization of modules in the robot will be non-hierarchical and monotonic. In fact, the behavior of the individual modules can, and should according to Principle 10, be context depended. In addition, this principle is a practical way to realize Principle 11.

3.4 Summary

In this chapter, we considered the design of reconfigurable modular robots. The design scope was limited to modular robots able to interact with the environment and further fulfill the design goals of being: i) scalable in the number of modules, ii) versatile to solve a large range of different tasks, iii) reliable to tolerate module failures and iv) autonomous to be independent from human guidance. Three systems that span this scope, and therefore will form the experimental platforms of this thesis, will be described in Chapter 4. Finally, we described a dozen design principles we will strive to follow. The design principles touch on all aspects of the design, from system concept to module behavior design. In essence, the principles advocate for specialized and bottom-up solutions rather than general and top-down solutions. We anticipate that by utilizing these principles the development of autonomous modular robots will be facilitated.
Chapter 4

Modular Mechatronics

In this thesis, we utilize the ATRON, Odin and Catom systems as case studies and for experiments. Combined, these three systems span the scope of the systems that we consider. In this chapter we describe the systems and evaluate their design based on the design principle described in Chapter 3.

4.1 The ATRON Self-Reconfigurable Robot

The ATRON self-reconfigurable robot were developed as part of the European funded HYDRA project[124]. Partners in the HYDRA project were from University of Southern Denmark, University of Edinburgh, University of Zurich and LEGO Company. Here, we briefly describes the ATRON modules, a more extensive description of the ATRON design can be found in [125] and of the current hardware implementation in [76].

4.1.1 Design of the ATRON Module

An ATRON robot, consist of a number of interconnected modules. The ATRON module is a simple, one degree of freedom, homogeneous, lattice-based module able to self-reconfigure in 3D. An ATRON module, see Figure 4.1(a), has a spherical appearance composed of two hemispheres, which the module can actively rotate relative to each other. On each hemisphere, a module has two actuated male connectors and two passive female connectors. A module weighs 0.850kg and has a diameter of 110mm. 100 hardware prototypes of the ATRON modules exist.

Center Axes Rotation Rotation around the centre axes is normally done in 90-degree steps. This move a module, connected to the rotating module, from one lattice position to another. The rotation is achieved using a single motor, which is geared down, to enable it to rotate up to two other modules in any direction (worst case is against gravity). The center-motor can be
Figure 4.1: (a) A single ATRON module, on the top hemisphere the two male connectors are extended on the bottom hemisphere they are contracted. (b) The ATRON modules sit in a surface-centered cubic lattice structure that defines the relative orientation (indicated by color) and position of each module.

actively braked to enable it to hold other modules in the lattice position. One full, 360 degrees, rotation takes about 6 seconds, without the load from other modules.

**Connectors** The male connector is actuated and shaped like three hooks, which come out from the hemisphere of the ATRON module and grasp on to two passive female connector bars. A connection or a disconnection takes about two seconds. To ensure a reliable connection the connectors can tolerate some misalignment (about half a centimeter) in any direction. Furthermore, the connection is strong, because of the aluminum hooks, and cannot be ripped apart under normal operation.

**Computation, Communication and Sensing** Each module is controlled using two Atmel ATMega128 microprocessors, one on each hemisphere. RS485 communication through a centre slipring is used to connect the two microprocessors. In relation to each connector, a module has an infrared transmitter and receiver. These allow a module to communicate with neighbor modules and sense distance to nearby objects. Because of the many active IR diodes, crosstalk between modules and noise is an issue, which we will address in Chapter 5.

Furthermore, each module is equipped with three tilt sensors that allow the module to know its orientation relative to the direction of gravity. Two encoders are used to control the rotation of the centre axes, one before the gearing and one in the centre slipring after the gearing. The resolution of the encoder is so that a module is able to rotate precisely enough, so that a
4.1. The ATRON Self-Reconfigurable Robot

Figure 4.2: Three different robots for locomotion made from ATRON seven modules each: Snake (left), Car (right) and Cluster Walker (back). The robots can autonomously change between the three configurations.

moved module is aligned to fit in the lattice structure. The encoders are also used as a mean of detecting collisions with the environment or other modules since this will stall the actuator. To control the connections/disconnections the modules sense the current that runs through the male-connector actuators. This allows the module to know when a male-connector is completely extended/contracted.

Energy Supply The modules have onboard batteries. The lifetime is dependent on the use of the module, but may last more than two hours in the worst case.

Lattice of Modules The placement of the connectors are so that the ATRON modules are connected in a surface-centered cubic lattice structure as illustrated in Figure 4.1(b). Like the atoms of a crystal, the lattice is global, meaning that every module sits in the same lattice. The mechanical parts of the modules are mainly build from aluminum. The connectors are designed so that they cannot be pulled apart without breaking metal. The combination of strong modules, connectors and modules that are locked in lattice positions cause that robots build from ATRON modules are structural strong. ATRON robots can carry quite some weight without breaking apart. However, since the module design does not easily allow modules to parallelize their rotational torque, they are generally not strong in terms of exerting force on the environment.

4.1.2 Application Versatility of ATRON

A large range of different robot applications is realizable with the current implementation of the ATRON system. Robots able to move can be used
Figure 4.3: (a) Three degree of freedom robotic arm (first two joints are locked), with a gripper, reaching for the yellow ball. The gripper is held by one male connector and actuated, via strings, with the other male connector on the same hemisphere. (b) Manipulation system assembled from ATRON modules. The box rest on a conveyor surface, in the back a small robot arm is equipped with a camera, in the front a robot arm with a gripper.

for exploration and transportation applications. Locomotion with ATRON robots can be achieved in different ways, see Figure 4.1.2. One way is in the form of a car, which exploits the spherical shape of the ATRON module and uses it as a wheel. Rubber rings on one hemisphere provides the friction. Other robots for locomotion include snakes, walkers, crawlers, as explored by Mikkelsen in [111]. In Part V, we will study online learning of locomotion for such robots.

One application of the ATRON system is as a rapid-robot construction kit. The modules can quickly be assembled in the robot configuration needed to solve a specific task. Another advantage of the system is its ability to transform from one shape or configuration to another. By self-reconfiguring the robot can adapt to the required task and environment, for example changing from a car to a snake to get through a narrow passage.

Many applications also require manipulation. ATRON modules can be assembled as a robot arm, see Figure 4.3(a), which can be equipped with different end-effectors. The ATRON system can also manipulate and transport objects as a conveyor surface. This can be combined with robotic arms, for a complete manipulation system, see Figure 4.3(b). By self-reconfiguring such a manipulation system could adapt to meet the demands on the production line, e.g. with more and longer robot arms. Another possible application is to provide adaptive structural support, for example to an insecure roof in a building or cave. Such applications take advantage of the ATRON modules high strength as a material. The ability of robots to self-repair by replacing defect modules with functional spare modules can be useful in any application. Especially applications that demands highly autonomous
4.1. The ATRON Self-Reconfigurable Robot

robots, such as in space applications, where ATRON modules for example could be used as support structure for solar panels. Physical visualization and 3D-prototyping are examples of the many types of applications that could benefit from a miniaturization of the modules. Application versatility of the ATRON system is further discussed in [12].

4.1.3 Related Work on ATRON Control

Østergaard and Lund used a genetic algorithm to evolve the control and co-evolve initial structure and control for groups of ATRON modules[126]. The groups varied in size from 12 to 20 modules and the evolved control were based on finite state-machines. The task was to produce forward locomotion of the ATRON group by self-reconfiguration (cluster-walk). However, only locomotion which quickly ended up in a local minimum was achieved due to the hard motion constraints of the ATRON modules. Rule-based control for producing cluster-walk behavior has also been explored[127]. Every module in the predefined structure was equipped with a table of hand-programmed rules. Each module compared its local state and configuration of modules with the rule preconditions. If a rule applied the module performed the corresponding action. The rule-based approach were extended in [13], to include generalization of rules and subdivision of behaviors. Rule generalization was in the form of “wildcards” that allowed the same behavior to be controlled using fewer rules. Behaviors, in the form of different rule-sets, were arbitrated using artificial gradients that were triggered by sensory information or special rules. Centralized planning of small ATRON structures, up to 7 modules, has also bee explored[10]. The work compares the RRT-connect and A* planning strategies, and illustrates the point that centralize planning does not scale well with the number of modules, but may be useful on smaller groups of modules. A 4-module meta-module has been developed to control 2D surfaces of ATRON modules[11], such as the conveyer surface in Figure 4.3(b). Finally, to facilitate control development different prototypes of a domain specific languages for the ATRON modules has been developed[149, 45].

4.1.4 Applied Design Principles

The ATRON robot is due to its lattice and modular nature scalable and reasonable versatile. However, it is not simple to extend because of its homogeneous design (partly violate Principle 1 and 2). The module design exploits its neighbor modules to perform self-reconfiguration, both in the connector and lattice design, as well as in inter-lattice movement, where a module is moved by a neighbor module (follows Principle 4). The module design attempts to ignore gravity by designing the connectors to be ideally rigid, which results in a rather heavy module (850 gram) compared to its
strength/speed (violates Principle 3). Finally, the module design is rather complex, although it has just 1 DOF, it has 4 actuated connectors, 2 microcontrollers, 8 communication channels, etc. (violates Principle 5).

4.2 The Odin Deformable Modular Robot

The Odin robot is developed at University of Southern Denmark, based on funding received from Intel Research, Pittsburgh. The concept of the Odin robot is as a rapid robot construction kit from robotic modules, a complete description of the Odin robot can be found in [166, 103].

4.2.1 Design of the Odin Robot

The Odin system is based on passive spherical joints and active links which can be manually reconfigured, see Figure 4.4. Joints are spherical, with a diameter of 55mm. The purpose of joints is to forward power and communication between links connected to the joint. Currently joints are designed so that the links are connected in a cubic close-packed (CCP) lattice structure, each joints contains 12 connectors, each of which is six redundant.

Links are normally cylindrical, with a diameter of 35mm and a length of 110mm. The connector on a link is a lock-and-key mechanism for easy assembly of Odin robots. Connectors contain a ball and socket joint that is spring loaded, which makes robots constructed from Odin modules deformable. Normally links contains a microcontroller and electronics for communica-
4.3. The Catom Ensemble for Programmable Matter

The communication bus of a joint’s bus, and in this way form a single larger bus\cite{60}. This extended bus is categorized as hybrid since it is neither local nor global. Many different types of links can be constructed within the constraints of the design. Each new link type should add a simple functionality to the robot. Functionalities can fall within the following types:

- Power (batteries, solar panels, ...)
- Tools (gripper, paint spray gun, ...)
- Sensing (camera, accelerometer, ...)
- Structure (passive, joints, ...)
- Actuation (linear, wheel, ...)
- Communication (wireless, infrared, ...)

Currently a number of different link functionalities has been developed: linear actuator, passive, ZigBee communication and battery as shown in Figure 4.4(b).

4.2.2 Applied Design Principles

The Odin robot is designed with the relevant design principles from Chapter 3 in mind. The Odin joints and links can be assembled in many different configurations and the number of modules can be scaled up. Further, the system is extendable with new functionalities through new link types (follows Principle 1). The Odin system consist of morphological differentiated links (follows Principle 2), each link is specialized to a simple functionality (follows Principle 5). The flexible connector design is an attempt to exploit physical aspects in several ways (follows Principle 3): i) the robot can deform to adapt to external forces, e.g., when moving through a narrow spaces, ii) it can deform to allow collective actuation of links without requiring that the individual links should to take special care to fulfill kinematic constraints. In the interest of fair comparison of the three systems (ATRON, Odin and Catom), it should be noted that, unlike the other two systems, the Odin does not have to self-reconfigurable, which makes the design principles much simpler to realize.

4.3 The Catom Ensemble for Programmable Matter

The Claytronics Atom (Catom) is a yet-to-be-built micron-scale module, envisioned by the Claytronics project seeded at the Carnegie Mellon Uni-
versity and Intel Research, Pittsburgh. Catoms serve as a platform for the exploration of the programmable matter concept\cite{61, 63}. It has a strong conceptual focus on scalability, where it seeks to develop systems with millions of sub-millimeter scale Catom modules. The long-term goal of this research effort is to produce physical artifacts that can dynamically change shape and therefore enable applications like, telepresence, interactive 3D design, and smart antennas.

4.3.1 Design of the Catom System

**Cylindrical Catom Hardware** Current hardware prototypes of Catom modules are planar with a cylindrical shape of radius 2.2 cm (see Figure 4.5(b)). Around the border of the cylinder are 24 electromagnets that can be energized to attract neighbor Catoms via magnetic forces. This causes one module to spin around another, thereby allowing the group of modules to self-reconfigure and take on a particular shape.

**Spherical Electrostatic Catom Model** In 3D, Catoms are spherical or faceted and can roll across the surfaces of other Catoms. Early prototypes have been constructed at the meter scale, using helium filled balloons with electrostatic surfaces for actuation (see Figure 4.5(a)). Future work is intended to decrease the size to millimeter or micrometer scale using MEMS (Micro-Electro-Mechanical Systems) technology. At such scales, the mechanism of actuation is likely to be electrostatics, which motivates us to define
4.3. The Catom Ensemble for Programmable Matter

Figure 4.6: The electrostatic Catom model we use for our analyses assumes insulated plates positioned near the surfaces of spherical modules. When charged, the plates generate a torque around the point of contact.

a simple electrostatic model of such a module to investigate the potential physical/electrical characteristics of such tiny Catoms.

First, we assume a miniaturized Catom to be constructed as a 5-micron thick shell of silicon. Insulation, to avoid short-circuiting, is assumed to be glass (SiO$_2$) with a thickness of $b = 1 \mu m$. This assumption implies a dielectric breakdown voltage of 200V. Conservatively we select the voltage drop between the faces to be $V_d = 100V$ for the purpose of our experiments.

Second, we assume the spherical surface to be filled with flat square faces (or plates) that can be charged to produce an electrostatic force between adjacent plates on neighboring Catoms. The torque around the contact point between two modules will be given by:

$$
\tau_{catom} = \frac{x}{2a + x} \frac{\varepsilon_0 \varepsilon_r x}{2(\varepsilon_r \theta_d + \theta_r)^2} \ln \left( \frac{a + x}{a} \right) V_r^2
$$

(4.1)

The force one module can use to adhere to another module if their plates are parallel is given by:

$$
F_{adhere} = \frac{1}{2} Q E = \frac{\varepsilon_r \varepsilon_0 x^2}{8b^2} V_r^2
$$

(4.2)

The notation used is shown in Figure 4.6. In practice, this force will be smaller since the plates may not be parallel. Moreover, we do not consider the opportunity to charge several plates on each module at once, which could be utilized to increase the strength.

Under the given assumptions, for a given scale, there exists an optimum angle between faces and thereby an optimal number of faces, when maximizing the size of the torque. The number of faces increases with the radius. Estimates of the required number of faces on the entire sphere varies from approximately 40 ($r = 11\mu m$) to approximately 2300 ($r = 698\mu m$). For each scale, we select the optimal (highest torque) number of faces. We also keep
Table 4.1: Characteristics of Catoms at different scales.

<table>
<thead>
<tr>
<th>Radius ($\mu$m)</th>
<th>Catoms lift (#modules against gravity)</th>
<th>Time to rotate ($T_{\text{rotate}}$) (sec. for full rotation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>698</td>
<td>1</td>
<td>0.055</td>
</tr>
<tr>
<td>83</td>
<td>5</td>
<td>0.0038</td>
</tr>
<tr>
<td>34</td>
<td>10</td>
<td>0.0012</td>
</tr>
<tr>
<td>11</td>
<td>25</td>
<td>0.00027</td>
</tr>
</tbody>
</table>

the insulation and shell thickness as well as the voltage potential constant while varying the module radius.

Smaller modules would be stronger relative to mass and would therefore be able to move faster. This is due to an increase in the surface area to volume ratio and thereby torque to mass, when scaling down. Table 4.1 summarizes some characteristics of Catoms at different scales. For a given radius, the corresponding number of Catoms which one fixed Catom can support in a cantilever is shown (i.e., assembled in a stiff horizontal chain, held static against gravity). The time it takes for one Catom to be rotated 360-degrees around another fixed Catom in zero gravity is also shown. Notice the large gain in speed and strength when scaling down the module size, we will study these effects further in Chapter 6. Although these effects are highly theoretical, future module designs may benefit from them.

4.3.2 Related Work on the Catom System

Although, the miniature Catoms does not exist, several studies have been conducted using simulation. The Catoms are not intended to be powered using batteries since this does not scale to small sizes. Instead Campbell et al. presented a distributed adaptive method of power-sharing[23]. Sources of power and ground are routed between modules using only a single wire between two modules. To self-reconfigure a distributed strategy has been presented[140]. Holes of missing modules are moved from a surface area with has an excess of modules to areas with a shortage of modules. On the development tool side, a distributed debugging strategy[141] as well as several programming languages for controlling self-reconfiguration, has been developed[2, 1].

4.3.3 Applied Design Principles

The Claytronics project, has its own design principles for the Catom design[62]:

- Catom should be self-contained (onboard communication, sensing, etc.)
- Local control, no external computation
- No static power for adhesion after attachment
4.4. Summary

- No moving parts

The first two principles also follows from the principles in Chapter 3. The next principle stems from the fact that micro sized modules due to scale effects can only store a very limited amount of energy. The “No moving parts” principles is in interest of mass production and reliability. Combined these four principles should ensure the systems scalability. Off course, since only mockup prototypes exist, many issues still have to be resolved in the concrete design. Although, the Catoms in this thesis is used for both locomotion and manipulation it is not intended to be a versatile platform but a specialized platform for dynamical physical rendering. Therefore, it does not make sense to evaluate the Catom design based on the design principles from Chapter 3 which scope is for autonomous robots that have to interact physically with its environment.

4.4 Summary

In this chapter, we described three quite different systems. ATRON and Catom are self-reconfigurable, while Odin is manually reconfigurable. Odin is heterogeneous with simple functionally differentiated modules, while the two other systems are homogeneous systems. ATRON and Odin are macro size systems in the centimeter scale, while the Catom is designed as a micro size, sub-millimeter system. Therefore, the three systems highlight different aspects of the research on modular reconfigurable robots. In this thesis, we will use the Catom system to study scalability aspects and we will use the ATRON system to study the self-reconfiguration process and adaptive collective behavior. Since Odin is still in development, we will use it mainly as a conceptual case study that will direct future work. First, in the next chapter, we will study issues in ATRON communication, which serves as an example application of the design principles on control aspects from Chapter 3.
Chapter 5

Reliable Information Exchange

In the practical design of self-reconfigurable modular robots, there are a few critical elements that need special attention: one is connectors, another power and a third communication between modules. In this chapter, we address communication, which is necessary to facilitate cooperation between several modules within the same robot. If the communication is unreliable, it will typically affect the performance of the whole system. Here, we describe the communication of existing systems, discuss the design trade-offs and identify critical issues that are special to self-reconfigurable robots. Further, we address two specific issues concerning communication between neighbor modules. The first issue is automatic neighbor detection that is due to modules self-reconfiguring, whereby the local communication network topology dynamically changes. The second issue is crosstalk between non-neighbor modules, where data packages, send through an infrared communication channel, are received by a non-neighbor module because of reflections. In this chapter, we propose algorithmic solutions to automatic neighbor detection and crosstalk elimination. The algorithms are simple, distributed, self-organizing and robust to follow the design principles of Chapter 3. For validation, they are implemented and evaluated on the physical ATRON system. In conclusion, the algorithms are efficient and effective and we argue that these algorithmic contributions may be applicable to other systems as well.

5.1 Introduction

The purpose of a communication system is to provide reliable information exchange with high bandwidth and low delay between modules. The communication system must enable a module to: i) Detect the local configuration of neighbor modules. ii) Localize itself within the robot. iii) Coordinate with
other modules, e.g., to synchronize actuation patterns, agree on connectivity changes or share sensor information.

Recall the distributed vs. centralized control discussion in Chapter 1, similar considerations also applies to the communication system. Local, neighbor-to-neighbor communication is scalable since the bandwidth of the individual channels do not depend on number of modules. But the overall delay when communicating from one end to the other end of a robot, increases with the number of modules. This last effect can for even small systems be noticeable, e.g., for a ten-module chain, the delay from head to tail will be one second if passing and processing one message takes 50-100 ms per channel, which is a realistic value. In global, any-to-any, communication many communication channels share the same medium. Therefore, the bandwidth of the individual channels will drop when the number of modules is increased. However, if the bandwidth requirement is low the reaction time can become very low compared to systems utilizing local communication.

5.1.1 Design Goals

The overall design goal of a communication system for a self-reconfigurable robot is to provide reliable communication between modules and enable detection of neighbor modules. Here, we summarize the design goals special to self-reconfigurable robots.

**Scalable:** State-of-the art self-reconfigurable robots consist of dozens of modules. We are, however, concerned with robots consisting of hundreds, thousands and eventually billions of microscopic size modules. To ensure scalability, the communication protocol must utilize distributed peer-to-peer communication between neighbor modules. Further, modules are homogeneous in hardware and should be the same in software, again to ensure scalability. Thus, the modules do not have globally unique IDs.

**Robust:** Since dealing with large numbers of modules, the communication protocol must be robust to module failures, like resetting or dead modules. Further, it must be robust to self-reconfiguration, crosstalk, noise etc.

**Self-organizing:** Changes in the network topology due to self-reconfiguration must be detected and dealt with locally while ensuring secure peer-to-peer communication with neighbor modules. Simultaneously, the system must adapt to changes in the communication load and filter out crosstalk messages. These adaptations must be based on local available information only.

**Minimal:** Self-reconfigurable robots are in general small embedded system with limited resources. The communication protocol must therefore have a minimum of memory, bandwidth and computational overhead.
5.1. Introduction

5.1.2 Related Work

Most self-reconfigurable robotic systems use neighbor-to-neighbor communication to detect the topology/configuration of the modules as well as for local coordination. Typically, there is a dedicated communication channel for each connector. In addition, some systems such as the M-TRAN and CKbot systems also have a global bus system running through the whole system. This is convenient for synchronization and centralized control strategies. Further, many systems also have wireless communication for remote control or for inter-module communication, as in the case of the YaMoR system. A critical point is how the communication system can be integrated with the mechanics. Some modules, like M-TRAN, SuperBot and CKbot use a surface area for connectors in which space also can be found for wires routing both power and communication. However, systems such as Catom and ATRON have point-to-point connectors in which there is limited space for wires. Therefore, these systems utilize directional and wireless infrared communication. Other systems, such as Polybot and CONRO, also use infrared communication to detect neighbors while they are not connected, and use the infrared signal actively to guide the alignment of connectors. In summary, the typical modular self-reconfigurable robot have a global bus (often a CAN bus) combined with local infrared communication and add-on wireless communication (often ZigBee).

Some systems use alternative forms of communication. The Odin modular robot is equipped with a hybrid communication system, which allows local busses to be joined to longer, hybrid busses. If all bussed are joined to one, every module in the system is able to communicate with every other module in the system. If all bussed are joined to one, every module in the system is able to communicate with every other module in the system. This allows both local and global communication to be implemented using a single medium, but not at the same time. SwarmBot, a self-assembling mobile robot, use a simple communication system based on colored light and a camera. Every robot can see the color emitted by other nearby robots and can use this information to coordinate behaviors. Using this minimalistic communication system SwarmBot are able to perform rather complex cooperative behaviors.

There are several other examples of simple control strategies, which only require local communication, and can be used to control the behavior of modular robots. Artificial gradients are basically a hop-count distance to a seed module, which can be dynamically found and updated using a simple communication strategy. Although simple, artificial gradients have been found to be highly useful, e.g., for attracting migrating modules toward a specific location in the robot configuration or to setup a global coordinate system within a distributed system. Digital hormone based control, utilize simple messages which diffuses through the robot and may trigger various behaviors in the modules it reach. The local context of a module decides how it reacts to a hormone, why no direct addressing is
needed. For hormone based control an adaptive communication protocol for the CONRO system has been presented by Shen et al. [154]. This protocol can detect changes in topology, due to self-reconfiguration. Artificial central pattern generators (CPG) running on different modules are synchronized using simple local messages. The oscillating output of these CPGs can be used to control the modules actuation to make the robot move [106].

These communication strategies are all robust to some level of communication loss, but if the quality of the communication becomes too low it does affect their performance and for other tasks, e.g. negotiation of connect/disconnect, reliable communication is crucial. In the following sections, we present a practical implementation of automatic neighbor detection and crosstalk elimination as required by the ATRON hardware to ensure reliable communication between neighbor modules.

5.2 The ATRON Communication System

The ATRON communication system must allow two neighbor modules reliably to exchange data packages with each other. Such local (neighbor-to-neighbor) communication scales well with the number of modules if well-designed distributed control strategies are used, since the load on individual communication channels is constant. In contrast, global communication such as wireless, would have scaling problems since the same medium is use by all the modules [213].

5.2.1 Infrared-based Communication Hardware

The ATRON communication system is implemented using an infrared receiver and transmitter for each connector. Infrared is preferred over wired communication since this allows two modules to communicate even if they are not physically connected.

However, crosstalk occurs because the physical spherical design of the ATRON modules did not allow the IR communication channels to be shielded off. This cause the transmitted IR light to be reflected off the metallic surfaces of the modules, which then reach other non-neighbor modules as illustrated on Figure 5.1. This crosstalk communication is highly undesired but has proven extremely hard to remove with solutions such as: non-reflexive film, IR caps for making light more diffuse, lower transmission power, etc. Some of these solutions have reduced the crosstalk problem to some extend. However, no solution has come anywhere close to removing the problem entirely.

A number of experiments have been performed on the unmodified hardware to measure the communication quality between different (neighbor and non-neighbor) modules as shown in Figure 5.1. This is the only configuration where crosstalk occurs in the ATRON system. Because of the ATRON
Figure 5.1: Crosstalk between 4 ATRON modules. One module, M1, sends messages through the IR transmitters that are received by M2. However, the infrared light is also reflected by mechanical parts of M2 causing M3 also to receive the messages from M1 (crosstalk). We did not observe any crosstalk from M1 to M4, but believe it to be feasible that it occur in rare cases.

lattice configuration, any communication channel will be arranged in such a four-module loop. The measured results showed a perfect communication (no noise or byte loss) between both M1 and M2 and between M1 and M3. Note, that communication between M1 and M3 is crosstalk and therefore undesirable. We found no communication between M1 and M4. However, we believe that it can occur in some rare cases. In addition, we cannot rule out that small uncertainties in the mounting of the IR transmitter and receiver on the PCBs, sometimes will make non-neighbor modules communicate even better than two neighbor modules.

5.2.2 Basic Communication Protocol

For comparison and as a foundation for further expansions a simple, custom build, communication protocol has been implemented to provide peer-to-peer communication between two neighbor modules. This basic protocol assumes no crosstalk and a prior knowledge of the local network topology (which channels have neighbors).

Each ATRON module contains two micro-controllers, one on each hemisphere. The two micro-controllers use RS485 to communicate with each other through a center slip-ring. Each micro-controller manages the communication of the four IR channels on its hemisphere by multiplexing a single UART. The micro-controller can listen for IR activity on all four channels at the same time but can only receive and transmit data on a single channel at a time.

The basic communication protocol therefore manages four channels at
a time. It can be in one of three states: listening (all channels), sending (one channel) or receiving (one channel). When listening, the protocol will as soon as some IR activity is detected switch to that channel and start to receive. Errors in data packages are detected using a cyclic redundancy check (CRC). Correctly received data packages will be acknowledged (single byte) by the receiver. Non-acknowledged packages will timeout and be resent. The packages are unnumbered, so if an acknowledge byte is lost a package may be sent and received several times. The process of switching between the listening and sending state is randomized, with some back-off if package collisions occurs in the communication (e.g. data error or receiving while sending).

In the following sections, we will present extensions to this basic communication protocol that enables it to automatically detect neighbors and eliminate crosstalk from non-neighbor modules.

5.3 Crosstalk Elimination and Neighbor Detection

This section proposes a distributed algorithmic solution to detect and remove crosstalk without the need of globally unique IDs. Further, we propose an algorithmic strategy for automatic detection of neighbors, which is necessary due to self-reconfiguration of modules and non-trivial because of imperfect communication. The algorithms are implemented and validated on the physical ATRON modules.

5.3.1 Extended Communication Protocol

Here, we extend the basic communication protocol to deal with self-reconfiguration and crosstalk in a simple, computationally cheap and completely distributed way, which provide the application layer with transparent functionality such as safe sending of a package to a neighbor module and information if a module is present on a given channel.

State of Module, Channel and Neighbor Channels

First we expand the basic communication protocol to include some state information about the module and any detected modules within communication range. A module has a locally unique ID, \( id_{\text{my}} \), which is used by the communication protocol. The next paragraph describes how local uniqueness of the ID is ensured. In addition, each of the eight communication channels has a state. Hence, the module’s communication state is given by its ID and the state of its eight channels:

\[
M = \{id_{\text{my}}, C_0, C_1, C_2, C_3, C_4, C_5, C_6, C_7\}
\]

The states of the channels are independent and we can therefore split their states into two, where two protocols manage four channels each on
different hemispheres. Only the \( \text{id}_{my} \) state must be shared between the two protocols.

Further, the state of an ATRON communication channel is given by the state of up to three neighbor channels (due to crosstalk) which can communicate with this channel \( i \). Only one of those neighbors can be its immediate neighbor, others may be sending data to that channel through crosstalk. The states are identified using the neighbor channel’s IDs. The channel state also include a state variable, \( t_{\text{comp}} \), which is used to compensate the neighbor channel’s strength for the communication load on that channel:

\[
C_i = \{ t_{\text{comp}}, S_{\text{id}1}, S_{\text{id}2}, S_{\text{id}3} \}
\]

In general, the state of a channel must include the state of as many neighbor channels as it can to communicate with (inclusive crosstalk).

Finally, the local state of a neighbor channel, \( i \), is given by its ID and two variables (strength and pingCount) used to measure the communication quality with that neighbor channel:

\[
S_i = \{ \text{id}, \text{strength}, \text{pingCount} \}
\]

The use of these states should become clear in the following sections.

### Automatic Neighbor Detection and Locally Unique IDs

In order to facilitate automatic discovery of neighbor modules a special ping byte is sent at a random time, with on average one ping per \( \Delta t = 20\text{ms} \) on each communication channel. The ping byte is encoded as follows:

\[
PING = \text{header} \quad \text{gender} \quad \text{id} \quad \times \times \times \times
\]

Every module is assumed to have a locally unique ID in the range 1-31 (0 is reserved as a no ID). The gender bit indicates if the sending connector is male or female - such information is useful for the application layer. Below we explain the approach used in details.

**Receive a ping:** We can expect to receive a ping byte every \( 20\text{ms} \) from neighbor modules within communication range. Every time a ping is received on a channel \( i \), from a neighbor with ID, \( \text{id}_{\text{rec}} \), the counter, \( S_{\text{id}_{\text{rec}}}.\text{pingCount} \), is incremented. If the ping was received from an unknown neighbor we construct its corresponding state \( S_{\text{id}_{\text{rec}}} \), potentially removing the neighbor state which has the lowest strength. See Algorithm 1.

**Update channel state:** Approximately every \( \Delta t = 20\text{ms} \) the state of a channel \( i \) is updated, see Algorithm 2. For simplicity this is done in conjunction with the sending of a ping on the same channel \( i \). When updating the channel’s state we first compute the amount of time spend, since last update, listening on that channel (for pings and packages), e.g. \( \Delta t_{\text{listen}} = 17\text{ms} \), and the total amount of time since last update, e.g. \( \Delta t_{\text{total}} = 23\text{ms} \). From this we update the channel’s state variable \( t_{\text{comp}} \) as a moving average of the percentage of time spend listening, e.g. 73% (time not spend
Algorithm 1: Receive a ping byte on channel $i$

**Require:** $id_{rec}$ read from ping byte

if $S_{id_{rec}} \notin C_i$ then
    if $C_i$ is full then
        delete $S \in C_i$ with $\min(S.\text{strength})$
    end if
    construct $S_{id_{rec}}$ from $id_{rec}$
    add $S_{id_{rec}}$ to $C_i$
end if

$S_{id_{rec}}.\text{pingCount} = S_{id_{rec}}.\text{pingCount} + 1$

sending or receiving packages).

Then, the states of the known neighbor channels are updated. The communication strength to a neighbor module is updated as a moving average of the number of pings received from that neighbor (typical 0, 1 or 2). However, since the channel has not spend all its time listening it may have missed some pings, this problem increases as the communication load increases. Hence, we compensate the strength based on the average percent listening time, $t_{\text{comp}}$, as shown in Algorithm 2. After updating the strength, the pingCount state variable is reset. If the strength of a neighbor channel falls below a threshold, $\epsilon_1$, the neighbor is removed. In the implementation on the physical ATRON modules, the floating-point moving average is replaced with a fixed-point version to minimize computational overhead.

Several parameters has to be selected in Algorithm 2. The choice of $\alpha_1$ should match the speed at which the load of the system can change. Likewise the choice of $\alpha_2$ should match the speed at which the system self-reconfigures. For the ATRON communication load changes faster than the system should be able to detect new neighbors, that is why we select $\alpha_1 = 1/16$ and $\alpha_2 = 1/32$. Which means that a neighbor modules will be detected in less than a second, see Section 5.3.2. Likewise, we select $\epsilon_1 = 0.022$ which implies that one ‘noise’ ping will be removed after 10 updates or approximately 200ms.

**Locally unique IDs:** In Algorithm 2, on a given channel $i$, if a neighbor has the same ID as the updating module and its strength is above a threshold $\epsilon_3$ then the updating module will randomly select a new ID. Dependent on the number of different IDs used and average numbers of neighbors this method will very quickly converge so that the modules locally but not globally have unique IDs. This strategy is a distributed one-hop version of the algorithm presented by Zhou et al.[223]. The communication system will adapt to the change of an ID the same as if it were a self-reconfiguration. The effects of changing ID on a module might result in that neighbor modules ‘forget’ the existence of that module, but only for a very short time and most likely not a all (dependent on the choice of $\epsilon_2$ in Algorithm 3). For ATRON $\epsilon_3 = 1/8$, which is a tradeoff between fast updates to ensure local...
5.3. Crosstalk Elimination and Neighbor Detection

Algorithm 2 Update state, $C_i$, of channel $i$

**Require:** $\Delta_{total}$ and $\Delta_{listen}$ since last update

$t_{comp} \leftarrow \alpha_1 \cdot \Delta_{listen}/\Delta_{total} \cdot + (1 - \alpha_1) \cdot t_{comp}$

**for all** $S_j \in C_i$ **do**

$S_j.strength \leftarrow \alpha_2 \cdot S_j.pingCount/t_{comp} + (1 - \alpha_2) \cdot S_j.strength$

$S_j.pingCount \leftarrow 0$

if $S_j.strength < \epsilon_1$ then

remove $S_j$ from $C_i$

end if

if $S_j.id = id_{my}$ and $S_j.strength > \epsilon_3$ then

$id_{my} = \text{newRandomID}()$

end if

end for

Algorithm 3 Is neighbor module on channel $i$?

**for all** $S_j \in C_i$ **do**

if $S_j.strength > \epsilon_2$ then

return true

end if

return false

end for

uniqueness and not being too sensitive to noise.

**Neighbor Detection:** The strength state variable allows the communication protocol to provide the application layer with information about whether or not the module have a neighbor on a given channel, see Algorithm 3. The algorithm simply checks if the channel has any neighbor channels with a strength above a threshold $\epsilon_2$. For ATRON we select $\epsilon_2 = 3/8$ which is below 0.5 since this will reduce the probability of undesirable ‘neighbor knowledge loss’ by the neighbor modules if the local ID is changed. This is because that strength of new ID will rise above $3/8$ faster than strength of old ID will drop from 1 to $3/8$. Note that false-positive does not occur since crosstalk does not occur if the channel does not in fact have a true (physical) neighbor to reflect the signal from.

The use of ping bytes as a mean of detecting, removing and estimating the communication strength to a neighbor module allows the structure of modules to be self-reconfigured while detecting local changes in the topology of the communication network.
Algorithm 4 Send package, $P$, through channel $i$

Ensure: $\exists S \in C_i : S.\text{strength} > \epsilon_2$

- add $id_{my}$ to $P$
- for $k = 0$ to $N$ do
  - select $id$ with $\max(S_{id}.\text{strength}), id \notin P \land S_{id} \in C_i$
  - add $id$ to $P$
- end for
- send $P$ on hardware
- if $P$ is not acknowledge or timeout then
  - schedule resend of $P$
- end if

Cross-Talk Elimination

To the package, about to be send, the algorithm attach its own locally unique ID and the ID’s of N of the channel’s neighbors ($N=2$ for ATRON). These extra bytes send are the only communication overhead involved in cross-talk elimination, see Algorithm 4. The algorithm is designed so that it only accepts (and acknowledge) packages, which it is confident, is send from an immediate neighbor module. Packages from immediate neighbors may initially be rejected, but will eventually be accepted when the states of the neighbor channels have adjusted. This is less serious than wrongly accepting a crosstalk package, since this package then would never reach its indented destination and could cause serious confusion at the receiving end.

Sending a package: When sending a package the algorithm first ensures that it has a neighbor on that channel (by comparing the neighbors strength with $\epsilon_2 = 3/8$). To the package the algorithm will attach its module ID as well a number, $N = 2$, of known neighbor IDs. The IDs attached is the potential immediate neighbor channels on that channel, the neighbors are selected based on their strength. These extra bytes are the only communication overhead involved, see Algorithm 4. For ATRON the three IDs are encoded as two bytes, giving an overhead of two bytes per package.

Receiving a package: Crosstalk elimination is performed as shown in Algorithm 5. If the two communicating channels do not agree that they are neighbors or if the communication strength is too low ($\text{strength} < \epsilon_2$) the package is not accepted (first and second condition in Algorithm 5). Further, the channels must have no common neighbors (third condition in Algorithm 5). This is due to the fact that a module (M1) and its crosstalk neighbor (M3) will also have a common neighbor in its immediate physical neighbor (M2), see Figure 5.1. The intuition is given in Figure 5.2. The algorithm assumes that the amount of crosstalk received from a module two hops away (M4) is low (in fact we have recoded no such crosstalk, but consider it a theoretical possibility). This simple algorithm rejects any received crosstalk
5.3. Crosstalk Elimination and Neighbor Detection

Figure 5.2: Intuition behind crosstalk elimination. Crosstalk is eliminated by comparing the neighbors of the sending channel with the neighbors of the receiving channel. If the channels have a common neighbor, the communication is crosstalk and ignored. This strategy does not work in general for any robot, but does work in general for the ATRON because the modules sit in a lattice. The figure shows three robots (A, B and C) with directional communication channels (triangles). Robot A and B has no common neighbor, why they can communicate. Robot A and C do have the common neighbor B which is why they cannot communicate.

packages. The dynamics of the strength ensures that no crosstalk occur, even when a module is briefly reset, change its ID or self-reconfigures. If a physical neighbor module is turned off or broken, crosstalk can happen since it may reflect the IR signal between non-neighbor modules.

5.3.2 Experiments

The proposed algorithms were first tested in a simulation of ATRON modules containing dozens of modules. Here, we saw that the reactive assignment of local IDs quickly stabilized, neighbors were detected, and that crosstalk was eliminated. Here, we present experimental validation on the physical ATRON modules and on a simple model of two communication channels.

Neighbor Detection

Here, we validate the proposed neighbor detection strategy with communication load compensation. As explained in Section 5.3.1 and Algorithm 1 and 2. Every module emits a ping every 20ms which is received by neighbor modules if they are listening. Neighbor detection is achieved by using two moving averages in combination, one for load compensation and one estimating the for communication strength.
Algorithm 5 Receive package, $P$, through channel $i$

Require: $id_{rec}$ and $ids_{rec}$ read from $P$

if $S_{id_{rec}} \notin C_i$ or $S_{id_{rec}} - strength < \epsilon_2$ then
    return // package is not accepted
end if

if $id_{my} \notin ids_{rec}$ then
    return // package is not accepted
end if

if $(ids_{rec} \cup S.id \in C_i) \neq \emptyset$ then
    return // package is not accepted
end if

acknowledge and process message // package accepted

In Figure 5.3(a) and 5.3(b) results obtained from simulation of a model of two communication channels are shown. A receiving channel will receive a percentage of the pings sent by another channel (percentage given by the communicating load). The model assumes a perfect load estimation and noise free communication between the two channels. The parameters of the system are otherwise the same as for the physical ATRON modules. As can be seen from Figure 5.3(a) an uncompensated version of the neighbor detection system fails to detect the neighbor under high loads. This problem is reduced with the compensated version, see Figure 5.3(b).

The theoretical average time to detect a neighbor is always $600\text{ms}$ independent on the communication load (for the given parameters). The time variation is, however, dependent on the load (percent time not spend listening). For 25% load the expected time to detect a neighbor with one standard deviation range from $540\text{ms}$ to $660\text{ms}$ while it at 75% load range from $460\text{ms}$ to $840\text{ms}$. On the physical ATRON modules, the time for one module to detect another has also been measured under various realistic loads. In 20 trials under a load of approximately 5% we found that the average time to detect a neighbor were $484\text{ms}$ with a max time of $780\text{ms}$ and min time of $372\text{ms}$. The reason that the physical system is somewhat faster than the theoretical model is due to the dynamics of the communication protocol which is likely to shift to another task (than listening) just after receiving a ping.

Crosstalk Elimination

In the following experiment, we compare three communication protocols for their ability to eliminate crosstalk in the ATRON system. The first is a basic protocol which makes no attempt to remove crosstalk, as explained in Section 5.2.2. The second protocol uses reactive local unique IDs, neighbor detection based on pings, strength based elimination of crosstalk, but do
5.3. Crosstalk Elimination and Neighbor Detection

Figure 5.3: Theoretical strength of a neighbor channel as a function of the load on the listening channel (load is percent time not spend listening). (a) No load compensation, note that under high loads neighbors will not be detected. (b) Load compensation is used (as in Algorithm 2) here neighbors will be detected in spite of high loads. The time to detect a neighbor is 600ms. Error bars indicate one standard deviation.

not uses the “neighbor” information to eliminate crosstalk (only use first condition in Algorithm 5). Third protocol is the full implementation as proposed in this paper, it is the same as the second protocol but also make use of the neighbor information to eliminate crosstalk (second and third condition in Algorithm 5).

The setup is shown in Figure 5.1, M1 sends out 100 packages on a given channel with a payload of 1 byte, total package size is 7 bytes inclusive header (3 bytes) and CRC check (2 bytes). For each protocol 20 trials were performed, five trials on each of four permutations of modules. The baud rate were set low (9600 bps) to increase the probability of collisions and thereby noise (to stress test the protocol). In Table 5.1 we report the number of packages received by M2, M3 and M4. We observe that neither the basic or strength based protocols works very well, both accept a lot of crosstalk. The full implementation performs much better, no crosstalk were observed and just 5 percent of the packages are received twice due to a collision between an acknowledge byte and a ping. In none of the trials do M4 receive any packages, but it does add to the IR noise and communication load to the system since it is pinging the other modules. In summary, the proposed algorithm eliminates the crosstalk between ATRON modules.

Self-reconfiguration

The communication system has been applied in the context of self-reconfiguration of several ATRON modules. For this purpose the baud rate was increased to 38.4 kbps which reduces the chance of communication collision and therefore resend of packages. At this baud rate we have found that it
Table 5.1: Crosstalk characteristics of three versions of the communication protocol. Percentage of packages received is reported. Resent packages, due to collisions, results in higher receive rates than 100%. Notice that the full implementation completely eliminates crosstalk.

<table>
<thead>
<tr>
<th>Module</th>
<th>Basic (no check) Mean (Std. Dev.)</th>
<th>Strength Check Mean (Std. Dev.)</th>
<th>Full Check Mean (Std. Dev.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module 2</td>
<td>103%(32%)</td>
<td>62%(16%)</td>
<td>105%(3%)</td>
</tr>
<tr>
<td>Module 3</td>
<td>120%(9%)</td>
<td>63%(16%)</td>
<td>0%(0%)</td>
</tr>
<tr>
<td>Module 4</td>
<td>0%(0%)</td>
<td>0%(0%)</td>
<td>0%(0%)</td>
</tr>
</tbody>
</table>

takes on average approximately 30ms for a package, with a 1-byte payload, to be send from a controller program on one module to a controller program on a neighbor module (the package passes through two RS485 as well as the IR channel in this case). For a 5-byte package, the time has increased to 35ms. The protocol has been tested with a payload of up to 40 bytes. Further, the communication protocol has been tested in a self-reconfiguration scenario with four modules, where it successfully detected appearing neighbor modules dozens of times and eliminated crosstalk hundreds of times.

5.4 Discussion, Conclusion and Future Work

5.4.1 Applied Design Principles

We designed the communication strategy to automatically detect neighbor modules and eliminate crosstalk based on the design principles of Chapter 3. To increase robustness and scalability, our strategy is completely distributed, i.e., modules maintain their own local communication state, and filters away crosstalk packages (follow Principle 8 and 10). The strategy is simplified by assuming that the other modules follow the same communication rules (follows Principle 9). The modules run identical communication programs, with no global unique ID, which allows modules to be dynamically replaced or reconfigured (follow Principle 11 and 12). Overall, the proposed communication strategy follows the relevant design principles of Chapter 3 and illustrates how the design principles can be used to ensure scalability and reliability on top of a distributed and unreliable system.

5.4.2 Conclusion

This chapter describes current implementations, challenges and design goals of communication in self-reconfigurable modular robots. We proposed a distributed and self-organizing communication strategy for dealing with self-reconfiguration and crosstalk. It is based on locally unique ID which are reactively assigned, ping messages send with a fixed time interval and the
exchange of known neighbor IDs when sending packages. The communication system was validated on the physical ATRON modules. Result showed that the communication system was able to detect/forget neighbor modules in less than a second independent on communication load and were able to eliminate crosstalk between modules completely. In conclusion the system is simple to implement and sufficient for ensuring reliable communication between neighbor modules and may be general enough to be used on other self-reconfigurable robots or mobile robots.

5.4.3 Future Work

Relying on the current hardware platform, several improvements could be made to the ATRON communication protocol. For example the throughput of the system could be improved and noise reduced by replacing the current randomized strategy with a self-organizing synchronization mechanism, such as those used in the context of sensor networks[41, 102].

Further, the proposed strategy for neighbor detection and crosstalk elimination may have applications beyond self-reconfigurable robots. Especially in the context of mobile multi-robot systems with directional communication channels, the proposed crosstalk elimination algorithm may be used as one of several means to ensure nearest neighbor communication. In such a scenario the degree to which the algorithm can eliminate crosstalk is still an open question, it is, however, a function of many parameters related to the physical aspects of the communication channels.
Part III

The Scalable Nature of Self-Reconfigurable Robots
Chapter 6

Physics of Myriads of Tiny Modules

It is a common vision in the research community, eventually to realize systems with billions of micron sized modules. However, scaling down the size of modules will have a profound effect on the physical characteristics of both the individual modules and the aggregated robots. In this chapter, we consider scale effects on miniaturizing spherical modules, such as the ATRON or Catom modules, as well as scale effects on self-reconfiguration when increasing the number of modules. A more thoughtful and general review on scaling laws of physics can be found in [181] and on actuation [133].

6.1 Effects of Scaling Down Module Size

6.1.1 Basic Scale Effects

Spherical modules have a surface area, $S$, and a volume, $V$, as follows:

\begin{align}
S &= 4\pi r^2 \\
V &= \frac{4}{3}\pi r^3
\end{align}

(6.1)  \hspace{1cm} (6.2)

Notice that the volume is dependent on the cube of the radius, $r$, while the surface area is dependent on the square of radius. This means that as modules get smaller their surface area increases compared to the volume. Since the mass is proportional to the volume, a module will become lighter compared to its surface area when it decreases in size. Using e.g. electrostatic actuation, this scale effect can potentially be exploited to produce faster and stronger modules.
6.1.2 Miniature Modules

In the following analysis we will use the model of Catom as described in Chapter 4. Recall, that our Catom model are spherical with a faceted surface. The faces can be charged or discharged to produce an electrostatic force between neighbor Catoms. The Catoms are appropriate for scaling since electrostatic forces scale favorably. When decreasing the radius of a Catom its surface area to volume ratio increases and thereby its torque to mass ratio (see Equations 4.1 and 4.2). In essence, smaller Catoms are stronger relative to mass and are therefore faster.

6.1.3 Scale Effects on Miniature Module

According to the Catom model, absolute strength of modules will decrease with module size. However, relative to mass the module’s actuation strength will increase radically. Figure 6.1(a) and 6.1(b) illustrates this as the number of modules a single fixated module can lift against gravity (in a horizontal chain) and can hold against gravity (in a vertical chain). Practical effects, such as misalignment errors, will affect these theoretical results, but the trends may hold in practice if such a system could be realized.

The predicted increased actuation strength may also increase the rotation speed of the modules. Thereby, decreasing the time to perform a basic self-reconfiguration action, such as a 90 degree rotation (see trend in Figure 6.2(a)). This allow individual modules to perform more steps of self-reconfiguration per second (see Figure 6.2(b)). According to this model, absolute velocity of such modules would also increase with smaller size, supporting the intuition that small things can be surprisingly fast and strong.

An important limiting factor of actuation strength is cooling, which is not
6.2 Effects of Scaling Up Module Numbers

To justify an increase in the number of modules it should somehow benefit the performance of the robot. In theory, higher numbers of modules may improve different characteristics of the robots. Versatility can increase since more modules allow for a wider range of possible configurations. Similarly, the redundancy of modules may decrease the dependence on the individual modules, thereby increasing the robot’s robustness (from the ability to self-repair and tolerate module failures). For specific applications such as transformation into exact shapes (e.g., for 3D prototyping), increasing the number of modules while decreasing their size can improve application characteristics such as the resolution of shape.

If our assumptions about miniaturized modules hold, the absolute speed
of self-reconfiguration can be increased with more and smaller modules. Since the absolute speed of modules increase, the same number of modules can be expected to self-reconfigure faster at a smaller scale. For solid structures of modules, keeping the robots volume constant (increase module number; decrease their size) self-reconfiguration can also be done faster if the control strategy is based on moving modules in the volume - not the surface - of the robot. Even if 100% of the modules on the structure surface moves towards a goal configuration, it will still eventually be outperformed by a control strategy where just 5% of the modules in the volume moves. Figure 6.3 illustrates this based on the characteristics of the Catom model, where a cluster-walker structure of 10x10x10cm is moving in a given direction with a calculated velocity. The surface based control strategy is quickly outperformed by the volume-based strategy. This is because as the number of modules increase, an increased proportion of the modules will be located in the whole of the structure not on the surface. Volume based control strategies for self-reconfiguration [188, 164, 140] are therefore more scalable than surface based control strategies.

6.3 Summary

In this chapter, we studied the effects of scaling down the module size and up the number of modules. Scaling down the modules size can potentially be exploited to produce modules, which are faster and stronger relative to mass. If these effects can be realized the speed of self-reconfiguration may increase for constant volume robot, but this is only true if we use a volume based self-reconfiguration strategy.

In the next two chapters, we will study scale effects more specifically. First, in Chapter 7, we will study how module size affects both the optimal
morphology and control for a given task. Then, in Chapter 8, we explore ways to control and organize robots consisting of myriads of modules. In Part IV, we return to the self-reconfiguration process, also with respect to scalability.
Chapter 7

Control and Module Size Interdependence

The interactions between morphology, control and environment defines the behavior of a robot. Specifically, morphology and control are interdependent and can therefore not be designed in isolation. Further, it is often the case that a given design problem can be solved in both hardware and software, but that one solution is much simpler than the other. Therefore, we must design the robots as a whole to keep the overall complexity as simple as possible. In the case of modular self-reconfigurable robots, one important morphological aspect is the size of its modules. In Chapter 6 we saw how module size affected the speed and strength of the modules themselves. Here, we will study how module size affects both a robot’s best morphology and control for a given task as well as the attainable performance.

In this chapter, we study the interdependence between module size, control and morphology. For the task of locomotion of Catom chains, we explore how velocity and best gait type change with the size of those modules. The simulated experiments we report on here examine module sizes from (11µm to 698µm radius) and chain lengths from 3 to 30 modules. All gaits tested were based on central pattern generators optimized using a genetic algorithm and hill climbing. Our results show that scaling affects both the preferred type of gait as well as a chain’s overall performance (average velocity). In summary, there is a tradeoff where larger scales face the challenge of overcoming gravity, while smaller sizes face the challenge of staying in contact with the ground and the friction it provides. We show that in between these two extremes lies a “best” module size for given environmental, physical, and engineering constraints.
Chapter 7. Control and Module Size Interdependence

7.1 Introduction

7.1.1 Modular Embodiment

Brooks[14] and later Pfeifer[137] has long advocated that intelligence cannot be studied in isolation from an embodiment. Instead, intelligent robots must be designed in a holistic, experimental and incremental manner to enable perception and interaction with the physical world in real-time. Further, the control system should be designed based on parallel loosely coupled processes which decomposes the robot behavior into different activities (e.g. avoid obstacle, wander, explore) not different functions (vision, planning, motor control).

In this context, modular reconfigurable robots have a unique role since their morphology is much simpler to modify than conventional robots. Therefore, the robot designer can easily experiment with the morphological design of a robot. This were illustrated by Sims, who were the first to use co-evolution to design both the control and morphology of simulated modular robots[159]. Nevertheless, the possible design space of robots is constrained by the design of the modules. This is one of the reasons why we, in Chapter 3, as a design principle requires that the system should be extendible with new module types (Principle 1). Still, many design choices at the system and module level affects the types of robot that can be constructed, e.g., choice of lattice/chain, connectors, actuation and weight. As we will return to in Chapter 9, such choices affect for example the motion constraints of a module, which affect a system’s ability to self-reconfigure. In this chapter, we consider the effects of module size on the example behavior of snake-like locomotion.

7.1.2 Related Work

In this chapter, we explore the interdependence between module size, chain length and control for snake-like locomotion of Catom chains. Related work which also study the interdependence between morphology and control include work on passive dynamic walkers. Such robots are walking machines that can perform locomotion, often biped, without any or with very minimal actuation[40]. Passive dynamic walkers are a good example of how complexity (and cost) can be reduced if the morphology of the robot is carefully designed. Tradeoffs in connector design has been studied the context of modular robots. In one study, Lyder found that flexible connectors in some cases produced a faster locomotion through narrow spaces[103]. In another study, Shimizu et al. found that heterogeneous connectors produced faster and more adaptive movement in the Slimebot[158]. In the context of stochastic self-assembly Miyashita et al. studied how the morphology of simple modules affected the self-assembly process[112]. Bondgaard and
Pfeifer studied how the morphology (e.g. number of modules) and neural substrate affected the locomotion of a simulated modular robot[9].

Another issue is the control strategy used to produce locomotion in a modular robot. Efficient and effective locomotion has been demonstrated for many different combinations of gaits, configurations, and platforms. Gait control tables[200], role based[172], hormone based[154] and phase automata patterns[222] are some of the control strategies used to make self-reconfigurable robots move. Genetically optimized central pattern generators (CPG) were used to control M-TRAN walkers and snakes[78][77]. Similarly, CPGs controlling the YaMoR modular robot were genetically evolved and online optimized in order to achieve locomotion[106]. These approaches are similar to the approach of this paper. They define the interactions between modules and the periodic trajectories to be followed by the module actuators. Further, these approaches allow gaits to be optimized by adjusting parameters such as frequency, phase shift from module to module, amplitude of trajectory, etc.

This work differentiates itself from the above in that the purpose is not to optimize the gaits and configurations for robots assembled from modules with fixed characteristics. Instead, the characteristics of the modules are varied for a fixed type of configuration (chain) to study the effects of scaling down module size.

7.2 Scale Effects on Locomotion of Tiny Robots

To explore scale effects on locomotion we utilize the model of the Catom module described in Chapter 4 and further studied in Chapter 6. Recall, that the model assumes spherical modules covered with electrostatic surface actuators, and from this we can calculate the maximum torque that one module can exert on another. Three parameters: radius, mass and maximum torque describe the scale dependent characteristics of our modules. Gaits are controlled using central pattern generators (CPGs), which are the artificial equivalent of self-organizing oscillating neurons. Two CPGs run on each module to generate the sinusoidal trajectories for steering the yaw and pitch angles between a module and its neighbor. Six parameters define the gait of a chain. By running a physics simulation, gait parameters are optimized for speed of locomotion by using a combination of hill climbing and a genetic algorithm. The simulations enable us to study the effects of scaling – specifically how the velocity and type of gait depend on the module size and chain length.

7.2.1 Miniature Catom Chains

This chapter addresses what we will loosely term “snake-like” locomotion of chains of miniature spherical modules, which are varied in numbers from 3
to 30. The ability of small ensembles to move is important in a number of situations: A self-assembly scenario could involve modules that are initially separated, but which move in small groups to cluster together and form a connected mass. Another example is moving through tiny cracks or holes exploring a pile of rubble for survivors after an earthquake. A third example would be tiny, swimming modular robots that could find an application in non-invasive microsurgery. Our approach does not scale directly to larger groups of modules. However, much as biological systems use the same basic structures (e.g. cilia or muscle fibers) repeatedly, we envision that small chains of modules can be used as basic building blocks that can be assembled into robots that are more complex, see Chapter 8.

7.2.2 Locomotion of Miniature Catom Chains

Defining a Coordinate System for Catom Pairs

In a chain of Catoms, each non-terminal Catom controls two angles with respect to its two neighbor modules. We assume that the modules are equipped with an accelerometer for measuring the direction of gravity, and furthermore, that modules are able to sense the direction of contact with each of their neighbors, (i.e. the direction vectors pointing from a module’s center of mass to its neighbors’ centers of mass). The angle in the horizontal plane between the two direction vectors is defined as the yaw angle of a Catom. Similarly, the pitch angle is defined as the angle between the two direction vectors in the vertical plane, aligned with the vector pointing from the center of mass to the contact point. Angle values of $\left(\theta_{\text{yaw}}, \theta_{\text{pitch}}\right) = (0, 0)$ correspond to a straight line of Catoms. Both angles can be varied within an interval of ±120 degrees.

Controlling the Connection Angles

By charging and discharging their electrostatic faces Catoms can roll around each other, affecting the yaw and pitch angles between neighbor modules. To control these angles we assume that the modules have a continuous electrostatic surface. This is a reasonable assumption when the number of faces is high. Accordingly, we do not take into consideration the discreteness of the faceted surfaces for purposes of our simulations. Therefore, we can always apply the maximum torque (for a given scale) between any pair of neighboring modules.

An obvious choice for controlling the torque between pairs of modules is a PD or similar type of feedback controller, based on angular error. However, because we want to explore the impact of scaling on modules we desire a single parameter-less controller able to handle modules of various sizes equally well. For this reason, we use a simple binary control of the torques.
Torques for each of the two angles are considered independently, and then combined to a single torque. The direction of the torque for each angle is always towards the desired angle. These two directions (which are orthogonal) are then combined to a single torque axis. That torque corresponds to a pair of forces acting on the surface of two neighbor modules. The directions of these forces are parallel to the line segment joining the centers of the two modules. In the simulation, forces are applied at a fixed distance (10 degrees) from the contact point between the modules. The point of force can then be specified as an angle: \[
\arctan(\frac{\theta_{\text{pitch, error}}}{\theta_{\text{yaw, error}}})
\] selecting the corresponding quadrant dependent on the sign of the errors. The size of the force, and thereby the torque, is independent on the size of errors and is always the maximum for a given radius.

This controller has been verified to control the selected scales equally well (average angle error) in a simulation of a sinusoid trajectory following of a 10 module chain in a gravity-free and frictionless scenario.

Central Pattern Generator (CPG)

CPGs are special neurons found in vertebrates, able to produce a rhythmic signal without any external sensory input. They are used to control muscles for locomotion. A single artificial CPG will produce a sinusoidal oscillating signal, which can be followed by the actuators of a robot. In this work we adopt the model proposed in [72] and further refined in [106], details are in the cited papers and will not be repeated here. This model is based on two coupled difference equations, describing the angle and velocity of the CPG. Coupling several CPGs of equal frequency will make them synchronize their signals to a particular phase shift dependent on the coupling strength between them. If there is no loop in the coupling of CPGs, the phase shift from a parent to a child can be set directly. Furthermore, amplitude and frequency can be selected directly for each CPG.

Gait Parameters

In the chain, non-terminal modules use two CPG’s to control the horizontal and vertical angles (yaw and pitch) between its two neighbor modules. In principle, amplitude, frequency, and a phase offset could be set for each CPG, which could be coupled with every other CPG in the robot. However, to reduce the dimensionality of the problem we limit the number of parameters to just six, summarized in Table 7.1.

One parameter is the frequency at which the CPG oscillates. Frequency is the only parameter selected in relation to the scale of the modules - this is because smaller modules are relatively stronger and therefore tend to oscillate faster. Frequency is scaled to the characteristic time parameter, which is equal to the time it takes for a Catom to make one full rotation
Table 7.1: The six CPG parameters defining a chain type gait.

<table>
<thead>
<tr>
<th>CPG Gait Parameters</th>
<th>Interval</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>[0.1, 2]</td>
<td>periods per $T_{\text{rotate}}$</td>
</tr>
<tr>
<td>Yaw amplitude</td>
<td>[0, 2/3$\pi$]</td>
<td>degree of yaw angle</td>
</tr>
<tr>
<td>Pitch amplitude</td>
<td>[0, 2/3$\pi$]</td>
<td>degree of pitch angle</td>
</tr>
<tr>
<td>Yaw phase shift</td>
<td>[0, $\pi$]</td>
<td>between neighbor modules</td>
</tr>
<tr>
<td>Pitch phase shift</td>
<td>[0, $\pi$]</td>
<td>between neighbor modules</td>
</tr>
<tr>
<td>Pitch to Yaw phase shift</td>
<td>[0, $\pi$]</td>
<td>only at head module</td>
</tr>
</tbody>
</table>

around another Catom in zero gravity. Yaw and pitch amplitude decide the width and height of the oscillations along the chain of modules. Yaw and pitch phase shift decide how many periods there are along the length of the module chain. Finally, a sixth parameter sets the phase shift from the pitch CPG to the yaw CPG at the first module of the chain.

CPGs are coupled to synchronize their oscillations with a phase shift. CPGs are coupled as parent to child couplings, from neighbor to neighbor, from chain head to chain tail. Except at the head module, no couplings are made between CPGs controlling the yaw angles and CPGs controlling the pitch angles.

7.2.3 Experimental Setup

Physical simulation

Experiments are performed in DPRsim, an Open Dynamic Engine (ODE[160]) based physics simulator designed to simulate Claytronics/DPR ensembles. The world consists of a ground surface (coefficients of friction and restitution are 0.7 and 0.3 respectively) and a number of Catoms. Each Catom is a hollow silicon sphere. Friction between Catom modules is infinite, with no slipping when modules rotate around each other. Neighboring modules exert torques upon one another, implemented as a pair of forces acting on the surfaces of the modules. Radius, mass, and maximum actuator torque are as predicted by the electrostatic model. The physical simulation includes collisions, gravity and Stoke’s drag law (in air). Reynolds number is usually below one in the experiments performed. The simulation runs at a timesteps equal to 1/100 of the time required for a single Catom to rotate one full rotation around a stationary Catom in zero gravity.

Executing a gait

Initially a chain of Catoms, of a given length and scale, lies in a straight line resting on the ground. All CPGs are initialized with the six parameters defining the gait, along with an initial CPG state $(x_0, v_0) = (0, 0, 1)$, where $x_i$ decides the angle at timestep $i$ and which avoid a singularity at $(0, 0)$. (Note
7.2. Scale Effects on Locomotion of Tiny Robots

Figure 7.1: GA optimization combined with simple hill climbing is used to improve gaits. The example fitness graph here shows ten different runs of gait optimization being performed on a chain of 10 modules with radius $83\mu m$ (able to lift a 5 Catom cantilever against gravity). The reevaluation of the best GA discovered gait at the beginning of hill climbing causes the fitness drop around iteration 100.

that $v$ in the CPG state vector does not directly correspond to any of the module or chain velocity values.)

Modules are controlled in a distributed fashion. At every time step they exchange neighbor-to-neighbor messages to synchronize their CPGs. Catoms attempt to follow the trajectories generated by the CPGs, by applying forces to the surfaces of neighbor Catoms. After a few oscillations, the CPGs are synchronized and its gait is executed by the robot. Not all selected trajectories can be followed, but we do not attempt to correct this.

Finding Gaits: Genetic Algorithm & Hill Climbing

We optimize the velocity of the gaits, by optimizing the six gait parameters with a genetic algorithm. Each gait parameter is encoded as a gene. We use a steady state algorithm with a binary tournament selection of two parents. A single crossover point is randomly chosen. Mutation is performed on the child, with 10% likelihood a random gene will be replaced by a new random value. The child replaces the weaker of two randomly chosen individuals in the population (binary tournament selection). The initial size of the population is 20 random individuals. Then, 100 iterations of child reproduction, replacement and evaluation are performed. An example run is shown in Figure 7.1.

The fittest individual found in the genetic algorithm is then further optimized using a simple hill-climbing strategy. Small mutations are made to the best-so-far individual until a better individual is found. The process is
repeated until there has been no improvement for 25 iterations of mutation and evaluation.

Fitness evaluation of the gaits is based on the horizontal velocity of the chain. We measure a chain’s velocity as sum of horizontal distances moved by its center of mass in the duration of 20 CPG periods:

$$fitness = \sum_{i=1}^{20} \frac{P_{cm,i} - P_{cm,i-1}}{20 \cdot Frequency \cdot T_{rotate}} \text{ (meter/seconds)}$$  \hspace{1cm} (7.1)

The smallest-scale modules often make little contact with the ground — due to their high mass/torque ratio most actions send them flying. We want to avoid locomotion gaits which only touch the ground very rarely, since a chain out of contact with the ground is out of control and likely to consume most of its energy lifting. Therefore, we assign zero fitness to gaits which at the time of evaluation are too far from the ground (with “too far” defined as every module more than 0.5 radius from the ground). The average gait velocity, used in diagrams, are measured during 200 CPG periods to limit the amount of noise.

7.2.4 Results

We performed a total of 3750 gait optimizations using the strategy described above. Experiments were performed for 15 different chain lengths ranging from 3 to 30 modules and for 25 different sizes of module radius varying from 11 \(\mu\text{m}\) to 698 \(\mu\text{m}\). Module sizes were selected based on the number of Catoms that a single Catom can lift against gravity when arranged in a horizontal chain. The 25 sizes correspond to Catoms able to lift 1 to 25 other Catoms, this selection strategy results in a greater density at the smaller Catom sizes. Ten optimizations for average velocity were performed for each combination of length and size. Each optimization yields a single, fastest gait found for that length and size. We then use the characteristics of these gaits to analyze scaling effects on both the types of gaits and on the average velocity.

Scaling Effects on Velocity of Locomotion

Velocity of the Catom chains were affected by scaling as shown in Figure 7.2. Larger modules moved relatively slowly, due to their limited force to mass ratio. As modules get smaller, locomotion increases in speed until some critical size around 80 \(\mu\text{m}\). Here, the problem of keeping in contact with the ground reduces their performance dramatically. We also observe that the velocity depends on the length of the chain. Especially around the fastest scale (80 \(\mu\text{m}\)), chains in the interval from 6 to 16 modules move faster than longer chains of more than 17 modules. The fastest gaits move with an average velocity of 0.11m/s.
7.2. Scale Effects on Locomotion of Tiny Robots

Figure 7.2: Density plot showing the average velocity as a function of chain length and module strength/radius. Each combination of chain length and module size is optimized 10 times using a genetic algorithm and hill climbing. We observe that there exists an area with highest average velocity around 6-16 modules and radius (54µ to 110µm).

Another issue is the degree to which gaits are periodic (can maintain their velocity). For a given gait we measure this as the standard deviation of the velocity divided by the mean velocity (see Figure 7.3). As can be observed in the diagram only a small fraction of the found gaits are periodic, and these corresponds roughly to the area (size and length) of the fastest gaits. In general, longer chains and smaller modules are less likely to be periodic, due to more complex module-to-module and module-to-environment interactions.

Scaling Effects on Types of Gaits

Although the gaits are specified with only six parameters, we have observed a great diversity of gaits, many of which can be recognized as similar to those found in nature. Figure 7.5 illustrates a few typical example gaits optimized for different scales and for different lengths.

Some typical gaits can be recognized by considering the normalized difference between the yaw and pitch amplitudes (see Figure 7.4). Gaits for large modules do generally not have large horizontal movement because of their limited strength. Similar gaits for small modules oscillate only in the horizontal plane to avoid jumping. Gaits for intermediate scales will often be almost perfectly “round” in the sense that they oscillate in both axes strongly, producing a spiral. This spiraling type of gait is somewhat similar to the sidewinding gaits of snakes (see Figure 7.5(b)). Almost regardless of
Chapter 7. Control and Module Size Interdependence

Figure 7.3: Density plot showing the average standard deviation of velocity divided by the mean velocity for gaits. This is a measure for how periodic a given gait is. Periodic gaits are more likely for Catom sizes corresponding to the area of the fastest gaits.

scale, caterpillar-like gaits seem to be appropriate for longer chains. These produce forward locomotion by having a vertical wave traveling along the length of the chain (see Figure 7.5(d)).

Alternatively, gaits that cannot be recognized from Figure 7.4 include gaits for short chains which typically hop (in smaller or larger hops) or roll (as in Figure 7.5(a)), where some of the modules are used as wheels. For the smallest modules, most of the gaits found are non-periodic, however, for chain lengths from around 6 to 12 modules there is an alternative strategy. This strategy (see Figure 7.5(c)) is similar to the movement of seahorses. The modules are aligned in a 45-degree angle to the ground and only a few modules touch the ground. Locomotion is achieved with a relatively slow moment of the tail - pushing on the ground. See Figure 7.6(a) for an overview of where the different gaits were typically observed.

Parameter Sensitivity

We performed a series of experiments to analyze the gait’s sensitivity to parameter change in the physics and Catom models. Using the fastest gait found, a chain consisting of 14 Catoms with radius $65\mu m$ (average velocity of $0.11 m/s$, see Figure 7.6(b)), each physics/Catom parameter was varied and the impact on maximum velocity (over 50 CPG periods) and rise time (to reach 90% of max velocity) was evaluated. Experiments were performed with all independent parameters kept fixed, all except one which was varied uniformly across the interval shown in Table 7.2. For each independent
Figure 7.4: Density plot showing the average of a gait-type metric (see text) as a function of chain length and module strength/radius. For larger modules and longer chains, the optimizations tend to find gaits with mainly vertical motion (caterpillar). For small Catoms and short chains, the optimization finds gaits with horizontal motion. In between, predominantly spiraling gaits are found — such gaits are also the fastest seen.

Table 7.2: Correlations between physics/Catom parameters and max velocity/rise time

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Interval</th>
<th>Max Velocity</th>
<th>Rise Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coeff. of Friction</td>
<td>[0.1, 1.0]</td>
<td>Small (r= .15)</td>
<td>Small (r= 0.11)</td>
</tr>
<tr>
<td>Percent of Torque</td>
<td>[0.2, 1.8]</td>
<td>Large (r= .95)</td>
<td>Medium (r= .47)</td>
</tr>
<tr>
<td>Coeff. of Restitution</td>
<td>[0.0, 0.9]</td>
<td>Large (r= -.98)</td>
<td>Medium (r= .56)</td>
</tr>
</tbody>
</table>

parameter approximately 100 experiments were performed.

Table 7.2 also shows correlation coefficients that express the strength of the relationship between the parameters. We observe that changing ground friction has almost no effect on the gait in the investigated interval (r= .15). Outside this interval, from friction coefficient 0.1 to 0, max velocity drops very quickly. Torque and max velocity have a strong correlation (r= .95). An increase in the available torque increases the max velocity (from 0.013m/s at 20% to 0.15m/s at 180%) and increases the rise time (from 8 to 17 CPG periods). In addition, the coefficient of restitution has a large effect on and a strong correlation (r= -.98) to max velocity. Max velocity is fastest (0.15m/s) and rise time shortest (9 CPG periods) when the coefficient of restitution is 0. For large coefficients of restitution the chain looses contact with the ground and max velocity drops off linearly. For instance, when the coefficient of restitution is 0.9 maximum velocity drops to 0.038 m/s and
Figure 7.5: Typical gaits at different lengths and scales. (a) Short chains often move by rolling or hopping. (b) Spiraling gaits are typical for medium to long chains of medium size Catoms. (c) Seahorse like gaits or pure horizontally oscillating gaits are typical for the smallest Catoms. (d) Caterpillar like gaits are typical for large modules and medium to long chains.
Figure 7.6: (a) An conceptual overview of the types of gaits found to be the fastest at different scales and chain lengths. (b) An illustration of the fastest gait found amongst 25 different module sizes and 15 different chain lengths. It was optimized using a genetic algorithm followed by hill-climbing. The gait is a spiraling motion, similar to the sidewinding of snakes, and its average velocity is $0.11\text{m/s}$. The chain contains 14 modules and the module radius is $65\text{µm}$.

rise time increase to 31 CPG periods. Finally, we also measured the effect of drag. Drag slows the max velocity of the gait from $0.16\text{m/s}$ (no drag) to $0.12\text{m/s}$ (drag), the difference is statistically significant.

Although both torque and restitution have large influence on max velocity and rise time, these effects occur over relatively large intervals. In conclusion, the gait investigated does not seem to be particularly sensitive to small changes in the physical parameters of the system or the environment.

7.3 Discussion, Conclusion and Future Work

7.3.1 Applied Design Principles

Although, the behavior of the robots studied in this chapter are simple, we have still applied the design principles from Chapter 3. The CPG based control strategy is distributed since there is no central controller (follows Principle 8). CPG’s are synchronized using local messages and a module’s actuation is based on the state of its CPGs and the relative angles of its neighbors (follows Principle 10). Although, all communication and actuation are local, a global behavior emerges in the form of snake-like locomotion (follows Principle 6). Every module is running an instance of the same program (follows Principle 12). Therefore, the behavior of the robot would not change even if the modules were dynamically reconfigured (follows Principle...
ple 11) - note, that this is not completely true since the head module is manually marked to avoid a head/tail conflict (direction of CPG couplings). Further, the robots are not critically dependent on its modules, if a chain breaks in two then both halves would continue to move (follows Principle 7). The benefits of following these principles were that the module controller was simple to implement and simple to optimize with a genetic algorithm. Therefore, it should also be relatively uncomplicated to transfer the control strategy to a real hardware platform.

7.3.2 Conclusion and Future Work

This chapter addressed the interdependence between morphology and control in modular reconfigurable robots. Specifically we experimented with scaling effects on gait and velocity of locomotion for simple chains of Catom modules. Scaling in terms of length of chain and size of modules was explored based on a physical simulation of electrostatic Catoms. Modules were controlled for locomotion using central pattern generators. Gaits were optimized at varying module sizes (11\(\mu m\) to 698\(\mu m\) radius) and length (3 to 30 modules), using a combination of genetic algorithm and hill climbing.

Our results indicate that very high-velocity gaits 0.11m/s or 1749 module radii per second can result given our assumptions. We observe that there seems to be an appropriate chain length and module size for locomotion - because small modules are uncontrollable since they tend to fly while larger modules are too weak to move. We expect a similar tradeoff to exist for other physical implementations of miniaturized robots and for other tasks such as self-reconfiguration. Future work should study this. In conclusion, morphology and control are indeed interdependent. Therefore, a holistic robot design approach should be taken to minimize the overall system complexity.
Chapter 8

Organizing a Myriad of Modules

Scaling of self-reconfigurable robots involves increasing the number of modules (ultimately to billions) while decreasing the size of the individual modules (to hundreds of microns). As an effect of scaling, a module will have a decreased ability to affect the global behavior of the robot, since the actuation strength of a module will decrease compared to the weight of a constant volume robot. This stresses the need for collaboration between the modules to achieve a desired macroscopic behavior, e.g., locomotion or manipulation. Further, the overall complexity of a modular robot increases with the number of interacting modules. Therefore, designing a functional robot involving a large number of modules becomes increasingly difficult.

This chapter addresses the question of how to organize modules to achieve scalable control of robot behavior. We propose a novel biologically inspired hierarchical approach to organize and control modular robots. The purpose of our approach is to decompose the complexity of assembling and commanding a functional robot made of numerous simple modules by introducing a hierarchy of structure and control. The robots we describe incorporate anatomy inspired parts such as muscles, bones and joints, and these parts in turn are assembled from modules. Each of those parts encapsulates one or more functions, e.g. a muscle can contract. Control of the robot can then be cast as a problem of controlling its anatomical parts rather than each discrete module.

First, we demonstrate the proposed anatomy-based organization on the ATRON system. Second, we experiment with simulated Catom robots using gradient-based primitives to control parts of increasingly complex robots, including snake, crawler, cilia-surface, arm-joint-muscle and grasping robots. Further, we discuss how the design of the Odin system is inspired by the principles of anatomy-based organization. We conclude that, although impractical with most state-of-the-art macroscopic sized modules it may potentially
Figure 8.1: How can we make functionality scale up with increasing number of modules? - E.g. the anatomy of a few-module moving robot cannot trivially be scaled up, since the actuation forces generated by the individual modules must somehow be parallelized and scalable joints must be introduced.

be viable for organizing myriads of miniaturized modules or for system, such as Odin, designed with an anatomy-based organization in mind.

8.1 Introduction

8.1.1 Module Organization

When a number of autonomous units, e.g. humans, cells or modules, need to collaborate to solve a task, some level of organization can be used to guide the process of solving the task. The organization will subdivide and assign task to specialized individual units and define relationships that can be used to coordinate the different units. The organization may be tall or flat depending on the number of hierarchical layers it contains. It may also be top-down if decisions are imposed from the top of hierarchy or bottom-up if decisions emerge from the individual units. This corresponds to centralized control, which is typical for tall organizations and distributed control typical for flat organizations.

A class of systems, related to modular robots, is swarm robots that have similar, yet, different organizational constraints. Just as modules need to collaborate, so must swarm robots collaborate to perform a task. However, since their interactions are simple the required level of coordination is lower. While, individual swarm robots are self-sufficient in terms e.g. of mobility, modules are physically connected and generally unable to perform any task alone. Therefore, modules must cooperate (e.g. for actuation) to achieve coherent behavior of the robot. The need for physical collaboration is increased with the number of modules, since the task that a single module can perform becomes increasingly smaller as the number of modules increase (see Figure 8.1). When organizing modules in a hierarchical organization an important consideration is how it copes with failures at the lowest levels of its
organization. If such failures are handled in a top-down fashion the system may not scale. Another central decision is how the modules are grouped, e.g. based on spatial position or function. This constrains how the tasks within the robot can be subdivided and assigned to individual modules.

8.1.2 Related Work

Large-scale self-reconfigurable robots have for obvious reasons so far only been studied in simulation. Here, the challenge of achieving a desired global behavior can be divided into a number of different classes based upon intended application: i) morphology for its own sake, ii) function from morphology, and iii) function from morphological transformation. Control strategies for the first class often focus on ultimate physical resolution, i.e., scaling up in number of modules and down in module size, and hence on centralized or distributed mechanisms for controlling shape [164, 140, 91]. This can enable applications such as 3D visualization. The second class involves tasks requiring mechanical interaction with the environment, where the robot by virtue of its shape provides some desired functionality. Examples include structural supports, as we shall see in Chapter 10, and grasping[6]. The third class includes tasks in which the robot’s function may emerge from a continuous change of shape, such as cluster-flow locomotion[127, 19].

Little related work has been done on scalable whole-body behavior of fixed-topology modular robots. For the task of self-reconfiguration control, a distributed control strategy with a flat hierarchical structure is often used. In some cases meta-modules are introduced, which is several modules organized as one, to increase the capabilities of the basic modules. For example we use an ad-hock, temporal limited, organization of meta-modules to control ATRON self-reconfiguration in Chapter 9. Control strategies based on planning often assumed that a single centralized control unit controls the actions of all the other modules in the robot. In addition, in the context of planning, meta-modules can be used to facilitate the planning process. In the meta-module based planner described by Bhat et al. nested layers of meta-modules allows the number of hierarchical layers to be adjusted to the number of modules in the system[5].

The papers on scalability, cited above, all have self-reconfiguration as a primary focus. In contrast, this chapter addresses the challenge of scalability in the context of fixed-topology robots, which achieve their function without using self-reconfiguration, but instead rely on local actuation of the modules. This is similar to Campbell’s and Pillai’s work on collective actuation, in which groups of Catoms act together in fixed-topology structures which can change physical aspect ratios (i.e., stretch and bend) [21, 22]. In the context of this work, such a structure can be considered an anatomical part to be utilized together with other anatomical parts to create functional robots.
Chapter 8. Organizing a Myriad of Modules

Figure 8.2: The organization of cells as tissue types, tissue types as organs and organs as animals serves as inspiration for the anatomical structures of modules. Examples are bones, muscles and neurons. The purpose is to achieve functional and fast responding robots from myriads of modules.

8.2 Anatomy-Based Organization

In Chapter 6 we saw that smaller modules potentially can become stronger and faster relative to their size, but also that the individual module will have less direct effect on the global behavior of the robot. In this section, we propose a method to organize individual modules to achieve global collective behavior in myriad module robots. The remainder of this chapter studies this organization method. First, we explore and discuss the method based on a number of ATRON anatomical parts. For Catoms, we propose control primitives for anatomical parts and present simulation experiments with robots developed based on the organization method. Finally, we discuss why the Odin robot is well suited for the method.

8.2.1 Biological Inspiration

Modules for self-reconfigurable robots are inspired by biological cells, in the sense that they can self-organize, self-assemble and self-repair on the module-level to emerge properties such as autonomy, morphological adaptation and robustness on the robot-level. In principle, artificial creatures could be evolved directly from these modular artificial cells. However, in practice this is currently intractable. We believe that further biological inspiration is useful as a first step towards realizing the same range of behaviors as those shown by multi-cellular animals.

Taking a hint from the biology of animals at the lowest level, physics and chemical processes emerge to cells that differentiate and form an organizational hierarchy of tissue types, organs, organ systems and complete functional animals. We observe that such hieratical differentiation of cells may inspire an differentiation of modules for myriad-module self-reconfigurable robots (see Figure 8.2).

A scalable anatomy represents a higher hieratical level of organization,
positioned in between the module-level and the robot-level. Like cells, the role of the individual modules can be differentiated to undertake specialized functionalities needed within the robot. To ensure scalability, the anatomical parts of such an anatomy must consist of structures assembled from modules that scales to large numbers of modules. We anticipate that such structures may support the realization of responsive and functional fixed-topology robots assembled from myriads of miniaturized modules.

8.2.2 Engineering Perspective

From an engineering perspective, our method to organize and control the modules of fixed-topology modular robots is modular and hierarchical. It is modular in the sense that it utilizes a set of anatomical parts, which encapsulate complexity and provide a simple interface to the rest of the robot, e.g., a muscle can contract or relax and should be connected with tendons in each end. Our approach also defines a hierarchical relationship from robot to anatomical parts to modules that isolates complexity within three layers. This modularity and hierarchy is reflected in both the control and the structural organization of the robots. The anatomy-based organization encapsulates the complexity of many basic tasks at the module level while at the same time provide design abstraction at the robot level. Tasks can be subdivided to several levels of the hierarchy: i) at the lowest level the module’s task is to maintain the integrity and function of the anatomical part, and ii) at a higher level the task of an anatomical parts is to perform its function within the robot to realize the overall behavior. Consequently, complexity is encapsulated in anatomical parts, which also defined the subdivision of tasks.

Our design strategy is to keep the control of the individual module as simple and local as possible. To do so we construct a set of simple functional roles that modules play. Modules playing the same role are combined into anatomical parts with well-defined function and structure. Robots are then constructed from anatomical parts and controlled using simple primitives that define the interaction between different anatomical parts, e.g. to allow a sensor in one part to activate a muscle reflex in another anatomical part of the robot.

8.3 Exploring the Anatomy Concept with ATRON

In this section, we present a series of biological inspired anatomical structures assembled from ATRON modules. The aim is not to demonstrate this as a practical approach to control the macroscopic sized ATRON, but merely to study the construction of such anatomical structures to increase our understanding of the basic tradeoffs and serve as inspiration for future work on other systems. In addition, the presented anatomical structures are
not unique for the ATRON but could reasonably be implemented on most self-reconfigurable robots.

8.3.1 Anatomical Parts

ATRON-Nerve

Self-reconfigurable modular robots are already equipped with the ability to perform computation, sense the environment and communicate with other modules to cooperate. That these abilities are important for self-reconfigurable robots to be functional is generally agreed upon. Not surprisingly similar abilities can also be found in animals for much the same reasons.

In animals, to orchestra the trillions of cells, different forms of coordination take place between cells. Certain forms of coordination are slow such as chemical hormones through the bloodstream. Others are faster, such as through electrical synapses between neurons. In self-reconfigurable robots, the main mean of coordination is communication, which is either local, global, or a hybrid. Coordination between ATRON modules are in the form of explicit infrared communication between neighbor modules, which is fine for small robots. However, for large robots neighbor-to-neighbor communication may introduce a delay too long for fast responding robots. Global communication on the other hand does not scale well if high numbers of communicating modules uses the same medium. An alternative is hybrid communication where physical communication busses can be created within the structure of modules when needed [60]. In combination, these three types of communication form a scalable solution to information exchange within the robot.

In animals, sensory neurons are used to sense the environment. Similar self-reconfigurable modules are equipped with sensors. The ATRON modules are equipped with different types of sensors for measuring acceleration, orientation and for sensing of internal states. This collection of sensors could be further extended e.g. to include the sensing of light, heat,
touch, sound and chemicals. In addition, structures of miniature modules can be assembled to achieve aggregated sensing functionalities (e.g. bend, vibration and sight using encoders and light sensors). Again, it seems that self-reconfigurable robots can be extended with the necessary sensors when scaling up the number of modules.

Finally, each module in a self-reconfigurable robot is able to perform computation using their on-board microprocessors much as animals use networks of neurons for processing information. Again, this does not seem to be a limiting factor when scaling up the number of modules.

As we see from these simple examples, nature and engineers have come up with different solutions to solve similar problems. Nevertheless, for a class of problems, namely those concerning whole body movements of myriad-module robots, engineering solutions have not yet been found. Although, it has already been solved by nature.

**ATRON-Bone**

As were the case with computation, sensing and communication, scalable rigid structures can also be assembled from most existing self-reconfigurable robots. Rigid structures bear a similarity with bones that support the weight of the animal body and enable movement. Some essential properties of bones are therefore their high strength and light weight.

ATRON modules are well suited for such strength related functionality. Modules sit connected in the lattice much like the atoms in a crystal. Each module is interconnected with up to eight other modules through strong aluminum connector hooks. Therefore, a structure of modules has a relatively high strength. The mass density of this ATRON material is approximately
Figure 8.5: Hinge-joint, which can be assembled with ATRON modules. (a) Hinge joint pattern. (b) Hinge joint has a single degree of freedom and scales along the rotational axes.

720 kg/m$^3$, for bone the mass density is about 1900 kg/m$^3$. Thus, the ATRON material is lighter than bone. It is however also much weaker than bone: yield stress for the ATRON material is 0.067 MPa [125] and more than 50 MPa for human femoral bone [59]. Although, such properties might improve with decreased module size it is clear that bones are not feasible on the macro size ATRON system.

Another similarity between biological and ATRON bones are that both are able to self-repair. As we shall see in Chapter 11, in simulation an ATRON-bone was first partly broken, compromising the strength of the bone. Then an emergent control strategy (relying on meta-modules) self-repaired the bone, largely recovering its strength. Figure 8.4 illustrates a bone-like structure of ATRON modules as well as the surface centered cubic lattice that the modules sit in.

In summary, most self-reconfigurable robots can construct rigid bone-like structures. However, at least for macro size ATRON the basic characteristics are inferior compared to biological bones, although both have the ability to self-repair.

**ATRON-Joint**

Self-reconfigurable robots are generally able to sense, compute and communicate, and to some extent form rigid bone-like structures. However, this alone does not enable myriad-module robots to produce whole-body movement. One potential approach to realize this is by utilizing scalable joints and muscle based actuation. Joints are needed to allow one ATRON-bone to be rotated relative to another ATRON-bone using the contractive force...
8.3. Exploring the Anatomy Concept with ATRON

Figure 8.6: A socket-ball joint has three degrees of freedom and scales in three dimensions. It is crucial to minimize the friction between socket and ball for the joint to be efficient. The lattice can be realigned using ATRON-muscles as anchors.

of ATRON-muscles. A single module may be considered a rotational joint, but this does not scale up well, since this single module should be able to withstand the weight of the bones and the forces generated by the muscles.

We can assemble a hinge joint which has improved strength compared to the single module joint since it distributes the forces on a larger number of modules (see Figure 8.5). This hinge joint has the feature that the modules on both sides of the joint always can return to the same global lattice. This is important in order to have complete reversibility when changing from one shape to another.

Ball-socket type joints (Figure 8.6), cannot be assembled from ATRON modules without separating the lattice of the ball from the lattice of the socket. One solution to maintain reversibility is to use ATRON-muscles as anchors between the ball and socket. Such muscles can function as bridges for communication, transporting modules and can by contraction realign the lattices. This technique equally allows the construction of other types of joints, e.g. saddle joints. Friction between the ball and socket could be minimized by careful design of the module shape combined with lubrication.

For few-module robots a single module can straightforwardly be used as a joint, realistically also the hinge joint can be used for somewhat larger robots, however, to use the ball-socket type joint for myriad-module robots seems impractical because of its complexity and potentially infeasible (e.g. because of friction and geometrical constraints).
Figure 8.7: ATRON-muscle made from 8 ATRON modules. The structure contracts by forming a compact helix shape.

Figure 8.8: Experimentally found contraction strength as a function of contraction length for a 6 module ATRON-muscle. Notice that the strength is highest when the ATRON-muscle is fully extended.
8.3. Exploring the Anatomy Concept with ATRON

Muscles are able to contract and as a result produce a force, which can move bones. Potentially joints, muscles and bones can be combined to produce whole-body movement of myriad-module robots.

ATRON modules are not designed with muscles in mind. It is, however, possible to assemble structures of ATRON modules that exhibit the contractile functionality of biological muscles. Long chain structures where each module (except the end modules) are connected to two other modules, one on each hemisphere can contract by forming a compact helix shape (Figure 8.7 and 8.9).

Experiments have shown that this type of ATRON-muscle can contract by a factor of 4.2 or 76%. In its extended form the cross-sectional area is 11x11cm, in its contracted form it is 21x21cm. Contraction forces are strongest ($\approx 160N$) when the ATRON-muscle is fully extended and decrease rapidly as it shortens (Figure 8.8). The maximum force delivered by the muscle is independent on the chain length. This force level is higher than the force a single module can exert, because the angle between the modules and the direction of contractive force works as a gearing. The strength of an ATRON-muscle can be scaled up by having a number of muscles in parallel just as muscle fibers in biological muscles (see Figure 8.9(c)). The main limitation of the ATRON-muscle is that it exerts a poor force/weight ratio. An ATRON-muscle of length 8 (see Figure 8.7) is just barely able to lift its own weight but, as discussed in Chapter 6, a miniaturization of the modules may improve the force/weight ratio.

It might be possible to assemble similar muscles in different ways using most existing self-reconfigurable robots. However, the issue of low force/weight
Figure 8.10: Two ATRON-skin patterns, removing modules from the structure of modules adds degree of freedom to the surface. However, ATRON-skin is not able to perform the function of its biological counterpart.

radio is a general issue, which must be solved before such muscle-based actuation becomes practical.

**ATRON-Artery**

Similar to biological cells requiring oxygen and glucose to survive, modules need energy in the form of electricity to function. Most self-reconfigurable robots are equipped with onboard batteries and some are able to share power between modules through the connectors. The ATRON modules have batteries and in an earlier version had power sharing through the connectors-hooks. This feature has however been removed to simplify the design and remove the risk of short circuits. As the number of modules increase, it becomes increasingly impractical to charge modules individually. Hence, miniaturized modules will need power sharing and the robot will probably need to be able to collect energy from the environment to achieve energetic autonomy. Scalable methods for one-wire power-sharing has been presented by Campbell et al. [23]. Alternatively, instead of utilizing power sharing, energy could be received wirelessly e.g. through radio or microwaves but the amount of energy that can be received in this way is limited.

**ATRON-Skin**

Animals use skin as a shield from the environment. Similarly, protection from the environment might be necessary in myriad-module robots. The nearest to a skin that we can construct from ATRON modules are sheets of modules. Such sheets can within a limited range bent if certain of the inter-module connections are disconnected (see Figure 8.10). A sheet made from a grid of ATRON-muscles can both bend and stretch. However, both
types of skin are probably not practical on a physical system.

8.3.2 Discussion

In this section, we explored the method of anatomy-based organization with the homogeneous macroscopic ATRON system. We assembled structures of ATRON modules in an attempted to construct anatomical parts that could be the artificial equivalents of biological nerves, bones, joints, muscles, arteries and skins. We anticipate that such anatomical parts will become necessary when scaling up the number of modules in ATRON robots. However, in most of the cases we did not succeed in constructing anatomical parts even remotely comparable to the functionality of their biological equivalents. For example for the skin, the question becomes: how can we from modules assemble soft, strong, bendable, stretchable, waterproof protective skin? - The short answer is that we cannot.

We observe that at least two characteristics of the ATRON is the root of this problem. First, the speed and strength of the ATRON modules are insufficient for parts such as muscles and bones. In Section 8.4 we will apply the method on the simulated Catom system, which due to its microscopic size is superior in this respect. Second, the ATRON modules are homogeneous why a single module has to comprise all the necessary functionality and material properties. Hence, most of its functionality is unavoidably suboptimal. Therefore, it seems inevitable that morphological adaptation and differentiation of modules are necessary to construct the many different anatomical parts with the same system. This point is further discussed in the context of the heterogeneous Odin system in Section 8.5.

8.4 Anatomy-Based Organization of Catoms

In this section, we study an anatomy-based organization of the Catom system. We described gradient-based control primitives for controlled anatomical parts, define a number of anatomical parts, and present experiments on increasingly complex robots that display behaviors such as locomotion, manipulation, collective actuation and grasping. Figure 8.11 gives an overview of our anatomy-based method and the developments of this section, from the perspective of control and morphology.

8.4.1 Anatomical Control Primitives

In this subsection, we explain the control primitives utilized at the level of anatomical parts. We control the internal organization of the anatomical parts as well as the interactions between different anatomical parts by using a combination of simple artificial reflexes, synchronization based on central pattern generators (CPG) and sensor feedback. Special seed modules
Figure 8.11: Our proposed method decomposes the organization and control of a modular robot into three layers: Robots are assembled from anatomical parts. Anatomical parts are assembled from modules. Robots are controlled using primitives that control its anatomical parts. Anatomical parts are controlled by assembling them from modules playing a corresponding functional role. The labels show the concrete implementations for the Catom system.

initiate these control primitives, which works across groups of modules by utilizing gradients. At a lower hierarchical level, the modules are performing local actuation and sensing as defined by their functional role, which will be described in the following subsection.

Gradient

Artificial gradients are basically a hop-count distance to a seed module[120]. All modules initially have a gradient value of zero. Then, a seed module, by communication, emits a gradient with some value. If a module receives a gradient-value, $G_{rec}$, which is higher than its current value, $G_{cur}$, it will set $G_{cur} = G_{rec} - 1$ and send $G_{cur}$ to each of its neighbors. In this way, the gradient constructs a breadth first search tree. Here, the neighbor module with the highest gradient-value is denoted the parent - if more than one exists a random one is selected. Modules with gradient-values lower than $G_{cur}$ are denoted children and modules with the same gradient value are called siblings. Figure 8.12(a) illustrates an artificial gradient on a group of modules.

Gradients are not used directly as a control primitive in this work. Rather, gradients form a component of the other control primitives, described below. Their role is to control the flow of and carry information between modules. Some modules become seeds by emitting a gradient of a particular type and containing some further information. A gradient affects the behavior of nearby modules if the modules are sensitive to that type of gradient. Modules playing a particular role are not sensitive to all types of gradients, but all modules pass all gradient types on to neighbors. Gradient
Figure 8.12: Illustration of primitives for organization and control. Note that the primitives are spatially local, since they are limited by the corresponding gradient value. Several different gradients can be active at once in the same module. (a) A hop-count gradient is used as basis for the other primitives. (b) Central pattern generators (oscillating neurons) are coupled to achieve synchronization from parent to child in the breadth-first three formed by the gradient. Such CPGs are used to control module actuation. (c) Sensor information is aggregated by seed modules that emit a sensor-gradient. Sensor information will be communicated up the gradient toward the seed. (d) A seed module may emit a reflex-gradient that allows it to turn on or off a behavior (e.g. actuation) in nearby modules sensitive to that particular type of reflex.
messages are sent to neighbor modules periodically (every \( n \) timesteps).

**Time Synchronization**

Synchronization of modules is often necessary, for example, to produce periodic locomotion gaits. A central pattern generator (CPG) running on each module can be used as a clock or periodic actuation pattern for control. A CPG consists of two coupled difference equations, which respectively describe its two states: angle and velocity\([72, 106]\). The angle is a sinusoidal oscillating signal, which can be synchronized with other CPGs by coupling their states. In practice, CPGs are easily coupled by communicating the two parameters between neighbor modules. Frequency, amplitude and the phase-shift of each such coupling form additional parameters for the system. In general, synchronization will only work if coupling links include no loops, and to ensure this we utilize a special CPG-gradient, emitted by a seed and setup the direction of coupling in a group of modules so that it forms a loop-free tree (see Figure 8.12(b)). Each CPG-gradient message contains a label, a gradient value, and the state (angle and velocity) of the CPG. The label defines the type and allows several CPG-gradients to be active at the same time. CPG couplings, using the received CPG state, are only received from the gradient parent module. We allow for several different types of CPGs to be active in the system at the same time, e.g., for controlling different actuated degree of freedom. In Section 8.4.3, CPGs are used to control cilia that are used for locomotion and distributed manipulation.

**Sensor Aggregation**

Sensors combined with gradients provide a convenient way for a seed module to retrieve information sensed by nearby modules. A seed emits a sensor-gradient, which is limited to some hop-count. Sensor gradient messages include four pieces of information: a sensor type label, an accumulated sensor value, a module counter, and a gradient value. Modules that have the requested sensor information and are within reach of the sensor gradient, will update the gradient message with their sensor values as they forward each gradient message. Thus, each module sums up the sensor values and module counts of its children, adds its own sensor value, and adds one to the module counter before forwarding the update to its parent. Every module knows the accumulated sensor value and the number of sensor modules in its gradient sub-tree. Figure 8.12(c) illustrates how the sensor information flows up the gradient toward the seed, which can then react to the collected values, e.g., by turning on or off a reflex. Section 8.4.3 demonstrates this technique with a whisker that activates a grasping reflex.
Reflex Activation

A reflex is a behavioral primitive, which allows a seed module to request a response by nearby modules. The reflex controls the behavior of modules sensitive to that type of reflex. For example, a muscle-reflex can make modules playing the role of muscles contract or relax. The state of a reflex is controlled by a reflex-gradient emitted by a seed module. The reflex-gradient message includes a label (type of reflex), a truth-value (reflex on/off) and a real value (some reflex parameter). The state of a reflex can then be controlled by the seed, for instance, based on collected sensory data. A seed can set the state of a reflex on or off to request a response from nearby modules. Note, however, that the seed does not control how the modules respond. The response of a module will depend on its functional role and whether or not it is sensitive to the type of reflex. Figure 8.12(d) illustrates that a seed module controls a reflex, which allows it to affect the behavior of a nearby module. In Section 8.4.3 we use reflexes for controlling the contraction of muscles (time activated) and bending of hinge-joints (sensor activated).

8.4.2 Anatomical Parts

In the previous section, some primitives for controlling anatomical parts were introduced. In this section, we present a small library of anatomical Catom parts, including the parts’ morphological structures and corresponding functional roles. Then, in the next subsection, we give examples of how these anatomical parts can be assembled into robots.

Muscle

We construct muscles able to contract by connecting Catoms in a chain.

**Role:** A reflex controls the behavior of a muscle module. If the reflex is off the muscle will simply adhere to neighbor modules with an electrostatic force at the point of contact. If the reflex is on the module applies electrostatic actuation to minimize the angle between a child module and its parent module. That is to move its two neighbor modules closer together.

**Anatomy:** Muscle modules are assembled into chains, which will contract if the reflex is turned on. Several muscles can be parallelized to increase contraction force.

Cilia

Motile cilium is a hair-like structure, which extends from the surface of a cell and beats in an oscillating pattern, for example, to transport unwanted objects away from the lungs of humans. A similar structure can be con-
structured from a chain of Catoms. The control and morphology is almost identical to that which produces snake-like locomotion in Chapter 7.

**Role:** The cilium module oscillates relative to two of its neighbor modules by following the sinusoidal trajectories generated by its two CPGs. The angle state parameters of these two CPGs steer the yaw and pitch angles between the module and two of its neighbors. Cilium modules adhere to any neighbor modules playing a different role.

**Anatomy:** Modules are assembled into a chain, which will oscillate in a waving or spiraling pattern. A CPG-gradient controls the coupling between the CPGs of neighbor modules as explained in Section 8.4.1.

### Bone

Bone like structures can be constructed from a lattice of Catoms.

**Role:** Each module in a bone will pair up those of its neighbor modules which are almost positioned 180 degree opposite each other. For each of these pairs it will apply electrostatic actuation in an attempt to move them so that they are completely opposite. This will maintain the lattice structure of a bone. Bone modules will simply adhere to neighbors that have no opposite.

**Anatomy:** Bone modules are assembled into a solid lattice structure that has opposite modules such as simple cubic lattice or cubic close packing (CCP).

### Tendon

A tendon is needed in-between a muscle and a bone, to avoid that the muscle twists the bones.

**Role:** Tendon modules apply adhesive forces to their neighbors.

**Anatomy:** Tendons are assembled as a chain with one end connected to a bone and the other end connected to a muscle. Several tendons can be combined in parallel to increase tension strength.

### Hinge-Joint

Hinge-joints provide a single rotational degree of freedom joint between two bones. The hinge-joint can also actively bend.

**Role:** A reflex controls the state of the hinge joint. If the reflex is off the module adheres to its neighbor modules. If the reflex is on the hinge joint module adheres to its siblings, and actuates the parent and child modules toward an angle specified with a reflex parameter. The direction of the actuation is controlled using a coordinate system constructed from the direction of the siblings and a child module.

**Anatomy:** Hinge joints are constructed from two chains of modules placed side-by-side (in a simple cubic lattice). The length of the chains, \( N \), is also the width of the joint. Two bones connected with a hinge joint
will have N common connection points which limits the strength of the joint (will not scale to very large bones).

**Whisker**

A whisker is a bending sensor constructed from Catoms to provide feedback from the environment.

**Role:** A whisker module will actuate its parent and a child so that they are as close as possible to 180 degrees from one another. The error of this angle (between parent and child) is reported as the module’s sensor value.

**Anatomy:** Whisker modules are assembled in a chain, which will be somewhat stiff and seek to maintain a straight posture. A seed module can collect the bending error by using the sensor aggregation primitive.

### 8.4.3 Experiments with Anatomy-Based Catom Robots

This section presents simulated experiments on locomotion, manipulation and parallel actuation. The robots described are listed in order of increasing complexity, as measured by the number of anatomical parts and in the number of types of parts. There is a large degree of structural reuse from one experiment to the next. For each setup, the dimensions of each part, the role of each module, and for some functional roles/parts, specific CPG parameters must be manually established. Thus, each module knows its functional role from the outset, and begins the experiment in an appropriate position with appropriate links to its neighbors. In addition, some modules are manually selected to become seeds for reflexes, and the conditions activating those reflexes must be defined.

Our experimental platform is simulated Catoms that have a radius of 65\(\mu\)m. According to the model in Chapter 4, such Catoms are able to support 6 other Catoms in a cantilever against gravity, and requires 2.8 milliseconds to rotate 360 degrees around another fixed Catom in zero gravity.

**Snake-like Locomotion**

This experiment utilizes a cilium part to achieve snake-like locomotion. A 7-Catom cilium chain initially lies flat on the ground. One of the terminal modules is a seed, which emits a CPG-gradient to direct the coupling of the CPG from one module to the next. Initially the snake oscillates out of synchronization, but after a few cycles it synchronizes. For the snake, illustrated in Figure 8.13(a), CPG parameters are adjusted so that it moves as a caterpillar via a vertical wave traveling from tail to head (the seed). In 3000 simulation timesteps the snake moves approximately 62 catom radii, corresponding to an average velocity of 0.048 meters per second (based on 5 experimental trials).
Crawler Locomotion

A crawler can be constructed by expanding the snake with two additional, shorter cilia-chain parts. The three cilia chains, connected as shown in Figure 8.13(b), comprise a 15-module crawler-type robot. The central chain (“spine”) and side chains (“legs”) are programmed with different CPG parameters, causing the robot to move forward with a gait similar to butterfly swimming strokes. The crawler moves 61 Catom radii in 3000 timesteps, corresponding to an average velocity of 0.047 meters per second (based on 5 experimental trials).

Cilia Surface for Distributed Manipulation

In this experiment a surface of 697 bone modules are assembled in a CCP lattice in a disk with radius 15 Catoms. On top of this disk, 177 short cilia (two modules long) are distributed across the bone surface. In total 1051 Catom modules, but just two types of anatomical parts, are used in this experimental setup. The bone module at the center is a seed, which emits a CPG-gradient that covers the entire surface.

When a trial is started, the cilia are initially unsynchronized. After a short time (less than 500 timesteps), they self-organize to beat in a synchronized pattern. Then a solid object (rectangular box) is dropped onto the cilia surface at a random position within 25 Catoms radii of the seed, and with a random orientation about the vertical axis. The box weighs 20 Catom masses, has a size of $8 \times 8 \times 1$ Catom radii, and is dropped from a height of 12 Catom radii. The box is moved by the oscillating cilia, and the direction of movement depends on the parameters of the CPGs. For the purpose of this experiment, two CPG parameter sets were constructed: one, which attracts the box towards the seed module and one, which repels
8.4. Anatomy-Based Organization of Catoms

Figure 8.14: A surface for distributed manipulation is constructed from bones and cilia modules. Depending on the CPG parameters utilized, a box can be repelled away from or attracted toward (shown) a gradient-emitting seed module placed at the center. The small inserted image illustrates how the two-module cilium beats, while sitting on top of a surface of bone modules.

Figure 8.15: A solid object (box) is dropped with a random position and vertical orientation on top of a cilia surface. A seed module is placed at the center. (a) CPG parameters cause the seed to repel the box. (b) CPG parameters cause the seed to attract the box (40 trials shown for each case).
Figure 8.16: Two Catom bones, connected by a hinge joint are actuated by a muscle connected to the bones using tendons. The muscle is initially relaxed before it contracts, activated by a reflex.

the box away from the seed module. The only difference between the two parameter-sets is that the yaw and pitch angles of the cilia oscillate in phase for the attraction type and out of phase for the repulsive type.

For both the attractive and the repulsive cilia surfaces 40 trails were performed. An example trial of the attractive type is shown in Figure 8.14 and the resulting motion tracks from all the trials are shown in Figure 8.15. For attractive motion patterns, the box was within our success criteria of 6 Catom radii from the seed after 3000 timesteps for 33 of the 40 trials. For the repulsive motion pattern, 37 out of 40 trials had the box fall off the edge of the cilia platform within 3000 timesteps. Several seeds, potentially of different types, can also act on the same surface for a more general-purpose distributed manipulation surface.

**Muscle Actuated Arm**

In this experiment, we use bones, tendons, muscles and a joint. The setup consists of a vertical bone \((2 \times 2 \times 16\) modules), a horizontal bone \((2 \times 2 \times 10\) modules), hinge-joint \((2 \times 2\) modules), two tendons (length 3 modules) and one to six muscles connected in parallel (length 12 modules). Figure 8.16 illustrate the setup with six muscles. All the modules are affected by gravity except the vertical bone, which is fixed. From one of the tendons, a seed module emits a reflex-gradient. Initially the reflex is \textit{off} and the muscles
Figure 8.17: Elbow angle of the arm is shown as a function of time. The average and standard deviation of ten experiments on muscle contraction for 1 to 6 muscles are shown. The arm is initially resting; contraction takes around 30 milliseconds due to the small module size (high strength/low mass). Notice that adding more muscles improves the contraction.

Relaxed. Muscle modules are sensitive to the reflex and will therefore start to contract when the seed module, at a particular time, turns the reflex on. Notice, that non-muscle modules are unaffected by the reflex state since they play a different role and are not sensitive to the reflex-gradient.

As shown in Figure 8.17 the completeness of contraction can be increased by adding parallel muscles (from one to six). Observe that the effect decreases as more muscles are added. In general, large-scale self-reconfigurable robots must utilize collective actuation of many modules.

**Whisker Feedback for Grasp Reflex**

This experiment utilizes a whisker constructed of Catoms as a way to sense the environment. Four fingers are attached to a base of bone modules. The fingers are assembled from modules playing the role of hinge joints. A three module whisker is attached to the bone base (see Figure 8.18). The bottom whisker module emits a reflex-gradient and a sensor-gradient. This seed collects the sensor value, which is the angle deviation from straight up. If the average sensor value exceeds a certain threshold the seed turns the reflex on, causing the hinge joints to bend. Thereby, the robot can grasp a falling object.

We performed experiments with a falling box of size $5 \times 5 \times 5$ Catom radii weighing the same as 50 Catoms. The box had an initial position of 15 Catom radii above the fingers. In each trial, the orientation of the box was randomly varied. We repeated the trial 100 times with three different methods for controlling the activation of the reflex:
Chapter 8. Organizing a Myriad of Modules

Figure 8.18: A falling object hits a whisker, which triggers a seed module to activate a reflex, which makes the hinge joint modules bend and as an effect grasp the box. The fast response from the grasping robot compared to the velocity of the falling object is due to the small time-constants of such small Catoms.

- No reflex: Never bend the fingers (ignore sensor).
- On/Off reflex: Fingers bend when sensor bends above a threshold, relaxed otherwise.
- On reflex: Fingers bend and keep bending if sensor at any point get above threshold.

As a metric of performance, we recorded the rate of grasp success/failure. We also recorded the time elapsed until the box touches the ground for failed grasps. The results are summarized in Table 8.1. With the No reflex method the box does by chance not touch the ground in 10% of the trials (it ends in a stable resting state on a finger). The percentage of grasping successes is significantly higher for the On reflex, than the two other methods (47% compared to 10% and 31%). Also, the time before impact in the “failure” cases is significantly longer than for the other methods. Indeed, the On reflex method outperforms the On/Off reflex method largely because it does not let go of the box if the box shifts during the act of grasping (see Figure 8.18).
8.5 Odin and Anatomy-Based Organization

In the previous sections, we studied anatomy-based organization of the ATRON and Catom systems. Although, the approach showed some promise to control the Catoms, we anticipate that the scalable anatomy approach might be more fertile if it was incorporated in the basic design of the modules themselves. We expect such a module design to be heterogeneous. Biological cells are after all not homogenous; in the adult human more than 200 cell types exist. This is different from most self-reconfigurable robotic systems that use homogenous modules (as ATRON and Catom). This means that the modules solely can adapt its function to meet its role in the robot. However, morphological specialization of the modules is likely to be necessary to solve the problem of scalable functionality. One approach is a stem-cell approach with homogeneous modules that have the ability to specialize its morphology to the functional role. However, such a module will presumably be very complex to design and build. An alternative approach is multiple module types designed specifically for the functional roles. However, we want the number of modules to be minimal, due to the challenges of construction and generality. A scalable anatomy can be used as a framework for identifying the required functionalities of such specialized module.

Recall, that the Odin is a system with focus on heterogeneity, which allows us to extend the system with new module types. The different module types are highly specialized, each containing only a simple functionality (see Chapter 4). The system lattice can be changed from CCP to any other lattice by having several types of joints. Further, the connector system allows both rigid and deformable structures to be constructed. We speculate that these characteristics of the system will allow us to more effectively apply an anatomy-based organization and thereby enable us to scale up the number of modules as well as the versatility of the system[166]. The Odin system is still in development, in future work we will study and evaluate such possibilities.

<table>
<thead>
<tr>
<th></th>
<th>Grasp Success (percent)</th>
<th>Time of Impact (timesteps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No reflex</td>
<td>10%</td>
<td>1284</td>
</tr>
<tr>
<td>On/Off reflex</td>
<td>31%</td>
<td>1691</td>
</tr>
<tr>
<td>On reflex</td>
<td>47%</td>
<td>1885</td>
</tr>
</tbody>
</table>

Table 8.1: Grasping performance
8.6 Discussion, Conclusion and Future Work

8.6.1 Applied Design Principles

The work behind this chapter has been the seed of several of the design principles described in Chapter 3. First, we observed that most state-of-the-art self-reconfigurable robots already had the infrastructure for realizing scalable nerves for collaboration, bones for structural strength and arteries for distribution of energy. However, when considering the ATRON system we were only to a inadequate extend able to produce scalable structures for actuation (muscles) and the relative movement of body parts (joints) and not at all able to produce protective tissue (skin). From these observations, Principle 2 follows naturally. It states that modules should be morphological differentiated, so that we can optimize the design of the modules to the function it must perform (e.g. within an anatomical part). In addition, an anatomical part provides a specialized functionality to the robot. Similarly, a module should provide a specialized functionality to the anatomical part (as follows from Principle 5). The design of Odin has, amongst other things, attempted to fulfill these design principles.

With respect to the control of Catom robots, we have only followed the design principles to some extent. At the lowest level, that of the individual modules, the control is completely distributed and based on the local context (follow Principle 8 and 10). Further, the simple behavior of an anatomical part emerges from the behavior of its modules (follows Principle 6). In addition, every module within an anatomical part runs the same program, which allows a module to be dynamically replaced. However, since modules are manually marked as a specific role, they cannot be moved between different anatomical parts (partly follow and partly violate Principle 11 and 12). Future work should develop methods to automatically select the appropriate role of a module depending on its local context. At the higher level of anatomical parts, the control is centralized around seed modules. Although, all experiments performed only had one seed module, it is possible to decentralize the control with more than one active seed module. To insure scalability and reliability, future work should find control strategies for anatomical parts to better meet the design principles.

8.6.2 Conclusion and Future Work

This chapter reported on experiments using a simple hierarchical approach to structure modular robots. Our approach composes anatomical parts to organize and control the overall behavior of fixed-topology self-reconfigurable robots. Our method supports reuse of anatomical parts, which we demonstrate by simulating robots able to move, manipulate, and respond to their environments. The experiments show that our approach is versatile enough to apply to several different types of robot, though the extent to which the
presented set of control primitives can allow behaviors that are more complex is still an open question. Further, although our approach here shows promise for future sub-millimeter Catom modules, it is impractical for existing macroscopic modular robots, such as ATRON, because it requires both too many modules and stronger actuators in strength/mass terms. To address this mismatch, we are currently working toward a novel heterogeneous modular robot, Odin, which will incorporate a hierarchical morphology approach at the design level of individual modules. We anticipate that Odin will allow us to increase the number of modules and the robot’s behavioral complexity along the lines described in this chapter. Future work includes increasing the complexity of physical robots (i.e. Odin) in terms of number of modules and behavioral diversity.
Part IV

The Self-Reconfiguration Process
Chapter 9

Unconstraining Motion Constraints

In this chapter, we examine motion constraints of lattice based self-reconfigurable robots. We describe the different types of motion constraints, and how they are addressed in related work. Then, we introduce a meta-module based control strategy to deal with the motion constraints of the ATRON system. Our strategy is novel in the sense that a meta-module is temporal limited and emerges from the environment created by other modules, move on the surface of other modules and stop at a new position. The flow of meta-modules, from one place to another on the structure of modules, realizes the desired self-reconfiguration. We compare six different meta-module types composed of ATRON modules. Variations of meta-module morphology and meta-actions are investigated for its ability to shape-change the robot. We conclude that two of the investigated meta-module types are able to shape-change the robot to an acceptable extent and we select one meta-module type which we will use for further experiments in Chapter 10 and 11.

9.1 Introduction

9.1.1 Types of Motion Constraints

A lattice-based modular robot can self-reconfigure if its modules are able to move in the lattice comprised of other modules. For a module, this involves disconnecting from some of its neighbor modules to enable movement to another lattice position where it can attach itself. However, motion constraints limit a module’s ability to move within the lattice of modules. Here, we summarizes several different types of motion constrains, which all adds to the complexity of the control problem.

**Kinematics constraints:** Any lattice-based system has a kinematic model that defines to which adjacent lattice positions a module can move.
from a given lattice position. The concrete kinematics is very system specific. Generally, purely simulated systems (e.g. cubic model) are much less constrained than physical systems (e.g. ATRON).

**Blocking constraints:** One module may block the movement of another module. Even if the blocking module is not at the same lattice position as the other module is moving towards.

**Support constraints:** Often a module requires support from neighbor modules to perform a move. In some system, e.g. ATRON, the modules does not have the necessary actuation to move itself. Therefore, a neighbor module must move the module. Although, the move is valid according to the kinematic model, it might not have the right neighbor modules to support it.

**Connectivity constraints:** Throughout the entire move, as well as in the new lattice position the module must stay connected to the rest of the robot. Connectivity is often simple to maintain at the level of the individual modules. However, the module must also take into account that any disconnection may potentially disconnect other groups of modules from the robot. Often, connectivity constraints are more complicated if the system utilize a male/female connector design.

**Physical constraints:** A module may be too week to perform a specific movement, e.g., against gravity or a movement may somehow break the robot, for example if too much weight is put on a given connector.

Even if we manage to deal with the motion constraints at the local level, higher-level constraints must also be dealt with. Modules may be trapped in local minima on their migration to a goal position. In addition, locked configurations in an area of the robot may emerge since any module able to move has migrated away and the rest of the modules are now locked due to motion constraints. In addition, global physical stability of the robot should be maintained during the self-reconfiguration process.

### 9.1.2 Related Work

In most related work on control of self-reconfiguration, the general problem considered is finding and performing a sequence of module movements to self-reconfigure from a start configuration to a goal configuration. All related work on control of self-reconfiguration must somehow deal with motion constrain. Here, we describe different strategies.

One centralized approach is to use planning and search for a sequence of actions which morph a starting configuration to a goal configuration[215, 183, 91]. This strategy take into account the motion constraints of the system, however, the problems are that the search space explodes with the
number of modules in the structure[30] and that the control strategy is sensitive to inaccuracy in its model of the system. Therefore, planning works best on systems with few modules[10].

In some cases the problem can be simplified by introducing an intermediate configuration, typically a line, which the robot self-reconfigures into, before self-reconfiguring into the goal configuration[144, 47]. In this way, a planner that can plan from a line to any configuration can also solve the reverse problem and thereby guarantee completeness.

A distributed approach is to use cellular automata style local rules, where a module selects its action based on the physical condition in its local context[16, 17, 123, 128, 13]. Such rules have been found sufficient to generate behaviors such as cluster-walking, but have not yet been shown able to solve the general problem.

Another distributed approach is to consider each module as an autonomous agent and let itself move closer towards the goal configuration. In such distributed control strategies a global coordinate system can allows a module to locate itself within the robot, artificial digital gradients can guide the movement of modules and stochastic movements deal with local minima[120, 7, 211, 164, 167]. Such solutions often only works if there are more than a few dozens modules.

Scaffolding of the structure of modules can be introduced to remove local minima problem as well as simplifying the connectivity, blocking and support constraints[164]. However, a good scaffold may not exist for a specific system and it may not always be convenient to use, since it decrease the granularity of the modular lattice. A similar strategy is to use attachment rules to constrain the positions on where a module can attach itself. This keeps a self-assembling process from getting stuck in local minima[194].

A meta-module is a group of modules that, combined and seen as a single entity, has different control characteristics than an individual module. Meta-module based control strategies have been used on various realistic types of platforms [145, 187, 139, 183, 109]. In general, meta-modules are used to unconstrain the motion constraints of the base modules at the cost of decreased granularity.

In this and the following chapters, we will use a meta-module based control strategy on the ATRON system. Our approach is different from previous work on meta-modules in that our meta-modules are autonomous, forms in an ad hock manner and have a limited lifetime.

9.2 Emergent Meta-Modules Control Strategy

In this section, we describe a control strategy based on emergent meta-modules, which we will use to control the ATRON system.
9.2.1 What is a Meta-Module?

A meta-module is composed of a number of modules collaborating to achieve the common task - to move the meta-module. Seen from the outside a meta-module is considered a single acting entity or agent. We define a meta-module type by its morphology and by the meta-actions it is able to perform:

- **Morphology**: Meta-modules of a given type are composed of a specific number (one or several) of modules, which are interconnected in a specific way.

- **Meta-Actions**: Meta-modules of a given type can perform a specific number of different meta-actions. Meta-actions are composed of a sequence of basic module actions, which are performed by the modules part of, or neighbor to, the meta-module. Basic module actions of the ATRON are rotation, connection and disconnection.

9.2.2 Meta-Module Life-Cycle

A meta-module starts its life by emerging from the structure of modules; that is, a module or group of modules agrees to form a meta-module. Then the meta-module starts performing a sequence of meta-actions, which result in movement of the modules that comprise the meta-module. At some point, the meta-module decides that it is time to stop and the modules become passive once again. This approach is similar to the division, migration and death of biological cells. The problem of self-reconfiguration is then to control the flow of meta-modules from one place to another on the structure of modules. In our implementation, the life cycle of a meta-module consists of 3 phases:

**Emerge**: Based on local information modules can decide to become a meta-module, if they are interconnected in a valid meta-module configuration. When they emerge the modules are no longer regarded as a part of the structure but as an autonomous agent moving on the surface of the structure.

**Move**: In this phase the meta-module moves on the surface of the structure, that is, the modules that are currently not part of a meta-module. Meta-modules are not allowed to move on other meta-modules as they might try to move themselves. The movement of meta-modules is based on attraction points that define the shape of the desired global configuration. The goal of the meta-module is to minimize its distance to an attraction point.
9.2. Emergent Meta-Modules Control Strategy

Stop: At some point in time the meta-module might reach an attraction point or discover that it is not possible for it to do so. Then the meta-module will stop and the modules cease to collaborate. When stopped the modules are no longer regarded as a moving agent but as a part of the structure such that other meta-modules can move on its surface.

This life cycle of the meta-modules can be repeated and the modules can at a later time again be part of a different meta-module. The local autonomous control based on attraction points makes it possible for a large number of meta-modules to be active at any given time increasing the speed of the shape change.

9.2.3 Attraction-points as Task Specification

We use attraction-points to control the flow of meta-modules from one place to another on the structure of modules. Attraction-points are virtual points in space, whose positions are known to the modules. Meta-modules are attracted by attraction-points and move toward them if possible. Attraction-points are used to specify tasks for the self-reconfigurable robot. E.g. if the robot is to change its shape to meet some specifications this could be done by providing the robot with a set of attraction-points in the desired shape. Two types of attraction-points have been used in this work: inhibiting and non-inhibiting. An inhibiting attraction-point turns off if a module is placed at its position, then meta-modules are no longer attracted by that point. Non-inhibiting attraction-points always attract meta-modules.

9.2.4 Reachable Space of Meta-Modules

Any meta-module that emerges will be able to move in some well defined space on the structure of modules. From its starting state (position and orientation) it will be able to perform a finite number of meta-actions each of which may be legal or illegal (e.g. would result in collision between modules). A legal meta-action performed by the meta-module will change the physical state of the meta-module. If we assume the structure to be static, this will allow us to build a graph for each meta-module, having the meta-module states as vertices and meta-actions as edges. This graph will define the reachable space of the meta-module:

- The reachable-space of a meta-module is a graph, where vertices are legal states (position and orientation) of the meta-module and edges are legal meta-actions which brings the meta-module from one legal state to another.

In Section 9.3 we use the reachable-space of meta-modules to measure some characteristics of the different meta-module types. In Section 10.2 a
local subset of a meta-module’s reachable-space is used to control it. Some example reachable-spaces of different meta-module types are visualized in Figure 9.3.

9.3 Selecting a Meta-Module for ATRON

To find a meta-module type able to handle the hard motion constraints of the ATRON module we compare six different meta-module types in this section. The meta-module should be small, highly moveable, effective and efficient.

9.3.1 Small and Non Lattice-based Meta-Modules

None of the six meta-module types we investigate in this section are composed of more than 3 modules. It is possible to build larger meta-modules from, e.g., 12 modules, which have very good motion capabilities. Such meta-modules are positioned and move within a lattice of meta-modules. In this work, we do not consider lattice-based meta-modules that are so large for a number of practical reasons:

- Meta-modules that sit in a lattice decrease the granularity of the system.
- The increase in cost and complexity of a single meta-module (e.g. 12-DOF) can hardly be justified by the improved motion capabilities.
- It would be impossible to do physical experiments with more than a few meta-modules using the existing 100 prototypes of the ATRON modules.

None of the above is true for the meta-modules explored in this work. In particular the meta-modules do not decrease granularity (except meta-module type 6), because they emerge, move and stop. When stopped, the modules which were part of a meta-module may at a later time become part of a different meta-module which is not necessarily composed of the same modules, as illustrated in Figure 9.1.

9.3.2 Different Meta-Modules Types

We compare six different meta-modules types, differentiated by their morphology and meta-actions. The meta-modules are referred to as meta-module type 1 – 6. The number of modules in the meta-module types and the number of meta-actions are summarized in Table 9.1. The three different meta-module types’ morphologies, shown in Figure 9.2, are selected so that they can often emerge from unstructured groups of modules. The meta-actions for type 1, 2, 3 and 4 are designed by taking direct inspiration from
9.3. Selecting a Meta-Module for ATRON

Figure 9.1: Difference between lattice-based meta-module types and meta-module types that emerge. (a) A two-module meta-module could sit in a permanent 2D lattice as indicated. The meta-modules would move from one lattice position to another. This decrease the system’s granularity. (b) In this work, any combination of two connected modules may emerge as a meta-module. The meta-modules do not sit in any particular lattice and modules are free to be part of different meta-modules at different times, therefore granularity does not decrease.

<table>
<thead>
<tr>
<th>Type of Meta-Module</th>
<th>Modules in Meta-Module</th>
<th>Number of Meta-Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 1</td>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>Type 2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Type 3</td>
<td>2</td>
<td>32</td>
</tr>
<tr>
<td>Type 4</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>Type 5</td>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>Type 6</td>
<td>3</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 9.1: Basic properties of different meta-module types.

the morphology of the meta-modules. Meta-module types 5 and 6 expand type 4. The expansions come from experience gained by investigations of type 4. The morphology and meta-actions of the six different meta-module types are explained below. Examples of reachable spaces for meta-module type 1 – 6 is shown in Figure 9.3(a)-(e). Investigations of such graphs indicate that meta-module types 3 – 6 are much more moveable than type 1 or 2.

**Meta-module type 1** is composed of a single module, see Figure 9.2(a). It has the ability to perform 16 different meta-actions that all follow the same blueprint. First, the meta-module disconnects all neighbor modules except one. Second, the meta-module remote controls the connected neighbor module to disconnect all other modules on that hemisphere and rotate
left or right, thus moving the meta-module. Later, when the meta-module has moved away, the neighbor module will reverse the rotation and disconnections to return to its initial state.

**Meta-module type 2** is composed of two modules which stay connected throughout the lifetime of the meta-module, see Figure 9.2(b). It is able to perform 4 meta-actions. First, the meta-module connects to all neighbor modules on one of the two hemispheres, which are not used to keep the two modules in the meta-module connected. Second, all other neighbor modules are disconnected. Finally, the connected module in the meta-module rotates and thereby moves the other module.

**Meta-module type 3** expands meta-module type 2 with 28 extra meta-actions. These meta-actions all follow the same blueprint. First, the meta-module disconnects all except one neighbor module (it has up to 14 neighbors). Second, the meta-module remote controls the connected neighbor module to disconnect all other modules on that hemisphere and rotate left or right to move the meta-module. The neighbor module will later return to its initial state.

**Meta-module type 4** is composed of three modules which are connected so that a centre (body) module is connected to two modules (legs), one on each hemisphere, see Figure 9.2(c). The meta-module is able to perform 8 different meta-actions. First, the meta-module connects to a neighbor module using a hemisphere on a leg-module not connected to the body-module. Second, all other neighbor modules are disconnected. Finally, the connected leg-module or the body-module performs a rotation either left or right. Such meta-actions allow the meta-module to move as a two-legged walker on a flat surface of ATRON modules.

**Meta-module type 5** expands meta-module type 4 with 4 extra meta-
Figure 9.3: Reachable Space of ATRON Meta-Modules. Examples of the six types of meta-module are shown in (a)-(e). (e) Represents both meta-module type 5 and 6 since they have identical reachable-spaces. Small black spheres are vertices and lines are edges in the graph of the reachable-space. For simplicity in this illustration, only the position of one of the modules comprising the meta-module is used to visualize the states. For type 2 and 3 the top module is used and for type 5 and 6 the middle body module is used.
actions. For simplicity, we will not explain the meta-action’s blueprint but only their effect. Two meta-actions allow the meta-module to move one leg-module so that it connects to the other leg-module, which then becomes a new body-module. The last two meta-actions either rotate the meta-module 90 or -90 degrees, changing its orientation. The four extra meta-actions require the help of a neighbor module. This meta-module type is further explained in Section 9.4, see also Figure 9.5.

Meta-module type 6 expands meta-module type 5 but not with extra meta-actions. Instead, we put constraints on the emerging and stopping orientation of a meta-module. Meta-modules may only start or stop in precisely the orientation shown in Figure 9.4. The motive is to help the emergence of flat structures of modules, on which it is particularly easy to move on for this meta-module type.

9.3.3 Characteristics of Meta-Module Types

Experimental Setup

We measure the characteristics of the ATRON meta-modules in simulation as they perform a shape-changing task. The task is to shape-change one randomly generated structure of modules to another randomly generated goal structure. Meta-modules are centrally controlled and only one meta-module is moved at each iteration.

In each iteration we calculate the reachable space of each of the possible meta-modules in the structure. From the reachable-spaces we find for each meta-module the state which will minimize the distance between the current and goal structure. The corresponding difference in distance we call a meta-module’s potential to decrease distance (PDD). Then the meta-module with the highest PDD is selected and it emerges, moves to and stops at the state

Figure 9.4: A meta-module of type 6 must never emerge or stop if it does not have precisely this orientation.
that realizes its PDD (in a single iteration). Using this control strategy the
distance between the starting structure and the goal structure will gradu-
ally decrease or stabilize, but it will never increase. Note that this control
strategy is not how we intend to control the meta-modules. The purpose is
only to have a common test scenario under which the different meta-module
types may be evaluated.

Characteristics of the different meta-module types are measured as the
average of 10 different random shape-changing tasks. In each experiment a
total of 90 modules are used in the initial and goal structure. Each exper-
iment runs for 20 iterations, which in most cases is plenty for the result to
stabilize.

Random structures of modules are generated by using the following algo-
rithm: Starting with a single seed module, the algorithm repeatedly insert a
module to the structure, at a randomly selected unoccupied connector on a
module, which is part of the structure. This algorithm produces initial and
goal structures, which on average have an initial overlap of 45%.

Distance between two structures (S1 and S2) is measured as the sum of
distances between the modules in S1 to S2. The distance between a module
and a structure is measured as the Euclidean distances from the module to
the nearest module in the structure.

**Task Related Characteristics**

Task-related characteristics measure how well the meta-module performs the
shape-changing task. We measure the *efficiency* as the effect achieved (de-
crease in distance between structures) compared to the number of rotations
performed by modules during the task. *Effectiveness* is measured as the
relative decrease in distance between the current and goal structure after 20
iterations.

\[
Efficiency = \frac{D_{\text{start}} - D_{\text{end}}}{\#\text{Rotations}} \tag{9.1}
\]

\[
Effectiveness = \frac{D_{\text{start}} - D_{\text{end}}}{D_{\text{start}}} \tag{9.2}
\]

**Meta-Module Related Characteristics**

Meta-module related characteristics measure the individual and cooperative
capabilities of meta-modules of a particular type. To increase parallelism
the system should contain a high proportion of moveable modules, which
can move as part of a meta-module, that is, modules that are not locked
in place by constraints on their movement from their morphology and/or
through blocking from other modules. We define this characteristic as *system
moveability*. It is measured as the ratio between the number of moveable
modules and modules in total in the structure. Similarly, a meta-module
should be able to move freely on the surface of other modules. We measure *meta-module moveability* as the ratio between the number of connectable modules and the total number of modules in the structure. Connectable modules are modules that a meta-module, within its reachable space, is able to connect to.

\[
\text{SystemMoveability} = \frac{\#\text{Moveable Modules}}{\#\text{Modules}} \quad (9.3)
\]

\[
\text{MetaModuleMoveability} = \frac{\#\text{Connectable Modules}}{\#\text{Modules}} \quad (9.4)
\]

When a meta-module moves, side effects may have a beneficial or undesired effect on the other meta-modules. We measure side effects in terms of change in the meta-modules’ potential to decrease distance (PDD). While performing a shape-change task side effects may occur at each iteration, from \(i\) to \(i + 1\). In terms of distance, \(D\), and PDD this can be expressed as difference equations:

\[
D_{i+1} = D_i - \text{PDD}_{i\rightarrow i+1}^\text{max}
\]

\[
\sum_{\text{mms}} \text{PDD}_{i+1} = \sum_{\text{mms}} (\text{PDD}_i) - \text{PDD}_{i\rightarrow i+1}^\text{max} + \text{SideEffects}_{i\rightarrow i+1}
\]

In each iteration the meta-module with a maximum PDD (\(\text{PDD}^\text{max}\)) moves to the corresponding state, this decreases the distance between the current and goal structure. However, as a side effect, the PDD of the other meta-modules (\(\text{mms}\)) in the system may also change. The sum of the side effects from iteration \(i = 0\) to \(i = n\) may be calculated as:

\[
\sum_{i=0}^{i=n} \text{SideEffects}_{i\rightarrow i+1} = (D_0 - D_n) - \sum_{\text{mms}} (\text{PDD}_0 - \text{PDD}_n)
\]

In general, the starting distance may be much larger than the starting sum of PDD, which means that the side effects should be positive. For the shape-change to be completed at iteration \(i = n\), the sum of side effects should be:

\[
\sum_{i=0}^{i=n} \text{SideEffects}_{i\rightarrow i+1} = D_0 - \sum_{\text{mms}} \text{PDD}_0
\]

From these considerations, we define the *side effect balance* which is positive if the movement of meta-modules generates beneficial side effects:
Table 9.2: Efficiency of meta-module types.

<table>
<thead>
<tr>
<th>Type of Meta-Module</th>
<th>Mean Efficiency</th>
<th>95% Confidence Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 1</td>
<td>0.89</td>
<td>[0.76, 1.01]</td>
</tr>
<tr>
<td>Type 2</td>
<td>1.31</td>
<td>[1.17, 1.43]</td>
</tr>
<tr>
<td>Type 3</td>
<td>1.18</td>
<td>[1.06, 1.31]</td>
</tr>
<tr>
<td>Type 4</td>
<td>0.29</td>
<td>[0.26, 0.32]</td>
</tr>
<tr>
<td>Type 5</td>
<td>0.34</td>
<td>[0.28, 0.41]</td>
</tr>
<tr>
<td>Type 6</td>
<td>0.36</td>
<td>[0.28, 0.44]</td>
</tr>
</tbody>
</table>

Table 9.3: Effectiveness of meta-module types.

<table>
<thead>
<tr>
<th>Type of Meta-Module</th>
<th>Mean Effectiveness</th>
<th>95% Confidence Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 1</td>
<td>0.20</td>
<td>[0.17, 0.24]</td>
</tr>
<tr>
<td>Type 2</td>
<td>0.21</td>
<td>[0.17, 0.24]</td>
</tr>
<tr>
<td>Type 3</td>
<td>0.62</td>
<td>[0.58, 0.66]</td>
</tr>
<tr>
<td>Type 4</td>
<td>0.66</td>
<td>[0.58, 0.73]</td>
</tr>
<tr>
<td>Type 5</td>
<td>0.66</td>
<td>[0.59, 0.72]</td>
</tr>
<tr>
<td>Type 6</td>
<td>0.51</td>
<td>[0.46, 0.56]</td>
</tr>
</tbody>
</table>

\[
\text{SideEffectBalance}_i = \frac{D_0 - D_i}{\sum_{mms} (PDD_0 - PDD_i)} - 1 \quad (9.5)
\]

9.3.4 Results and Conclusion

The task-related characteristics of the different meta-module types are shown in Table 9.2 and 9.3. In terms of efficiency, the one- and two-module meta-module (type 1, 2 and 3) performs much better than the three module meta-modules (type 4, 5 and 6). Meta-module type 3, 4 and 5 performs similar in terms of effectiveness (0.62, 0.66 and 0.66). Such effectiveness is acceptable in a range of applications, which does not require shape-change into precisely specified structures.

The meta-module related characteristics of the different meta-module types are shown in Table 9.4, 9.5 and 9.6. Meta-module type 6 stands out since it is highly moveable and has a positive side effect balance. However, it is not high enough to also have a high effectiveness. Because of its extra meta-actions meta-module type 5 is slightly more moveable than type 4. Overall types 3 and 5 seem to be the best choices of types investigated. From the effectiveness, we see that both have the ability to shape-change a
Table 9.4: System moveability of meta-module types.

<table>
<thead>
<tr>
<th>Type of Meta-Module</th>
<th>Mean System Moveability</th>
<th>95% Confidence Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 1</td>
<td>0.43</td>
<td>[0.40, 0.45]</td>
</tr>
<tr>
<td>Type 2</td>
<td>0.76</td>
<td>[0.72, 0.79]</td>
</tr>
<tr>
<td>Type 3</td>
<td>0.67</td>
<td>[0.62, 0.71]</td>
</tr>
<tr>
<td>Type 4</td>
<td>0.67</td>
<td>[0.62, 0.72]</td>
</tr>
<tr>
<td>Type 5</td>
<td>0.69</td>
<td>[0.63, 0.74]</td>
</tr>
<tr>
<td>Type 6</td>
<td>0.36</td>
<td>[0.34, 0.39]</td>
</tr>
</tbody>
</table>

Table 9.5: Meta-module moveability of meta-module types.

<table>
<thead>
<tr>
<th>Type of Meta-Module</th>
<th>Mean Meta-Module Moveability</th>
<th>95% Confidence Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 1</td>
<td>0.096</td>
<td>[0.09, 0.10]</td>
</tr>
<tr>
<td>Type 2</td>
<td>0.088</td>
<td>[0.087, 0.090]</td>
</tr>
<tr>
<td>Type 3</td>
<td>0.21</td>
<td>[0.19, 0.22]</td>
</tr>
<tr>
<td>Type 4</td>
<td>0.23</td>
<td>[0.21, 0.25]</td>
</tr>
<tr>
<td>Type 5</td>
<td>0.28</td>
<td>[0.27, 0.29]</td>
</tr>
<tr>
<td>Type 6</td>
<td>0.79</td>
<td>[0.76, 0.81]</td>
</tr>
</tbody>
</table>

Table 9.6: Side effect balance of meta-module types.

<table>
<thead>
<tr>
<th>Type of Meta-Module</th>
<th>Mean Side Effect Balance</th>
<th>95% Confidence Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 1</td>
<td>-0.10</td>
<td>[-0.24, 0.052]</td>
</tr>
<tr>
<td>Type 2</td>
<td>-0.28</td>
<td>[-0.32, -0.23]</td>
</tr>
<tr>
<td>Type 3</td>
<td>-0.49</td>
<td>[-0.55, -0.42]</td>
</tr>
<tr>
<td>Type 4</td>
<td>-0.66</td>
<td>[-0.69, -0.63]</td>
</tr>
<tr>
<td>Type 5</td>
<td>-0.71</td>
<td>[-0.74, -0.67]</td>
</tr>
<tr>
<td>Type 6</td>
<td>0.73</td>
<td>[ 0.33, 1.33]</td>
</tr>
</tbody>
</table>
9.4. The Selected ATRON Meta-Module Type

In the previous section, we investigated six different meta-module types and found that meta-module type 5, were the best type investigated. Here, we will explain this meta-module in somewhat greater detail, since we will use it for experiments in Chapter 10 and 11.

9.4.1 Morphology

The meta-module type 5 is composed of three ATRON modules: one centre module (body) is connected to two other modules (legs), one on each hemisphere, see Figure 9.5(a).

Figure 9.5: Illustrations of the morphology of the ATRON meta-module type 5 and its meta-actions. The dark modules comprise the meta-module. The *-marked modules in (b) and (c) are required to participate in the corresponding meta-actions.
9.4.2 Meta-Actions

To move meta-modules perform meta-actions, which consist of a sequence of connections, disconnections and ±90 degree rotations. The meta-module is able to perform twelve different meta-actions, which fall into three different blueprints:

Blueprint 1 (8 meta-actions): Meta-actions following this blueprint allow the meta-module to move as a two legged walker on a flat surface of modules. First, the meta-module connects to a structure-module using a hemisphere of a leg that is not connected to the body. Second, the meta-module is disconnected from all other modules. Third, either the connected leg-module or the body-module makes a ±90 degree rotation, as illustrated in Figure 9.5(e) and 9.5(d).

Blueprint 2 (2 meta-actions): The *-marked module in Figure 9.5(b) is required to help the meta-module when performing this meta-action. The four modules perform a sequence of connections, disconnections and rotations, which makes it turn around a “corner” as illustrated in Figure 9.5(b). A different meta-action following the same blueprint allows it to turn around a corner in the opposite direction.

Blueprint 3 (2 meta-actions): The *-marked module in Figure 9.5(c) is required to rotate one leg-module towards the other leg-module, which then becomes the new body-module of the meta-module. Similarly, a meta-action that rotates the other leg follows this blueprint. The effect of these meta-actions is to shift the orientation of the meta-module as illustrated in Figure 9.5(c).

The combination of morphology and meta-actions provides the meta-module with a high ability to move on the surface of other modules. Movement is achieved by performing a sequence of meta-actions.

9.4.3 Control Strategy

Modules are controlled with a strategy that makes them emerge meta-modules, causes the meta-modules to move and to stop again. Below we summarize this control strategy which will be the basis of experiments in Chapter 10 and 11. An overview of this control strategy is given in Figure 9.6, note that it has been fully implemented in simulation and partly transferred to the physical platform.

Passive or active Modules are either passive or part of a meta-module. Passive modules can respond to simple request from meta-modules such as connect-to-me, but can also decide to form and emerge a meta-module from the structure of modules. A meta-module can only emerge if three passive modules are connected in a legal meta-module configuration (no constraint on orientation). A meta-module is then able to move, but may also decide
Figure 9.6: Each module is controlled using the strategy illustrated above. Modules can be passive or active as part of a meta-module. A meta-module selects an action based on the local configuration of modules and guided by attraction points. It then performs the action, which can either fail or succeed.
to stop its movement. If the meta-module stops its modules become passive once again and the process can be repeated later. In Chapter 10 we will evolve two artificial neural networks to take the decisions to emerge or stop a meta-module.

**Action selection**  At any given time, many meta-modules will be moving in the system. To guide the flow of meta-modules we use attraction points, which define the shape of the desired global configuration. Meta-modules move towards attraction-points by performing a sequence of meta-actions. A module will inhibit an attraction-point if placed at the same location. Inhibited attraction-points are ignored by meta-modules. Meta-modules perform some local planning to select which meta-action to perform: First, the meta-module constructs a map of the local configuration of modules (6 hops). Second, the meta-module calculates a local subset of its reachable-space (see Figure 9.3). The reachable-space is a graph where vertices are legal states (position and orientation) of the meta-module and edges are legal meta-actions, which brings the meta-module from one legal state to another. Third, the meta-module calculates (using an A* algorithm[68] on the reachable-space) a shortest path sequence of meta-actions towards a goal-state. The goal-state is selected by a artificial neural network (will be evolved in Chapter 10) based on characteristics such as proximity to attraction points. Finally, the meta-module will perform the first meta-action from the found sequence, and the action selection process can be repeated.

**Action execution**  A meta-module performs a sequence of meta-actions to move. A meta-action is composed of a sequence of basic module actions (rotation, connection and disconnection), which are performed by the modules part of, or neighbor to, the meta-module. Meta-modules can perform four different types of meta-actions (see Figure 9.5). Each type represents 2 or 4 different meta-actions, so in total a meta-module can perform 12 different meta-actions. However, in a given situation only a subset of these 12 meta-actions will be legal. Robustness is increased by handling meta-actions that fail. Usually this means that a rotation results in a collision, or that a failed module is connected to and therefore locks the module. Collisions are detected by the rotating modules using its encoders. The roll-back strategy is to reverse the rotation, mark the state (in the reachable space) as unreachable and select another action to perform (recalculate shortest path).
9.5 Discussion, Conclusion and Future Work

9.5.1 Applied Design Principles

In this chapter, we considered the design of meta-modules to control the self-reconfiguration in ATRON. Modules are controlled in a distributed fashion where each module autonomously controls itself (follows Principle 8). To self-reconfigure, modules rely on the willingness of other modules to form meta-modules (follows Principle 9). Although, meta-modules are guided by global attraction points the behavior of a meta-module is highly dependent on its local context (follows Principle 10). Every module runs identical programs, which allows a module to be dynamically replaced (follow Principle 11 and 12). In Chapter 10 and 11 we will consider how the application of these design principles affects the robot.

9.5.2 Conclusion

This chapter considered the challenge of motion constraints in self-reconfiguration control. Every lattice-based system is subject to similar types of motion constraints, and therefore a self-reconfiguration strategy must take measures to unconstraint them. For the ATRON system, six different combinations of meta-module morphology and meta-actions were investigated. The morphology varied from 1 to 3 modules and the number of meta-actions from 4 to 32. Different characteristics of the meta-module types are measured as they perform the task of shape-changing a structure of ATRON modules. Based on the measured characteristics a two-module meta-module (type 3) and a three-module meta-module (type 5) are found to be the best investigated way of shape-changing the ATRON system. However, since meta-module type 5 is more autonomous it was selected for further investigations in the following chapters.

9.5.3 Future Work

Here, we investigated the characteristics of different meta-module types for the ATRON system. However, the approach taken could also be adopted to investigate the characteristics of meta-modules for other self-reconfigurable robots. Moreover, the approach could also guide the design process of new types of lattice-based modules, since it is important to know if it is possible to control the modules before they are built. Further, it is straightforward to have more than one meta-module type active at the same time in a system. This might be an advantage in different situations. The difference in the characteristics of the meta-module types may result in the emergence of implicit cooperation and coordination between meta-modules of different types. Indeed, preliminary experiments along these lines indicate that this
is true. Because the task related characteristics increase when combining meta-module types 3 and 5.
Chapter 10

Automatic Control Generation

In this chapter, we consider automatic development of module controllers for self-reconfiguration, which can increase the abstraction of controller development and shorten development time. In addition, by utilizing such strategies we can more efficiently address other design issues, for example, a potential module design can quickly be evaluated since we automatically can generate a controller for it. Specifically in this chapter, we evolve a distributed artificial neural network controller for the simulated ATRON modules. The controller is identical on every module and controls when a meta-module emerges, how it moves and when it stops. In simulation, we demonstrate how this control strategy allows the ATRON robot to shape-change, to support an unstable roof and to build a bridge across a gap. We conclude that the control strategy is able to shape-change the ATRON robot in the range from dozens to thousands of modules. In the following chapter, we will study the control strategy’s ability to tolerate module failures and self-repair.

10.1 Introduction

10.1.1 Design Scope

The self-reconfiguration control problem is characterized by being highly complex, e.g., it must unconstraint the different types of motion constraints, described in Chapter 9. Therefore, it is still beyond our reach to automate the entire control development from scratch. Instead, we will develop a strategy that automatically finds a solution to a well-defined sub-problem. In an adjustable part of the controller, we encode a range of potential solutions, which we then optimized in a trial-and-error fashion. Potential implementation of adjustable controller parts include: parameters, rules, state machines.
and artificial neural networks.

To obtain a more generic solution the automatically generated controller should be scalable and configuration independent: i) Since self-reconfiguration is a scalable problem and we do not want to limit ourselves to a specific number of modules. Therefore, we will optimize a homogeneous controller that is used for every module in the system. This allows the number of modules to seamlessly be scaled up or down. ii) Similarly, we do not want to limit the developed controller to a particular configuration, why no assumptions can be made about the starting configuration or the goal configuration during optimization.

10.1.2 Related Work

In most related work, the control strategy for self-reconfiguration is manually developed. However, some attempts have been made to automate the control development by artificial evolution. Østergaard et al. [126] evolved a distributed state-machine based controllers for small structures of 12 to 20 ATRON modules. The purpose was to produce cluster flow style locomotion for specific or co-evolved configurations. However, due to the complexity of the task only short, non-periodic movements were achieved. Similar, for the 2D metamorphic system genetic programming was used to generate controllers for movement of 8 modules to solve tasks such as moving through a narrow passage[3].

Due to the required number of evaluations of potential solutions, evolution is generally performed offline in a simulator. A good alternative is to use learning to automatically develop a controller online. For the task of self-reconfiguration this has been done using a reinforcement learning strategy for the sliding cube model[185]. One of the finding in this work was that incremental learning could improve the convergence, by starting to learn with small structures and then gradually adding module modules.

In this chapter, we evolve the weights of artificial neural networks that control ATRON meta-modules. Similar, strategies have previously been used on mobile robots for tasks such as obstacle avoidance[49]. Automatic control development has also been used to optimize fixed-topology style locomotion of modular robot, as we will review in Chapter 12.

10.2 Evolving Control for ATRON Meta-Modules

This section describes the automatic development of a controller for the ATRON meta-module type 5 (see Chapter 9). A flow diagram with an overview of the controller is shown Figure 9.6.
10.2. Evolving Control for ATRON Meta-Modules

10.2.1 Artificial Neural Network Controller

A module makes autonomous decisions concerning:

1. When the module should emerge as part of a meta-module?

2. Which meta-actions the meta-module should perform to move?

3. When the meta-module should stop?

Decisions are taken by three feedforward, 3-layer, sigmoid activation function, artificial neural networks (ANN), one for answering each of the questions. The controller calculates on-line a number of inputs to the ANN’s. The inputs are calculated based on positions of attraction-points and the state of the local surrounding, such as positions and orientation of nearby modules and obstacles. In this section we first explain how the meta-modules calculate a subset of their reachable-space from which the inputs to the ANN’s are calculated. Second, we explain how the ANN’s are used to control the meta-modules.

Meta-modules Calculates a Subset of their Reachable-Space

Meta-modules calculate online a small subset of their reachable-space, to produce inputs for the ANN’s: 1) The meta-module builds a map of the local surroundings using neighbor-to-neighbor communication. The communication range is limited to six neighbors away. The map contain information about which positions, in the ATRON lattice, are known (by modules) to contain passive modules, modules part of a meta-module, obstacles and which positions are empty. 2) The reachable-space subset is calculated using the map in a breadth-first manner. Initially, the subset only contains one state - the actual state of the meta-module. Iteratively the reachable-space subset is expanded by repeatedly applying rules to the states in it. Rules correspond to meta-actions, so in total there are twelve rules one for each meta-action. A rule has a pre-condition which states which positions relative to the meta-module should be empty, which should contain passive modules and constraints on the orientation of the meta-module (for some meta-actions). A rule also has a post-condition, which give the new state of the meta-module relative to the old one. To avoid computational explosion, states already seen are not recalculated and a fixed number (12) of iterations on the graph are done. This keeps the size of the graph down. Based on 5000 test samples there are on average 83 and a maximum of 687 vertices in the graph. The relative small size of the graph enables it to be calculated at runtime on the ATRON modules.
Emergence of Meta-module

With a low probability, a passive module will attempt to emerge a meta-module. It randomly selects two passive neighbor modules, one on each hemisphere. It then calculates the reachable-space subset of that meta-module. From this, it calculates the following inputs to an ANN, which has 3 neurons in input-layer, 3 neurons in hidden-layer and 1 neuron in output-layer:

- Distance to nearest attraction-point from current state of the meta-module.
- Distance to nearest other meta-module from current state of the meta-module.
- Biggest known possible reduction in the distance between the meta-module and its nearest attraction-point.

The distance is calculated as the sum of the Euclidian distances for the three modules in the meta-module. If the output value of the ANN is greater than 0.5 the meta-module will emerge and the ANN’s for movement and stopping will control the meta-module.

Movement of Meta-module

When a meta-module has emerged, it starts to move by selecting one of the twelve meta-actions. The selected meta-action is then performed and the process is repeated. An ANN with 4 neurons in input-layer, 4 neurons in hidden-layer and 1 neuron in output-layer is used to select which meta-action to perform next. Each state in the reachable-space subset is evaluated separately. We make extensive use of the shortest path sequence of meta-actions (SPSM) that brings the meta-module from its current state to the state being evaluated. The following inputs are given to the ANN for each state:

- Number of meta-actions in the SPSM.
- Number of common meta-actions between the current SPSM and the previous SPSM. The previous SPSM is the latest sequence of meta-actions from which the meta-module performed the first meta-action.
- Shortest distance to another meta-module, measured from a state along the SPSM.
- Reduction in distance between the meta-module and its nearest attraction-point, if it moves to the state being evaluated.
The state, which is being evaluated, is assigned a fitness value from the output of the ANN. The state in the reachable-space subset that has the highest fitness value is selected and the first meta-action in the corresponding SPSM is then performed by the meta-module. Since the structure of modules is dynamic and information in the map may be incomplete, it happens that the performance of a meta-action fails. The meta-module then recovers as good as possible, e.g. if a rotation fails because of collision the rotation will be inverted and the meta-action cancelled and ignored until another meta-action succeeds.

**Stopping of Meta-module**

Each time the meta-module has performed a meta-action it decides if it is time to stop or perform another meta-action. An ANN with 5 neurons in input-layer, 3 neurons in hidden-layer and 1 neuron in output-layer makes this decision based on the following inputs:

- Biggest known possible reduction in the distance between the meta-module and its nearest attraction-point.
- Distance to nearest attraction-point from current state of the meta-module.
- Number of possible connections between the meta-module and its passive neighbor modules.
- Reduction in distance to nearest attraction-point over the last five meta-actions.
- Number of cancelled meta-actions (e.g. because of collision) in the lifetime of the meta-module.

If the output of the ANN is greater than 0.5 the meta-module will stop moving and connect to all passive neighbor modules.

### 10.2.2 Evolution of Artificial Neural Network Controller

Evolution is chosen to optimize the value of the ANN’s weights, since there is no obvious way of training the network and since evolution may be good to exploit implicit cooperation between meta-modules. The actions of one meta-module may affect other meta-modules in ways which are difficult to analyze and harder to exploit.

**Encoding of Artificial Neural Networks**

The topologies of the networks are fixed, only the weights are optimized by means of evolution. The genome of each individual is the 50 weights,
Figure 10.1: The graphs show the average and highest fitness for each generation, when evolving the weights of the artificial neural network controller for the ATRON modules.

which are directly encoded as floating point values. Initially the weights have random values between -0.5 and 0.5.

Genetic Algorithm

A simple genetic algorithm is used. Each generation consists of 100 individuals. The individuals in each generation are evaluated and their fitness calculated. The 3 fittest individuals are used as elites and directly copied to the new generation. A child has two parents randomly selected from the group of the 25 fittest individuals. The child is produced by using a randomized 12-point crossover and a mutation rate of 5%. When mutating a gene it is with equal likelihood replaced with a new random value or a small random value added to or subtracted from the gene.

Fitness Evaluation

To evaluate the individuals they perform two tasks. A task is to shape-change a structure of 50 modules into another random structure specified by 50 inhibiting attraction-points, placed within the ATRON lattice. The random structures are generated as described in Section 9.3.3. The individuals in a generation are all evaluated on the same random tasks, but from generation to generation, new random tasks are generated. A task is terminated if there within 300 simulator time-steps (approximately 45 sec. on the physical modules) has been no decrease in Euclidian distance between the modules and the attraction-points. The fitness of an individual is calculated as the average fitness from each of the two tasks. Fitness of a task is simply the effectiveness, as defined in Equation 9.2, which is calculated
10.2. Evolving Control for ATRON Meta-Modules

Figure 10.2: In order to support an unstable roof the structure of 500 ATRON modules shape-change, stretching upwards to achieve the functionality of a pillar. The process is guided by attractions-points that are shown as small spheres.

as the relative change in distance between the structure of modules and the attraction-points: \( \text{fitness} = (D_{\text{start}} - D_{\text{end}})/D_{\text{start}} \). A number of evolutionary experiments were performed before reaching the details described above. The final controller, used for experiments in this work, was obtained from a evolutionary run for which the fitness graph is shown in Figure 10.1. The noise in the fitness evaluation indicates that some tasks are harder to solve than others are.

10.2.3 Experiments

In this section, we first demonstrate how shape-changing behaviors may be achieved using attraction-points and the evolved artificial neural network controller. Second, we present experiments to validate that the evolved controller is scalable.

Experiment: Support Unstable Roof

In earthquake or cave environments, it might be desirable to have systems that can support an unstable roof. In this experiment, a random structure of 500 modules initially lay on a floor. A roof is positioned at the height of 26 modules above the floor. The target functionality is the same as that of a pillar. To achieve this functionality 117 inhibiting attraction-points are placed in a column shape. As the robot change-shape upwards the attraction-points of lower positions will be inhibited by the modules at their
Experiment: Bridge Gap

In a range of scenarios, it may be useful to have a system that can build a bridge across a gap. In this experiment, 1000 ATRON modules are initially placed in a random structure. A single non-inhibiting attraction-point is placed as shown in Figure 10.3. The shape of the ATRON robot stretches towards the attraction-point, effectively building a bridge across the gap. The used ATRON simulator does not support physics. During this experiment, long thin arms of ATRON modules are build, which most likely would break off in a real-world gravity environment. This problem could be reduced by adding more attraction-points to guide the shape-change. Note that the structure does not move it only stretch. This is because groups of modules, due to motion constraints, are confined in configurations from which meta-modules cannot emerge. To solve this limitation a less constrained meta-module must be found.

Experiment: Scalability

The ANN controller was evolved using tasks containing 50 modules. To investigate how the controller scaled to shape-changing structures of more
modules we measured the controller’s performance as it performed a series of tasks. Tasks were of the same random type as used when evolving the controller, but the number of attraction-points and modules were varied from 50 to 1200 in steps of 50 modules. The stop criterion was no improvement in 300 time-steps.

The graphs in Figure 10.4 indicate that the relative change in distance, when performing a task, does not decline with the number of modules in the structure. Therefore, the experiment indicates that the evolved controller scales up to at least 1200 modules. Further experiments with 1500, 2000 and 2500 modules indicate that the evolved control scales even to this number of modules. For example, with 2500 modules, the average relative decrease in distance is 0.345, which is slightly better than average (based on 10 trials).

For comparison the equivalent graph of an alternative controller is also shown in Figure 10.4. This alternative controller is hand-coded based on its reachable-space subset and does not use ANN’s. The alternative controller:

- Emerge, if it is able to reduce its distance to the nearest attraction-point.
- Move, along the shortest path of meta-actions, towards the state that minimize its distance to an attraction-point.
Stop, if it no longer is able to reduce its distance to the nearest attraction-point.

The experiments indicate that the evolved controller performs significantly better than the alternative controller does.

Figure 10.5 shows how the time to self-reconfigure scales in relation to the number of modules in the structure. We can see that in the interval from 700 to 2500 the time to self-reconfigure (without any decrease in effectiveness) is almost constant. This is somewhat counter-intuitive but may be a result of the high degree of modules able to move as a meta-module at any given time (system moveability) and the ability of evolution to exploit it. We can, however, not expect this time to stay constant for much higher number of modules.

10.3 Discussion, Conclusion and Future Work

10.3.1 Applied Design Principles

As discussed in Chapter 9, the module’s behavior follows the design principles from Chapter 3, since it is distributed with autonomous modules, controlled based on their local context. The results of this chapter, demonstrated how the behavior of the robot emerge from the behavior of its modules (follows Principle 6). By using artificial evolution, a homogeneous controller was automatically designed for the modules (follows Principle 12).
Our approach forced the evolutionary process to exploit emergent phenomena and find a scalable solution. In the next chapter, we will demonstrate how this controller is also resilient to module faults and can be used to achieve self-repairing robots.

The use of external attraction points, allowed us to interact and guide the emergent process, so that the same controller could be applied for different tasks. Attraction points do not violate the design principles since the modules can maintain them collectively (by following Principle 9). However, the use of global attraction points is a weakness in the control strategy at the robot level, since attraction points are manually defined the emergent robot is not fully autonomous. Further, the attraction points are not based on the local context of the robot. Why it is not straightforward to let the robot (i.e. modules collectively) autonomously select where to place attraction points.

10.3.2 Conclusion

In this chapter, we studied the problem of automatic control development for the problem of self-reconfiguration. We addressed the problem with a distributed control strategy for the ATRON robot. A key ingredient in the control strategy is the modules’ continuous calculation of a subset of their reachable-space graph. Based on this reachable-space graph we evolved an artificial neural network controller for the modules. The controller controls the emergence, movement and stopping of meta-modules composed of three ATRON modules. Tasks for the robots are specified using attraction-points, which trigger meta-modules to emerge and move towards them. The combination of meta-modules and evolved artificial neural network controller are shown to be able to deal with the difficult motion constraints of the ATRON modules. The control strategy allows the robot to change its shape to meet some desired functionality. In simulation, we have verified that the control strategy scales to several thousands of modules in the robot.

10.3.3 Future Work

The next chapter will present experiments on a partial transference of the proposed controller to the physical platform. Future work would involve a complete transference. Further, the presented control strategy has several limitations which future work should address: i) Only relatively crude approximations of shapes can be constructed due to motion constraints on the meta-modules. ii) The use of meta-modules that crawls on the surface of other modules, not through the structure, limits the control strategy’s scalability (as we saw in Chapter 6). iii) The control strategy’s implementation is rather complex, since each module must be able to construct local maps and simulate the kinematics of a meta-module.

A potential future direction would be to for the meta-modules to learn
parameters of their neural networks online. We could exploit the fact that many meta-modules are moving in parallel, by letting them learn independent controllers which are affected by the other controllers active in the system, e.g., by utilizing a particle swarm optimization. This could presumably give the effect that larger systems would learn a controller faster than a small one, and that a solution could be found in a single long run of the system.
Chapter 11

Robustness to Faults and Errors

In this chapter, we study the possibility to use the redundancy of self-reconfigurable robot to realize fault tolerant and self-repairing robot. Concretely, we presents a series of experiments on fault tolerant self-reconfiguration of the ATRON robotic system using the meta-module based control strategy developed in Chapter 9 and 10. We perform experiments on three different types of failures: 1) Action failure: On the physical platform we demonstrate how roll-back of actions are used to achieve tolerance to collision with obstacles and other meta-modules. 2) Module failure: In simulation we show, for a 500 module robot, how different degrees of catastrophic module failure affect the robot’s ability to shape-change to support an insecure roof. 3) Robot failure: In simulation we demonstrate how robot faults such as a broken robot bone can be emergent self-repaired by exploiting the redundancy of self-reconfigurable modules. We conclude that emergent, distributed control, action roll-back, module redundancy, and self-reconfiguration can be used to achieve fault tolerant, self-repairing robots.

11.1 Introduction

11.1.1 Biological Motivation

Biological organisms are highly robust. Bone and skin will heal itself if broken, the brain and body will adapt to the loss of an eye or an arm - staying alive, functioning as well as possible. Robots, on the contrary, are in general not very fault tolerant. The loss of a sensor, an actuator or a piece of mechanics will in most cases leave the robot completely helpless and unable to perform its function. One reason for the successfulness of biological organisms is the trillions of cells making up the body. A multi-cellular biological organism has no single-point-of-failure; this is ensured by
the redundancy of cells - which may divide, migrate, differentiate, or die to assemble or repair the organism they compose.

In several ways, self-reconfigurable robots have the same distributed nature as biological system, why it is a common made claim that such robots can become more robust than conventional robots.

11.1.2 Types of Errors

In this chapter, we consider three types of errors are especially applicable to self-reconfigurable robots:

Action Failure The action of a module can fail, e.g., a particular actuation sequence or a connect/disconnect. Clearly, a robot cannot assume a perfect action execution, but must be able to tolerate such failures.

Module Failure At any time a module may fail. Especially when scaling up the number of modules failures become inevitable. Therefore, the robot must be able to tolerate failures of its individual modules and preferably adapt to maintain its function.

Robot Failure The morphological integrity of a robot may fail due to external events, similar to a broken bone on biological organisms. Such events affect the function of many modules. In such cases, the robot must self-repair if possible.

11.1.3 Related Work

Related work on self-repair of self-reconfigurable robots generally involves the detection of module failure, decisions on how to remove a defect module, and how to replace it with a spare module[48, 214, 179]. This is a top-down, essentially centralized, approach to the problem. Alternatively, as in this work, self-repair can emerge as a side effect of the self-reconfiguration, without having a specialized self-repairing part of the controller[168].

11.2 Fault-Tolerance and Emergent Self-Repair

11.2.1 Experimental Setup

A partial implementation of the meta-module-based controller has been transferred to the physical ATRON platform. The implementation builds on abilities of the modules, such as rotate, connect and disconnect. It corresponds to the lower level control of individual meta-modules, see Chapter 9, specifically the flow diagram in Figure 9.6. Each physical meta-module is able to:
• On-line find a shortest-path of meta-actions from a reachable-space, which is known at compile-time.

• Perform meta-action types in Figure 9.5(d) and 9.5(e).

• Detect and perform roll-back of failed actions.

When performing these experiments, the module-to-module communication was unstable due to crosstalk from reflections of IR-communication. This were the motivation for the extended communication protocol presented in Chapter 5. The communication limitation is the main reason for only demonstrating a partial transference, and the small number of experiments performed on the physical modules.

In the physical experiments, 24 passive modules are initially assembled as a horizontal sheet, on which the meta-modules can easily move. Meta-modules (with white shells), move on top of these modules. Meta-modules are placed at a predefined position (usually the corner) on the sheet of modules. By sending a special message to the meta-module, using another module as a remote control, the meta-module is started.

Simulation experiments are, as in previous chapters, performed in a transition-based simulation (no physics except collisions), which contains a full implementation of the meta-module-based control strategy described above.

11.2.2 Experiments

Basic Meta-Module Behavior

In this experiment (see Figure 11.1), the meta-module selects a random state, then moves (following shortest path) to that state, and then selects a new random state and so forth. After a meta-action is performed, the meta-module always recalculates the shortest path. This allows the meta-module, in principle, to adapt to possible changes in the environment or configuration of modules. The details of four experiments (all with the same initial setup) are shown in Table 11.1. On average, a meta-action takes 6.3 seconds to perform, including the calculation of shortest path and the coordination between modules comprising the meta-module.

Tolerance of Action Failure - Unknown Obstacle

In this experiment, we demonstrate how a meta-module handles collisions with obstacles in its environment. By using its encoder, a meta-module detects an obstacle when colliding with it. If a collision is detected while performing a meta-action, the meta-module performs a roll-back rotation to the lattice-position it came from. The corresponding state in the meta-module’s reachable-space is then assumed to be filled with an obstacle and is
Chapter 11. Robustness to Faults and Errors

Figure 11.1: The meta-module repeatedly calculates and then moves shortest path from one randomly selected state in its reachable-space to another.

<table>
<thead>
<tr>
<th>Exp.</th>
<th>#Meta-Actions</th>
<th>Exp. time (seconds)</th>
<th>Second pr. meta-action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>52</td>
<td>6.5</td>
</tr>
<tr>
<td>2</td>
<td>48</td>
<td>330</td>
<td>6.9</td>
</tr>
<tr>
<td>3</td>
<td>38</td>
<td>228</td>
<td>6.0</td>
</tr>
<tr>
<td>4</td>
<td>93</td>
<td>570</td>
<td>6.1</td>
</tr>
<tr>
<td>Total</td>
<td>187</td>
<td>1180</td>
<td>6.3</td>
</tr>
</tbody>
</table>

Table 11.1: Single meta-module following online-planned sequences of meta-actions.
11.2. Fault-Tolerance and Emergent Self-Repair

Hereafter ignored when doing shortest-path search. The meta-module then finds an alternative shortest-path and follows that until it perhaps again collides with an obstacle. By repeating this pattern the meta-module is able to find its way around unknown obstacles (Figure 11.2). Table 11.2 summarizes the results of three different experiments with unknown obstacles. The position of the attraction-point and obstacle are varied for each experiment. The number of meta-actions performed by the meta-modules using this trial-and-error approach ranges from being 29% to 47% higher than what could optimally be achieved using global knowledge.

Tolerance of Action Failure - Colliding Meta-Modules

There is a tradeoff between the amounts of coordination and the rate of collisions between moving meta-modules. In this experiment, there is no coordination between the meta-modules, so they will collide with each other from time to time. To handle this we apply the same roll-back rotation strategy as is used to handle collision with unknown obstacles.

In the trial, shown on Figure 11.3, three independent meta-modules...
Table 11.2: Meta-module moving from a starting to a goal position around an unknown obstacle.

<table>
<thead>
<tr>
<th>Exp.</th>
<th>#Collisions</th>
<th>#Meta-Actions performed (optimal)</th>
<th>Percent longer than optimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>32 (20)</td>
<td>38%</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>24 (17)</td>
<td>29%</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>38 (20)</td>
<td>47%</td>
</tr>
<tr>
<td>Total</td>
<td>19</td>
<td>94 (57)</td>
<td>39%</td>
</tr>
</tbody>
</table>

Table 11.3: Two and three meta-modules coexisting - no coordination.

<table>
<thead>
<tr>
<th>#Meta-Modules</th>
<th>#Meta-Actions (#Collisions)</th>
<th>Collisions pr. meta-action</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>56 (7)</td>
<td>0.125</td>
</tr>
<tr>
<td>3</td>
<td>52 (10)</td>
<td>0.19</td>
</tr>
</tbody>
</table>

moves following shortest path of meta-actions, from one randomly selected state to another. The meta-modules collide but roll-back resolves this conflict and the meta-modules can find their way around each other. Table 11.3 summarizes two experiments with two and three meta-modules respectively moving on a surface of modules. During the experiment 12.5% and 19%, for two and three meta-modules respectively, of the performed meta-actions are rolled back due to collisions.

**Tolerance of Module Failure**

To investigate the meta-module-based controller’s ability to tolerate catastrophic module failures, a series of experiments have been performed in simulation. The task is to support an insecure roof. The initial structure consists of 500 modules and 117 attraction points. If no modules fail the robot will reach the roof and thereby support it (see Figure 11.4). The experiment is repeated with failure rates of 5%, 10% and 15%. A failure rate of e.g. 10% means that initially and during the experiment 50 randomly selected modules out of the 500 ATRON modules are non-functional from the start. This means that the modules are unable to communicate, rotate, connect, or disconnect. The initial state of a connector is connected, so a failed module will generally lock other functional modules in place with its male connectors. The functional modules cannot use the failed module to move on since they have no way of detecting it as a module, in fact failed modules will be treated as obstacles. As can be seen from Figure 11.4, the robot is able to support the roof up to a failure rate of 10%, but the strength of the robot tends to become lower as the failure rate increase. Also, the
Figure 11.3: In this trial three meta-modules move on the same surface of modules. They do not communicate so they collide with one another, but the control system is able to tolerate this so that they can co-exist.
Figure 11.4: The meta-module based control is fairly tolerant to module failures. The initial (random) configuration of 500 ATRON modules is shown on the left. The robot shape-change guided by attraction-points shown as small dots. On the right, the result is shown for different degrees of module failures. The failed modules are black and are failed from the start of the simulation.
The performance of the robot declines when the module failure rate increases (Figure 11.5). We measure performance as $\text{effectiveness} = (D_{\text{start}} - D_{\text{end}})/D_{\text{start}}$, which is the relative decrease in sum of Euclidian distances between the modules and the attraction-points. This may not a particular good performance measurement for this particular task, it is however a good overall performance measurement for the ability to shape-change.

The observed degree of tolerance to module failures emerges from the redundancy of modules and the use of distributed control of meta-modules. This tolerance could be improved further by removing failed modules from the system (i.e. let them fall off). However, the current connector system would require up to four functional modules to be “sacrificed” per failed module.

**Tolerance of Robot Failure**

A future miniaturization of modules would open up for new possible applications, e.g. smart material that could self-repair. This experiment demonstrates the use of miniature ATRON modules in an emergent self-repair scenario. Initially, using a CAD model, 3426 ATRON modules are assembled in the form of a bone (Figure 11.6). A total of 1663 inhibiting attraction-points are placed at the same positions as modules which are connected to eight neighbors. This leaves the surface of the bone free of attraction-points. At
11.3 Discussion, Conclusion and Future Work

11.3.1 Discussion

We envision self-reconfigurable robots will ultimately consist of billions of micron-size modules. In such robots, at any given time, modules will fail and a considerable portion of the modules can be expected to be non-functional. In this scenario, distributed methods that handle failures locally and in a non-explicit fashion are desirable. Based on our experiments we observe that the impact of action, module, and robot failures can be reduced, as follows:

*Action Failures:* Control can be simplified, and robustness increase, by
using short action that can be rolled back locally. Because this control approach limits the need for collaboration between modules and assumptions about the environment.

Module Failures: The impact of module failures is limited by using meta-modules that emerge from unstructured groups of modules and move somewhat independently on other modules. However, we also observe that morphological adaptation (such as two-way disconnect) of the modules could increase the system’s robustness. A key factor is limiting the dependence between modules.

Robot Failures: The system can self-organize its modules to achieve some level of self-repair, without any part of the system ever having to be “aware” of any faults. This is due to the use of emergent, distributed control where modules react locally to the removal of modules. Key factors are redundancy and self-reconfiguration, which limits the robot’s dependence on its modules.

11.3.2 Applied Design Principles

This meta-module based control strategy follows the design principles from Chapter 3 related to module behavior design, since it is distributed, it utilizes identical controller, etc. Since, we have already discussed this in Chapter 9 and 10 it will not be repeated here. Based on the results of this chapter, we can add that the control strategy enables the robot not to be critically dependent on its modules (follows Principle 7). This enables it to adapt to failures at the action, module and robot level.

11.3.3 Conclusions and Future Work

This chapter has presented some experiments on emergent self-repair and fault-tolerant self-reconfiguration of the ATRON robot. We have in part transferred a meta-module-based control strategy to the physical modules and verified the basic characteristics of meta-modules and the use of failed action roll-back. This allows the meta-modules to co-exist with other meta-modules and find their way around unknown obstacles in their environment. Simulated experiments shows that the emergence of meta-modules from unstructured groups of modules helps tolerate up to 10% failed modules. We have also demonstrated how the redundancy of modules allows self-repair of a bone to emerge. Future work includes a complete transference from simulation to the physical world of the results obtained with meta-module-based control of the ATRON system.
Part V

Adaptive Collective Behavior
Chapter 12

Learning with any Morphology

Conventional robots are born with a flexible control system and a fixed body. That is, the behavior of the robot can be changed with little effort by reprogramming the robot. However, this is not the case with morphology. Conventional robots can therefore adapt their control to the task, but must do so under the constraints of their morphology. On the contrary, the morphology of modular robots is easy to change by reassembling the modules. Hence, the design process can be transformed, so that the control is kept unchanged, while only the morphology of the robot is changed. This approach put unique requirements on the autonomous control of modules, since it should be able to adapt to morphological changes.

This chapter explores minimal, model-less, configuration independent, life-long adaptation for fast learning of locomotion in modular robots. Since a modular robot is polymorphic, it is time consuming to design controllers manually for all of its different shapes. We observe that learning can automate the process of controller design. To realize this we study a simple distributed reinforcement learning strategy. ATRON modules with identical controllers are assembled into some robot configuration not known in advance. Then, based on a single global reward signal, each module independently learns to adjust its behavior in order to optimize the locomotion speed of the robot as a whole.

In simulation, we study both the learning strategy’s performance on different robot configurations, its ability to adapt to module faults and self-reconfiguration, as well as its scalability characteristics. On the physical platform, we perform learning experiments with ATRON robots learning to move as fast as possible. The following chapter will experiment with extensions to the learning strategy, which enables it to be used on other modular robots.
12.1 Introduction

12.1.1 Scenario

Consider a modular robot as an interactive toy that can be programmed by assembling the morphology of the robot. The user can change the behavior of the robot not by changing its program, but by changing its morphology. If two modules are connected, they will perform some behavior that is appropriate in the context of their morphology and their environment. As more modules are added, the behavior will change to match the changed morphology. Any robot assembled will perform some behavior, which emerges from its context (morphology/environment). Further, the robot will improve its behavior by learning online to improve its performance on the given task. This robot learns an appropriate behavior given the task, environment and morphology.

12.1.2 Design Challenges

The robot must have a controller that can gradually adapt to the environment and its morphology to optimize its behavior. In order to learn with many different morphologies, the learning system must be able to deal with a number of challenges. Below we summarize and discuss some of them.

Noise For the robot to learn we will provide it with some measurement of performance. However, this measurement can be highly influenced by uncontrollable parameters, which introduce noise. Such noisy performance measurements may hamper the learning.

Self-Damage While learning the robot may perform movements and actions that potentially can damage the robot itself. Similar, while learning the robot may fall over or become stuck, which would disable the robot from learning.

Repeatability For the learning to be successful a behavior should be repeatable, so that performing the same action under equivalent conditions would yield similar performance every time. This is not always the case since uncontrolled parameters may affect the result, for example, the dynamics of the robot, its initial pose or the characteristics of the environment.

Any Body In general the number of different robot configurations is exponential with the number of modules. Even with few modules, it is intractable to explore the whole design space of robot morphologies and controllers by hand. Since we are considering configuration independent learning, we cannot optimize the learning strategy to a specific robot morphology. This is different from most related work,
where the learning space is reduced, e.g., by exploiting symmetries in
the robot.

12.1.3 Related Work

Related work on configuration independent learning is sparse. However, a
number of papers have explored the more general problem of adaptation
in modular robots. Here, we consider related work on adaptation such as
evolution and online learning for tasks such as locomotion. Evolution and
learning are different and complementary in a number of ways. Evolution
adapts morphology and behavior over large timescales, over generations of
individuals, to ensure the survival of the species. Learning, on the other
hand, enable adaptation during the lifetime of an individual to ensure its
survival.

Evolution  In modular robots a classical approach to automate behavior
and morphology design is to co-evolve the robot configuration and control[159,
100, 105]. Although appealing, one challenge with this approach is to trans-
fer the evolved robots from simulation to physical hardware and once trans-
ferred the robot would no longer be able to adapt. To avoid the transference
problem, we utilize online learning instead of evolution.

An example of adaptation by evolution in modular robots was reported
by Kamimura et al., who evolved the coupling parameters of central pat-
ttern generators for straight line locomotion of M-TRAN self-reconfigurable
robots[77]. The used CPG model incorporated feedback from the actua-
tors, which would affect the limit cycle of the CPG and through coupling
with other CPGs would affect the overall gait. For example, it could adapt
the locomotion frequency to the mechanical limitations of the robot. This
global entrainment made the robot gait robust to external disturbances and
allowed it to adapt to changes in surface properties.

Learning  Most related work on robot learning utilizes some degree of
domain knowledge, typical about the robot morphology, when designing a
learning robot controller. In this work, we want to avoid such constraints
since our modular robot may be reconfigured or modules can be added or
replaced. Therefore, we do not know the robot’s morphology at the design
time of the controller.

Our approach in this chapter utilizes a form of distributed reinforcement
learning. A similar approach were taken by Maes and Brooks who performed
distributed learning on a 6-legged robot which learned to walk[104]. The
learning was distributed to the legs themselves. Similar, in the context of
multi-robot systems, distributed reinforcement learning has been applied
for learning various collective behaviors[108]. To the best of our knowledge,
Chapter 12. Learning with any Morphology

this paper is the first to apply distributed learning to locomotion of modular robots.

Bongard et al. demonstrated learning of locomotion and adaptation to changes in the configuration of a modular robot[8]. They used a self-modeling approach, where the robot developed a model of its own configuration by performing motor actions, which could be matched with sensor information. A model of the robot configuration was evolved to match the sampled sensor data (from accelerometers) in a physical simulator. By co-evolving the model with a locomotion gait, the robot could then learn to move with different morphologies. The work presented here is similar in purpose but different in approach: Our strategy is simple, model-less and computational cheap to allow implementation on the small embedded devices that modular robots usually are.

Marbach and Ijspeert has studied online optimization of locomotion on the YaMoR modular robotic system[106]. Their strategy was based on Powell’s method, which performed a localized search in the space of selected parameters of central pattern generators. Parameters were manually extracted from the modular robot by exploiting symmetries. Online optimization of 7 parameters for achieving fast movement was successfully performed on a physical robot in roughly 15 minutes[161, 162]. As in this paper, they try to realize simple, robust, fast, model-less, life-long learning on a modular robot. The main difference is that we seek to automate the controller design completely in the sense that no parameters have to be extracted from symmetric properties of the robot. Only the robot morphology must be manually assembled from modules with identical control programs. Furthermore, in this work modules have no shared parameters (except time and reward) since learning is completely distributed to the modules. These properties minimize the amount of communication and simplify the implementation.

12.2 A Strategy for Learning Actuation Patterns

The ATRON modules are simple embedded devices with limited communication and computation abilities. Therefore, the learning strategy must require a minimal amount of resources and ideally be simple to implement. In this learning scenario, the robots may decide to self-reconfigure, modules may realistically break down or be reset and modules can manually be added, removed or exchanged at runtime. Hence, the learning strategy must be robust and able to adapt to such events. By utilizing a simple, distributed and concurrent learning strategy such features can be naturally inherent. We let each module learn independently and in parallel based on a single reward signal. The learning is life-long in the sense that there is no special learning phase followed by an exploitation phase.
Algorithm 6 Learning Module Controller

// $Q[A]$ is discounted expected reward $R$ of choosing Action $A$
// $ALPHA$ is the smoothing factor of an exponential moving average
// $1 - EPSILON$ is the proportion of “greedy” action selections
// $Accelerated$ is a boolean for turning on a heuristic

loop
  if $max(Q) < R$ and $Accelerated$ then
    Repeat Action $A$
  else
    Select Action $A$ with max $Q[A]$ with prob. $1 - EPSILON$ otherwise random
  end if
  Execute Action $A$ for $T$ seconds
  Receive Reward $R$
  Update $Q[A] = ALPHA \cdot (R - Q[A])$
end loop

Learning Strategy We utilize a very simple reinforcement learning strategy, see Algorithm 6. In a learning iteration, every module will perform an action and then receive a global reward for that learning iteration. Each module estimates the value of each of its actions with an exponential moving average, which suppress noise and ensures that if the value of an action changes with time so will its estimation. The algorithm can be categorized as a $TD(0)$ with discount factor $\gamma = 0$ and with no representation of the sensor state[176]. A module can perform a number of actions. Each module independently select which action to perform based on a $\epsilon$-greedy selection policy, where a module selects the action with highest estimated reward with a probability of $1 - \epsilon$ and a random action otherwise.

Acceleration Heuristics Performance of a module is highly coupled with the behavior of the other modules in the robot. Therefore, the best action of a module is non-stationary. It can change over time when other modules change their action. Hence, the learning speed is limited by the fact that it must rely on randomness to select a fitter but underestimated action a sufficient number of times before the reward estimation becomes accurate. To speedup the estimation of underestimated action we tested a heuristics to accelerate the learning: If the received reward after a learning period is higher than the highest estimation of any action, the evaluated action may be underestimated and fitter than the current highest estimated action. Note that this is not always true since the fitness evaluation may be noisy. Therefore, a simple heuristic is to repeat the potentially underestimated action, to accelerate the estimation accuracy and presumably accelerate the learning, see Algorithm 6.
Controller Permutations  A robot must select one action for each of its 
modules, therefore, the number of different controllers are $\#\text{actions}^{\#\text{modules}}$.
For example, in this chapter, we use three actions and experiment with 
seven different robots that must learn to select a controller from amongst 27 
two-wheeler with 3 modules) to 531,441 (walker with 12 modules) different 
controller permutations. Therefore, for the larger robots, brute force search 
is not a realistic option.

12.3 Learning with Simulated ATRON Robots

12.3.1 Experimental Setup

Physical Simulation

Simulation experiments are performed in a open-source simulator named 
Unified Simulator for Self-Reconfigurable Robots (USSR)[38]. We have 
developed USSR as an extendable physics simulator for modular robots. 
Therefore, USSR includes implementations of several existing modular robots 
besides the ATRON. The simulator is based on Open Dynamics Engine[160] 
and contains a physical model of a simple world and of ATRON modules. 
The parameters, e.g. strength, speed, weight, etc., of the simulation model 
and the existing hardware platform has been calibrated to ease the transfer 
of controllers developed in simulation to the physical modules. Through 
JNI, USSR is able to run the same controllers as would run on the physical 
platform, however, this is not utilized here.

Learning to Locomote

In the following experiments, every module runs identical learning con-
trollers. Unless otherwise stated, the learning parameters are $ALPHA = 0.1$ 
and $1 - EPSILON = 0.8$. When starting a trial the action value estimation, 
$Q[A]$, is initialized with the first reward received after executing action $A$
In some experiments we compare with randomly moving robots, i.e. we set 
$1 - EPSILON = 0.0$ and do not use the acceleration heuristics. An ATRON 
module may perform the following three actions:

- HomeStop - rotates to 0 degrees and stop
- RightRotate - rotate clockwise 360 degrees
- LeftRotate - rotate counterclockwise 360 degrees

When performing the HomeStop action, a module will always rotate to 
the same home position. After a learning iteration, a module should ideally 
be back at its home position to ensure repeatability. Therefore, a module will 
try to synchronize its progress to follow the rhythm of the learning iteration.
If too far behind, the module will return directly to its home position by taking the shortest path.

The reward is distance traveled by the robot in the duration of a learning iteration. A learning iteration is seven seconds long, since six seconds is the minimum time (without load) to rotate 360 degrees, the extra one second is used for synchronization.

\[ \text{Reward} = \text{Distance Traveled in 7 Seconds} \] (12.1)

One potential limitation with this approach is that the selected action primitives may be insufficient to control all robots, for example, snakes may require oscillating motor primitives. We address this limitation in the following Chapter 13, where we experiment with extensions and alternatives to this basic strategy.
Robot Morphologies

Since each module is running identical programs, the only difference between the different robots is in what configuration the modules are assembled. Figure 12.1 shows seven ATRON robots, with different morphologies, which we used for experiments.

The presented approach is limited to morphologies that do not contain closed loops of modules and we generally avoid configurations that can self-collide. The reasons for this are mainly practical and we do not consider it a principal limitation. We plan to add a control layer below the learning layer to deal with these problems.

12.3.2 Experimental Results and Discussion

In this section, we present experiments on simulated ATRON modules to study the characteristics of this learning strategy. We study a typical learning trial, followed by systematic learning experiments on different morphologies and its ability to adapt to faults and configuration changes during the learning. Finally, we study the scalability characteristics of the learning strategy.

Quadrupedal Crawler

In this experiment, we consider a quadrupedal crawler consisting of 8 ATRON modules. To simplify the analysis we disable (i.e. stops in the home position) four of the modules and only allow the four legs to be active, as indicated in Figure 12.2(a). Also, we force the robot to start learning from a completely stopped state by initializing $Q[A]$ to 0.1 for the HomeStop action and to 0.0 for the other actions. Our objective is to investigate how the proposed learning strategy behaves on a typical robot.

First consider the two typical learning examples given in Figure 12.2. The contour plot in Figure 12.2(b) illustrates how the robot controller transitions to gradually better robot controllers. The controller eventually converges to one of the four optimums, which corresponds to the symmetry axes of the robot (although in one case the robot has a single step fallback to another controller). The graphs in Figure 12.2(c) and 12.2(d) shows how the velocity of the robots jumps in discrete steps, that corresponds to changes in the preferred actions of modules.

Figure 12.3 compares the convergence speed and performance of the learning with and without the acceleration heuristic. The time to converge is measured from the start of a trial until the controller transitioned to one of the four optimal solutions. In all 20 trials the robot converged, in 4 trials the robot had short fallbacks to non-optimal controllers (as in Figure 12.2(c)). On average accelerated learning converged faster (19 minutes or 1146 iterations) than normal learning (32 minutes or 1898 iterations). The
12.3. Learning with Simulated ATRON Robots

Figure 12.2: Typical simulated learning examples with and without the acceleration heuristic. (a) Eight module quadrupedal crawler (four active modules). (b) Contour plot with each point indicating the velocity of a robot performing the corresponding controller (average of 10 trials per point). The arrows show the transitions of the preferred controller of the robot. (c) And (d) shows the corresponding rewards received by the robots in duration of one hour. The horizontal lines indicate the expected velocity based on the same data as the contour plot.
Chapter 12. Learning with any Morphology

Figure 12.3: The velocity of a quadrupedal crawler with four active modules as a function of time. Each point is the average of 10 trials. The horizontal bars indicate average convergence time and standard deviation. Note, that accelerated learning converges significantly faster (P=0.0023) for this robot.

The difference is statistically significant (P=0.0023). Note, that the convergence speed is severely prolonged by the facts that the robot is forced to start in a stopped position. Also note, that accelerated learning on average reaches a higher velocity, this is not due to the type of solutions found. Rather the faster velocity is due to the acceleration heuristics, which tends to repeat good performing actions at the cost of random exploration. This can also be seen by comparing Figure 12.2(c) with 12.2(d).

As summarized in Table 12.1 the learning strategy behaves in roughly the same way independent of the acceleration heuristic. A typical learning trial consists of 4-5 controller transitions, where a module changes its preferred action before the controller converges. In about 90% of these transitions it will only changes the action of one module. This indicates that at a global level the robot is performing a localized random search in the controller space. Although, the individual modules are collectively searching in any explicit manner, this global strategy emerges from the local strategy of the individual modules.

Different Morphologies

An important requirement of the proposed online learning strategy is the ability to learn to move with many different robot morphologies without changing the control. In this experiment, we perform online learning with seven different simulated ATRON robots, see Figure 12.1. In each learning trial, the robot had 30 minutes to optimize its velocity. For each robot type
12.3. Learning with Simulated ATRON Robots

<table>
<thead>
<tr>
<th>Transitions per Trial</th>
<th>Normal</th>
<th>Accelerated</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-Step Transitions</td>
<td>87%</td>
<td>90%</td>
</tr>
<tr>
<td>2-Step Transitions</td>
<td>13%</td>
<td>6%</td>
</tr>
<tr>
<td>3-Step Transitions</td>
<td>0%</td>
<td>4%</td>
</tr>
<tr>
<td>4-Step Transitions</td>
<td>0%</td>
<td>0%</td>
</tr>
</tbody>
</table>

Table 12.1: Average number of controller transitions to reach optimal solution, with standard deviations in parentheses. To measure the number of controller transitions very brief transitions of one or two learning steps (7-14 seconds) are censored away. The results are based on 10 trials of quadrupedal crawler with 4 active modules learning to move. Note, that there is no significant difference in the type of controller transitions. Also, 1-step transitions are by far the most common, which indicate that the search is localized.

Figure 12.4: Velocity at the end of learning in simulation. Each bar is the average velocity (reward) from the 25 to the 30 minute of 10 independent trials. Error bars indicate one standard deviation of average robot velocity. Note, that both normal and accelerated learning has an average higher velocity than random movement. In addition, both types of learning converge to similar performing gaits but that accelerated learning removes some random exploration, why it on average moves the robot faster.
10 independent trials were performed. Results are shown in Figure 12.4.

Compared to randomly behaving robots, both normal and accelerated learning improves the average velocity significantly. We observe that each robot always tends to learn the same, i.e., symmetrically equivalent gaits. There is no difference in which types of gaits the normal and accelerated learning strategy finds. Overall, the learning of locomotion is effective and the controllers are in most cases identical to those we would design by hand using the same action primitives. Except for the snake robot, which seems to have no good controller given the current set of action primitives. In very few cases, the learning converges to a suboptimal solution or fails to converge at all within the given 30 minutes. There is no general trend in the how the morphology affects the learned gaits. For example, there is no trend that smaller robots or larger robots are faster, except that wheeled locomotion is faster than legged locomotion.

Self-Reconfiguration and Faults

In this experiment, we test for the learning strategy’s ability to adapt to changes in robot morphology. Initially we let a crawler type robot learn to move. At learning iteration 250 (after 29 minutes), the robot is then programmed to self-reconfigure into a quadrupedal type robot. Afterwards the learning is continued without resetting the learning system. After additional
250 iterations, we simulate a module failure by stopping a leg module in a non-home position. 250 iterations later we reactivate the module and let the learning continue for another 250 iterations.

Figure 12.5 shows the average results of 10 trials. After both the self-reconfiguration and module fault, we observe a drop in fitness as expected. In both cases, the learning system is able to adapt to its changed morphology and regain a higher velocity. In the case there a leg module is reactivated there is no initial drop in fitness, but afterwards the robot learns again to use its leg and the average velocity increases again.

**Scalability**

To study the scalability of the learning strategy we performed experiments with a scalable robot. We utilized a millipede robot as shown in Figure 12.6. This robot has a best-known controller as indicated in the figure. In the following experiments, we vary the number of legs from 4 to 36 in steps of 4 with 10 learning trials per robot. We use the basic action set (rotate left, right and stop in home) and enable the acceleration heuristic.

We define the time of convergence as the time at which 85% of the leg modules has learned to contribute to the robot movement. That is, the leg module rotates either left or right dependent on its position in the robot and the direction of locomotion. The time to converge is shown in Figure 12.7(a). As expected, an increase in number of modules also increase the convergence time, the relation is approximately linear for this robot in the interval shown. The increase in convergence time is rather slow, for each module added the convergence time is prolonged with 52 seconds (based on
Chapter 12. Learning with any Morphology

Figure 12.7: (a) Convergence time versus number of modules. (b) Divergence versus number of modules. The robots are millipedes with 4 to 24 legs. Error bars indicate one standard deviation.

Figure 12.8: The graph shows an clear example of divergence, during learning of a millipede with 20 leg pairs (100 modules). The learning diverges twice, where the direction of locomotion shifts between forward and backward. The robot’s maximum velocity is 3 cm/s if all legs were contributing to the movement.

A least square fit: \( \text{convergenceTime} = 52 \cdot \#\text{modules} + 182 \) seconds. Beyond this interval of up to 60 modules, divergence becomes the dominating factor, i.e. the robot forgets already learned behavior.

We measure learning divergence as a major drop in number of leg modules contributing to moving the millipede. A clear example of divergence is shown in Figure 12.8 for a 100 module robot. The frequency of diverges of each robot is shown in Figure 12.7(b). We observe that the divergence frequency increases with the number of modules. The reason behind this is that as the number of modules increase the effect that any individual module has on the robot decreases. Therefore, for a given module the estimates for each of its actions will almost be identical and small disturbance can cause the divergence effect. This effect is illustrated in Figure 12.9 for a 6-legged...
12.4 Learning with Physical ATRON Robots

Figure 12.9: The intuition behind divergence in large scale robots. If the number of modules are low a single module may have a large effect on the overall velocity compared to noise, the opposite is the case for many module robots.

and 20 legged millipede. Initially the robot is moving with the best know controller. Then, every 200 seconds, one of the leg modules reverse its action from rotating left to right and visa versa. This cause the robot to slow down its movement until it is almost fully stopped. For the small robot, the difference in velocity from one module change to the other is large and a module can easily detect the difference. For the large robot, this difference is small compared to noise, which explains why the divergence effect increases with the number of modules.

12.4 Learning with Physical ATRON Robots

In the previous section, we studied the configuration independent learning strategy purely in simulation. To validate our results, in this section we perform experiments on controller transference from simulation to reality and of online learning on physical ATRON robots.

12.4.1 Experimental Setup

Transference Here we transfer controllers learned in simulation to physical robots. The code transference is trivial since each module just performs a basic motor action. The environment is a normal linoleum office floor. For comparison, we measure the locomotion velocity of identical robots and controllers in simulation and on the real robots.

Online Learning The ATRON modules are not equipped with sensor, which allows them to measure their own velocity or distance traveled, as required for the reward signal. To compensate for this we construct a setup, which consists of an arena with an overhead camera connected to a server.
Figure 12.10 illustrates the experimental setup. The server tracks the robot and sends a reward signal to the robot. The reward signal is, as in the previous section, distance traveled within 7 seconds. The original ATRON module does not have wireless communication. For this (and other) reasons, we are developing a number of modified ATRON modules, which have an integrated Sun SPOT. In these experiments, we utilize that the Sun SPOT is equipped with wireless communication. In each learning robot, a single Sun SPOT enabled ATRON module is used, which receives reward updates from the server. The Sun SPOT enabled ATRONs are in development and for reliability reasons we do not actuate these modules in these experiments. Instead, we place the Sun SPOT modules so that its effect on the learning results can be disregarded.

The learning algorithm, presented in previous sections, is running on the modules. Each module runs identical programs, is learning independently and in parallel with other modules. With 10 Hz every module sends a message containing its current state, timestep and reward to all of its neighbors through its infrared communication channels. The timestep is incremented and the reward updated from the server side every 7 seconds. When a new update is received a module performs a learning update and iteration as defined in Algorithm 6. The state can from the server side be set to paused or learning. The robot is paused by the server when it moves beyond the borders of the arena and is then manually moved back onto the arena before the learning is continued. In the presented results the paused time intervals have been removed.
12.4. Learning with Physical ATRON Robots

Figure 12.11: (a)-(c) Images from locomotion experiments with ATRON robots, where controllers learned in simulation have been transferred to physical robots.
Table 12.2: Velocity when transferring controllers to hardware.

<table>
<thead>
<tr>
<th>Robot Type</th>
<th>Simulation</th>
<th>Reality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Two Wheeler</td>
<td>4.1 (0.11)</td>
<td>6.2 (0.28)</td>
</tr>
<tr>
<td>Bipedal</td>
<td>2.7 (0.063)</td>
<td>6.7 (0.42)</td>
</tr>
<tr>
<td>Quadrupedal</td>
<td>2.7 (0.23)</td>
<td>5.3 (0.66)</td>
</tr>
</tbody>
</table>

12.4.2 Experimental Results and Discussion

Transference from Simulation to Reality

In these experiments, we have transferred controller learned in simulation to the physical modules. For the two-wheeler, bipedal and quadrupedal robots (see Figure 12.11) a typical controller, learned in simulation, is selected. We performed fifteen experiments in both simulation and on the physical hardware with the same controller and measured the velocity as distance traveled. Table 12.2 shows the results. Although, we carefully tuned the simulation parameters the velocity in simulation and reality is quite different, mainly slipping between surface and robot happens too much in the simulation. Since, we are using quite different morphologies such as wheeled and legged locomotion, we found no single parameter setting in the simulator which were better for all three robots. Such limitations are to be expected and qualitatively the locomotion gaits are identical in both in the simulated and real world. Based on these observations we expect learning to perform similar on the physical robot as it did in simulation.

Physical Online Learning

In these experiments, learning is performed directly on the modules and only the reward signal is computed externally. We perform experiments with two different robots, a three-module two-wheeler and an eight-module quadrupedal, which has a passive ninth module for wireless communication. For each robot, we report on five experimental trials, two extra experiments (one for each robot) were censored away due to mechanical failures during the experiments. An experimental trial ran until the robot had convincingly converged to a near optimal solution. Since not all physical experiments are of equal duration, we pad some experiments with the average velocity of its last 10 learning iterations to generate the graphs of Figure 12.13(a) and 12.15(a). In total, we report on more than 4 hours of physical experimental time.

Two-wheeler

A sequence of pictures from a learning experiment with a two-wheeler is shown in Figure 12.12. Table 12.3 shows some details for five experimental trials with a two-wheeler robot. The time to converge to
Figure 12.12: Pictures from learning experiment with two-wheeler. From the first to the second picture the robot moves forward for 7 seconds. In the next three pictures the robot turns, again for seven seconds, by rotating on one wheel and keeping the other stopped. In the five experiments with the two-wheeler robot, the learning converges to either forward or backward movement, which were also the case in simulation.

<table>
<thead>
<tr>
<th>Exp.</th>
<th>Conv. Time (sec)</th>
<th>Exp. Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>28</td>
<td>629</td>
</tr>
<tr>
<td>2</td>
<td>35</td>
<td>707</td>
</tr>
<tr>
<td>3</td>
<td>567</td>
<td>967</td>
</tr>
<tr>
<td>4</td>
<td>28</td>
<td>695</td>
</tr>
<tr>
<td>5</td>
<td>798</td>
<td>1020</td>
</tr>
<tr>
<td>Total</td>
<td>1456</td>
<td>4018</td>
</tr>
<tr>
<td>Phy. mean</td>
<td>291</td>
<td>804</td>
</tr>
<tr>
<td>Sim. mean</td>
<td>225</td>
<td>—</td>
</tr>
</tbody>
</table>

Table 12.3: Results of online learning on two-wheeler robot.
driving either forward or backward is given. For comparison the equivalent convergence time measured in simulation experiments is also given. In three of the five experiments, the robot converges to the best-known solution within the first minute. As were also observed in simulation trials, in the other two trials the robot was stuck for an extended period in a suboptimal behavior before it finally converged. We observe that the physical robot on average converges a minute slower, than the simulated robot, but there is no significant difference (P=0.36) between simulation and physical experiments in terms of mean convergence time. Figure 12.13 shows the average velocity (reward given to the robot) as a function of time for the two-wheeler in both simulation and on the physical robot. The results are similar, except that the physical robot moves faster than in simulation.

**Quadrupedal**  Pictures from an experimental trial is shown in Figure 12.14, where a 9-module quadrupedal (8 active modules and 1 for wireless communication) learns to move. Table 12.4 summarized the result of
12.4. Learning with Physical ATRON Robots

Figure 12.14: Pictures from learning experiment with quadrupedal walker. A 7 seconds period is shown. The robot starts in its home position, performs a locomotion period, and then returns to its home position. In each of the five experiments, the quadrupedal converged to symmetrically equivalent gaits. All five gaits were equivalent to the gaits found in simulation.

Figure 12.15: Average velocity of five trials as a function of time for both physical and simulated experiments with a quadrupedal. Points are the average reward in a given timestep, the line indicate the trend.
five experimental trials. In all five trials, the robot converges to a known best gait. The average convergence time is less than 15 minutes, which is slower than the average of 12 minutes it takes to converge in simulation. The difference is, however, not statistical significant (P=0.29). Figure 12.15 shows the average velocity versus time for both simulated and physical experiments with the quadrupedal. We observe that the measured velocity in the physical trials contains more noise than the simulated trials. Further, the physical robot also achieves a higher velocity than in simulation. Another observation we made was that the velocity difference between the fastest and the second fastest gait is smaller in the real experiments than in simulation, which together with the extra noise may explain why the physical trial on average converges almost 3 minutes slower than in simulation.

12.5 Discussion, Conclusion and Future Work

12.5.1 Applied Design Principles

The learning strategy proposed and studied in this chapter follows the design principles described in Chapter 3. In particular, the principles related to robot and module behavior design applies. Since the learning is distributed to independently acting modules (follows Principle 8), each running identical learning program (follows Principle 12) the robot behavior emerges from the collective behavior of its module (follows Principle 6). A module learns a context dependent behavior (follows Principle 10), by relying on peer modules in the robot to learn their appropriate behavior (follows Principle 9). The ability to adapt allow modules to be dynamically replaced (follows Principle 11) and keeps the robot from being critically depending on any of its modules (follows Principle 7) as we say in experiments on module faults and self-reconfiguration.

The main design principle violation of the proposed learning strategy is the use of an external global reward signal. The successfulness of the system is highly dependent on the existence and quality of this signal. Ideally, a more resilient learning system would only rely on what could be sensed locally or collectively by the modules. Hence, we can see this external global reward signal as a partial violation of Principle 8. This violation can be addressed by adding sensors to the robots.

12.5.2 Conclusion

In this chapter, we explored a online learning strategy for modular robots. The learning strategy is simple to implement since it is distributed and model-less. Further, the strategy allows us to assemble learning robots from modules without changing any part of the program or putting severe constraints on the types of robot morphologies.
In simulation we studied a learning quadrupedal crawler and found that from its independently learning modules, a higher-level learning strategy emerged, which were similar to localized random search. We performed experiments in simulation of ATRON modules, which indicates that the strategy is sufficient to learn quite efficient locomotion gaits for a large range of different morphologies up to 12-module robots. A typical learning trial converged in less than 15 minutes depending on the size and type of the robot. We also performed simulated experiments with faults and self-reconfiguration that illustrated the advantages of utilizing a configuration independent learning strategy. We saw that the modules after reconfiguration were able to learn to move with a new morphology and adapt to module faults. In simulation, we studied the scalability characteristics of the learning strategy and found that it could learn to move an robot with up to 60 modules (60 DOF millipede). However, the effects of divergence in the learning would eventually become dominant and prevent the robot from being scaled further up. We also found that the convergence time increased slowly approximately linearly, with the number of modules within the functional range.

We transferred several gaits to the physical platform to validate the simulation. We found that the behaviors were equivalent, although the actual measured speed varied from simulation to reality. Further, we performed experiments with physical ATRON robots online learning to move. These experiments validated our simulation results, and indicate that the proposed learning strategy may be a practical approach to design locomotion gaits.

12.5.3 Future Work

A potential future direction could be to increase the behavioral complexity by utilizing a skill-based architecture to organize different learned motor skills such as move forward or turn left. At a higher control level, it might also be beneficial to store efficient, already found, controllers, to counter-effect the change of divergence in the learning strategy. We also plan to include self-reconfiguration in the learning loop to allow the robot to learn not only a suitable behavior but also a suitable morphology. Finally, a future direction is sensor feedback at the module level as a local reward or at a global level to orchestrate robot behavior.
Chapter 13
Learning with any Module

In the previous chapter, we designed a basic learning strategy for the ATRON robot. The strategy was minimal with each module learning to select one of just three actions: stop in a home position, rotate 360-degree left and rotate 360-degree right. This is possible with the ATRON since effective locomotion gaits can be constructed with such simple actions. However, most other modular systems do not have infinite rotation and can therefore not produce locomotion gaits with these actions. Therefore, in this chapter, we study two extensions to the learning strategy, which makes it more generic. The first extended strategy let each module learn several things in parallel. Concretely, we let each module learn both its actuation action and its home position. The second extension lets each module learn the parameters of a gait-table, which is a more generic locomotion framework. This extension makes the learning strategy directly applicable to almost every module type. We test the algorithms on three different systems: normal ATRON, joint limited ATRON and M-TRAN. Results show that the extended learning strategy enables robots, made from the three systems, to learn effective locomotion gaits.

13.1 Introduction

13.1.1 Design Requirements

The basic learning strategy, from the previous chapter, has a number of design features: i) it is distributed and model-less, to be robust and configuration independent, ii) it is online, to enable life-long learning and iii) it is minimal, to enable simple implementation. However, it is also quite closely tied to the kinematic of the ATRON modules. Therefore, we will extend the basic learning strategy to fulfill two more design features:

Extendible If each module is able to learn several things at the same time the learning strategy becomes more extendible and flexible. For this we
will take a distributed approach within a module, i.e., we will let each
module run several learning processes in parallel which are learning
independently based on the same global reward signal.

Module Independence  To enable the learning strategy to learn indepen-
dently on a specific module design, it must be independent on the
specific module kinematics. A module may have any number of de-
grees of freedom and it may have both translational and rotational
(limited or unlimited) types of actuation. To realize this we will use a
generic control framework, which can be optimized online.

13.1.2 Related Work

To the best of our knowledge, no previous work has proposed an online
learning strategy that is independent on the module type. However, several
locomotion control strategies are generic enough to control different mod-
ules. One example is central pattern generators (CPG), which has both
been used to control the YaMoR and M-TRAN systems. As described in
Chapter 12, Marbach et al. have studied an online learning strategy for the
YaMoR system, which learns CPG parameters[106, 161]. CPG would be
able to control most existing modular robots. However, for a system such as
the ATRON it is not straightforward to take advantage of the oscillations
produced by CPGs, for its unlimited rotation ability. Instead, we will use
gait-tables as the basic learning framework. Recall, that gait-tables were
originally developed by Yim to control the locomotion of Polypod[199]. A
gait-table is a two dimensional table, where the columns are identified with
a module id and the rows are identified with a time-stamp. The values in the
gait-table are angle (or linear) positions that a module, with a given id, must
actuate toward at the corresponding time-stamp. In this sense, the row of a
gait-table defines a pose of the robot. To produce periodic locomotion gaits,
the gait-table is evaluated in a cyclic fashion. The advantages of gait-tables
are that they can easily be applied to almost any module type, independent
on their type of actuation. Further, they are simple to understand and to
implement and can produce any feasible locomotion pattern, since they do
not directly impose any particular style of actuation, e.g., oscillations. The
granularity of gait-tables can be increased by introducing more rows and
they can be scaled up to more modules by adding more columns. The weak-
ness of a gait-table is that it is open-loop, with no direct way to introduce
sensor feedback. In Section 13.2 we will describe how to learn the angle
values of a gait-table online and in a distributed fashion.
13.2 Extended Learning Strategy

13.2.1 Learning Pose and Actuation

In the basic learning strategy the robot always return to its initial pose (the posture the robot assume) at the end of a learning iteration. However, this “home pose” may not always be optimal for the given morphology. Therefore, we extend the basic strategy to also learn the home pose of the robot. We do so by adding another learning process (i.e. instance of Algorithm 6) in each module to learn its home position. This learning process learns in parallel and independently from the other learning process, which is learning actuation. Both processes receive the same global reward signal. Figure 13.1 illustrates the learning strategy. This approach of learning pose and actuation in parallel can easily be generalized simply by adding a learning process for each module component that should be optimized.

13.2.2 Learning Gait Table

To enable learning with any module, we utilize a generic locomotion framework that can be learned online. For this purpose, we have selected gait control tables, which define the goal angles for the motor controller in a give time period. To learn the angle values in a gait-table we take the approach of parallel learning processes. Hence, we have a learning process for each entity in the table, which means each module has one learning process for each row. The learning processes learn independently and in parallel based on a shared reward signal. The strategy is illustrated in Figure 13.2. To utilize this approach for a given system we must define the set of angle values that can fill the table and the number of rows in the table.
13.3 Extended Learning with ATRON

In this experiment, we will compare the performance of the basic learning strategy with the performance of the two extended learning strategies. We will use these strategies to learn actuation, actuation plus pose, and a three-row gait-table for seven different ATRON robots.

13.3.1 Experimental Setup

We utilize the seven different ATRON robot morphologies, which were presented in Chapter 12 and shown in Figure 12.1. For each learning strategy, we perform 10 simulated experimental trials per morphology. Each trial runs for one simulated hour. The three learning strategies all utilize the acceleration heuristic. A learning iteration is seven seconds, to allow one full 360-degree rotation. Reward is distance traveled by the robot’s center of mass within a learning iteration. In general, the learning space for a robot with N modules contains $C^N$ controller permutations, where $C$ is the number of permutations per module.

The first learning strategy is the basic strategy also used in Chapter 12. These experiment are identical to those presented in Section 12.3.2. The module learns to select one of three actions: rotate left, rotate right and stop in a 0-degree home position. When stopped or when a learning iteration starts the module will start from its home position. The learning parameters of Algorithm 6 are $\text{ALPHA} = 0.1$ and $1 - \text{EPSILON} = 0.8$. The size of the learning space for a module is $C = 3$ permutations per module.

The seconds learning strategy extends the basic learning strategy to also learn the pose. Each module learns which one of four positions, should be the home position of its actuator: 0, 90, 180, 270 degree. The pose learning is performed in parallel with the actuation learning, but delayed by a factor of
13.3. Extended Learning with ATRON

Figure 13.3: Average velocity at the end of learning with three learning strategies on seven different ATRON robots. Each bar is the average velocity (reward) from the 50 to the 60 minute of 10 independent trials. For comparison, we also show the average velocity of robots that are moving randomly using the basic actions. Error bars indicate one standard deviation of average robot velocity.

Therefore, the pose learning sums up three rewards (for 21 seconds) before performing a learning update. Parameters are $ALPHA = 0.1$ and $1 - EPSILON = 0.8$ for both of pose and actuation learning. The size of the learning space for a module is $C = 3 \cdot 4 = 12$ permutations per module.

The third learning strategy learns the angle values of a gaits-table. The gait-table contains three rows with an angle value per module. Each value in the gait-table can take on one of five values: 0, 72, 144, 216, 288 degrees. The ATRON module will rotate shortest path toward that angle value. The controller performs each gait-table row for $7/3$ seconds, so that one full cycle takes 7 seconds. Parameters are $ALPHA = 0.1$ and $1 - EPSILON = 0.93$ for each of the three learning processes. The size of the learning space for a module is $C = 5^3 = 125$ permutations per module.

13.3.2 Results

The average velocity at the end of a learning trial is shown in Figure 13.3 and the median velocity during learning for four of the seven robots is shown in Figure 13.4. We observe that the three learning strategies tend to reach similar levels of performance for the different robots. This is confirmed by inspecting the gaits, which in almost all the cases are identical in terms of actuation. This means that the gait-table based strategy also converge
Figure 13.4: Convergence graphs for the four selected ATRON robots. Each graph is the median velocity of 10 trials, where the velocity in each trail has been smoothed. Therefore, a graph can be seen as a “typical trial”. Note how the two-wheeler converge slower using the gait-table based strategy, than with the other two strategies. The snake tends to diverge, especially for the basic strategy since its combination of pose and actions are not appropriate for this morphology. The quadrupedal learns to move faster when it also optimizes its pose. This is not the case for the walker, where the extra learning space only slows down the convergence.

to a continuous rotation style gaits, the same as the other two strategies. The found gaits are generally the fastest known for the different robots and similar to what would be designed by hand. As can be seen in Figure 13.3, in the case of the snake and quadrupedal the parallel learning of pose and actuation reach a significantly higher average velocity than learning of actuation alone ($P = 0.0090$ and $P = 4.4 \cdot 10^{-6}$). This indicate that the initial pose is not the optimal for these robots, visual inspections of found gaits confirms this. Further, since the learning space of the gait-table strategy is larger than the other two strategies, the found gaits tend to vary more and converge slower.

Figure 13.5 shows the average, minimum and maximum convergence time for the different combinations of robots and learning strategies. We measured the convergence time of a trial as the time for its average velocity
13.4. Extended Learning with Joint Limited ATRON

In this experiment we will use joint limited ATRON modules, i.e., which cannot be rotated infinitely. In effect, the gait-table based learning strategy must find alternative gaits to move the robots, instead of gaits based on continuous rotations as found in previous experiments. The purpose is to study how the learning works on a system, where the modules have a
different kinematics.

13.4.1 Experimental Setup

The simulation model of the ATRON module is modified for this experiment. Instead of being able to rotate unlimited, the center rotation joint is limited to only rotate in the ±90 degree interval. Otherwise, its simulation model remains unchanged.

We experiment with a snake-7 (chain with seven modules), millipede-2 (two leg pairs, 10 modules), millipede-3 (three leg pairs, 15 modules) and a quadrupedal (8 modules). See Figure 13.8 for an illustration of the robots.

We use the gait-table based learning strategy to optimize the gaits. The gait-table has 5 rows, so each module must learn five angle values from the set: -60, -30, 0, 30, 60 degrees. Hence, a module has five independent learning processes and the size of the learning space is $C = 5^5 = 3125$ permutations per modules. A learning iteration is 7 seconds. The algorithm parameters are $\text{ALPHA} = 0.1$ and $1 - \text{EPSILON} = 0.96$.

13.4.2 Results

Figure 13.6 shows the average velocity of the different robots compared with randomly moving robots (using the same gait-table approach but with randomly varying angle values). We, observe that the learning strategy is able to find significantly better than random gaits for the different robots. Although, the robots moves slower than if they could perform unlimited rotation, the gaits found are quite efficient. Also, note that in the case of the snake robot the basic learning strategy fails to converge since the robot entangles itself, while this joint limited gait-table based strategy converges to undulation style gaits. Figure 13.7 shows the convergence as the average velocity achieved over time. Note, that a robot tends to quickly learn a better than random gait, and that this gait gradually improve over time. All the 40 experimental trials converged to good performing gaits. Divergence happens in a few cases when a snake robot rolls upside down during learning and then had to learn to move from this state.

Figure 13.8 shows some typical gaits found. The snake is moving with a side-winding gait, with a traveling wave down the chain. The snake lifts parts of its body off the ground as it moves. The shown quadrupedal gait tends to use one of its back foot partly as a wheel, partly as a foot. Its side legs moves back and forward for movement, while the front leg is used just for support. The millipede-2 has a trot style gait, where the diagonal opposite legs move together. The millipede-3 uses a similar gait with each leg oscillating back and forward with some unrecognizable scheme of synchronization between the legs.
Figure 13.6: Average velocity from the 50 to the 60 minute of learning with joint limited ATRON. For comparison, the average velocity a randomly moving robot is also shown. Error bars indicate one standard deviation.

Figure 13.7: Convergence graphs for the four different robots assembled from joint-limited ATRON modules. For comparison the average velocity of random moving robots is also shown. Each graph is the average of 10 trials. Error bars indicate standard deviation.
13.5 Extended Learning with M-TRAN

In this experiment, we will use a simulation of the M-TRAN modules to study how the gait-table based learning strategy works on four different M-TRAN robots.

13.5.1 Experimental Setup

We have implemented a model of the M-TRAN modules in the USSR simulator, based on available specifications[98]. However, we do have access to the physical M-TRAN modules. Therefore, although the kinematics is correct, specific characteristics might be quite different from the real system. We accept this, since our purpose is to validate the gait-table based learning strategy on a different system, not to find efficient locomotion gaits for M-TRAN robots.

We experiment with three different M-TRAN robots. A 6-module caterpillar (12 DOF), a 4-module mini walker (8 DOF) and a 8-module walker (16 DOF). The robots are illustrated in Figure 13.11. For each robot, we define an initial pose that the actuation is performed relative to. Selecting a pose is a tradeoff between high potential to move and being stable so that the robot does not fall over while learning.

Since an M-TRAN module has two actuators, we let each actuator be controlled by independent gait-tables. Again, each entry in each of the gait-
13.5. Extended Learning with M-TRAN

A major challenge with learning M-TRAN gaits is that the robot often falls over while learning. This happened in 23 percent, 8 percent and 47 percent of the two hour trials with the mini walker, caterpillar and walker respectively. These trials were censored away in the presented results, which is based on 10 completed trials per robot.

Figure 13.9 shows the average velocity after one hour of learning compared to randomly moving robots. The learning succeeds in finding efficient gaits for all three robots. Because of the short learning iteration (1.5 seconds) even a pose shift can be measured as quite high velocity, why randomly moving robots incorrectly seems to move quite fast. Figure 13.10 shows convergence graphs. Notice, that the performance of the gaits quickly becomes better than random and that the gaits gradually improves over time. The large learning space leaves room for incremental learning.

Figure 13.11 shows some typical learned gaits. Typical gaits for the mini walker consist of hopping movement, with two modules producing movement and two modules creating stability. For the caterpillar, the learning typically finds gaits either with a horizontal traveling wave down the chain of modules or gaits that uses the head and tail modules to push on the
Figure 13.10: Average velocity as a function of time for three M-TRAN robots. Each graph is the average velocity of 10 independent trials. Average velocity of randomly moving robots is shown for comparison. Errorbars indicate one standard deviation.
Figure 13.11: Typical gaits found for M-TRAN robots. Each picture sequence shows a learning iteration of 1.5 seconds. The mini walker moves in small hops (moves left to right), the caterpillar moves with a horizontal traveling wave (moves right to left) and the walker moves with three legs on the ground and one in the air (moves right to left).

ground. Successful gaits for the walker take relative short steps, since the robot would otherwise fall over. In the shown gait, the walker use three legs to produce movement, while the forth leg is kept lifted off the ground in front of the robot.

13.6 Discussion, Conclusion and Future Work

13.6.1 Applied Design Principles

The extensions made in this chapter follow the design principles of Chapter 3. Here, the same discussion applies as for the basic learning strategy, see Chapter 12. The main difference is that the extended strategies expand the degree to which it is distributed (extends the application of Principle 8). This is because we utilize independent learning processes to learn different components within a module. In this chapter, such components were actuator actions, encoder home position and time-sliced actuation goal positions. This distribution process can be extended further to let every component within a module learn independently and in parallel, e.g., each connector could learn if it should be connected or disconnected.

13.6.2 Conclusion

In this chapter, we presented simulated experiments on configuration independent learning of locomotion for a wide range of robots constructed from three different systems: ATRON, joint limited ATRON and M-TRAN. We
extended the basic learning strategy in two ways: i) First extension also include learning the pose of a robot. ii) Second extension is to learn the angle values in a gait-table. These extended strategies are distributed, with several parallel and independently learning processes running on each module. We saw that the velocity of gaits for ATRON robots in some cases could be improved by also learning the pose. The gait-table based strategy, demonstrated that it was able to learn effective gaits for the different robots and systems. However, as could be expected, we also saw that the increased size of the learning space came at the cost of prolonged time to learn a gait. Yet, even the most complex gaits are typically learned within one hour. In conclusion, learning can effectively be distributed by introducing independent processes learning in parallel. Further, the extended learning strategy based on gait-tables is a simple to implement learning strategy, which can be used on almost any existing modular platform.

13.6.3 Future Work

Besides the future work suggested at the end of Chapter 12, two issues become apparent from the experiments presented in this chapter. The first issue is that robots tend to fall over while learning, in general, we need to address issues of detection and recovery from events that harness the learning. Most likely, this complexity does not belong at the level of learning, but somewhere else in the higher-level control of the robot. A second issue, is that of the duration of a learning iteration and thereby the length of a locomotion period. Currently we manually select this parameter for the system. However, it is clear that not all morphologies and gait types should be forced to use the same duration. Therefore, future work will explore ways to make it adaptive, so that the robot can rapidly adjust the duration of a locomotion period to the environment, its morphology and its style of gait.
Part VI

Concluding Remarks
Chapter 14

Conclusion

14.1 Summary

The road to artificial autonomous systems is long and uncertain. In this thesis, we took the strategic path of self-reconfigurable modular robots. From one perspective, this journey is equivalent to that of conventional robots, just with the extra complexity of a distributed modular system. From another perspective, this journey is completely different since it by nature, and as nature, is distributed. This alternative path could potentially make the journey harder or it could make it simpler - we will never know unless we try it. In this thesis, we explored a small section of the road and described a sketchy map that we anticipate can be used in future explorations.

First, we considered the design of self-reconfigurable robots. We took a holistic approach by limiting the scope to scalable robots autonomously able to perform a versatile range of physical tasks. We decided on a set of design principles, which advocate simple, self-sufficient and distributed solutions. By following these design principles throughout the thesis, we gave them validation. Our first use of these principles was on basic communication issues in the ATRON system. The algorithms were distributed, self-organizing and proved simple to implement.

Second, we considered the scalable nature of self-reconfigurable robots. To understand the effects of module size, we analyzed an electrostatic model of Catoms and found that they potentially could become faster and stronger when scaling down their size. We also found that volume based approaches to self-reconfiguration scales well, while surface based approaches do not. Therefore, this is a critical choice when designing a scalable self-reconfiguration algorithm. A second fundamental issue is how morphology and control are interdependent. We studied this issue in terms of module size and chain length for snake-like locomotion of simulated Catoms and found that the most effective gaits varied with chain morphology and module size from caterpillar to side-winding gaits. Similar results can likely be found for all
snake robots and can be used to find a sweet spot in the design space. A third
fundamental issue related to scalability is how to organize modules to achieve
scalable control of robot behavior. We addressed this problem by proposing
a method that introduced a hierarchy in the control as well as in the physical
structure of the robot. This anatomy-based organization successfully allowed
us to construct rather complex simulated Catom robots. However, our study
highlighted several problematic issues with the classical design choice of
using homogeneous modules. Instead, we proposed the design principle that
a system should consist of heterogenous, functionally specialized modules to
realize scalable functionality. Currently we are developing a new hardware
platform, Odin, which will allow us to investigate if this design principle is
valid.

Third, we considered the process of self-reconfiguration. This problem
is already well studied, but mainly for abstract types of modules that have
very different characteristics than what was needed for the ATRON sys-
tem. To address this challenge we developed a novel distributed control
strategy, where meta-modules emerge from passive modules and only have
a limited time of existence. First, we experimentally compared several poten-
tial meta-modules and developed a set of measurements for selecting a
good meta-module design. Then, by using evolution of artificial neural net-
works, we automatically developed a controller that could control the meta-
modules and demonstrated how this control strategy was both scalable and
fault tolerant. To get this far, several concepts besides emergent meta-
modules were introduced, such as local reachable-space, module life cycle
and meta-actions. Such concepts could find its application to control other
systems as well. Overall, the studied meta-module based approach proved as
a good validation of thouse of our design principles, which states that mod-
ules should be dynamically replaceable and controlled autonomously based
on their local context.

Forth and finally, we addressed the problem of adaptive collective be-
havior by developing a learning strategy to learn efficient locomotion gaits
for modular robots. Our approach was unconventional in the sense that we
did not utilize any central optimization algorithm. Instead, we designed a
distributed strategy where each module independently tried to optimize a
global reward signal. We found, perhaps a bit surprisingly, that a central
strategy emerged from these independent learning components. Collectively
the modules were performing a localized random search in the controller
space. This strategy proved sufficient to find effective gaits for a large range
of different ATRON robots. However, as one would expect there are limits
to this strategy’s scalability. Nevertheless, is it sufficient for robots with up
to several dozens degree of freedom and it has several other good charac-
teristics, such as simplicity, configuration independence, as well as ability to
adapt to faults and self-reconfiguration. Further, a simple extension of the
algorithm proved sufficient to learn gaits for both M-TRAN and joint lim-
The contributions of this thesis can be summarized in the following categories.

**Design Principles** We presented a set of design principles, many are well known, but not generally agreed upon. The principles fell within the areas of system, module, robot, and behavior design. Our contribution is to collect these principles to a coherent whole and give them validation by utilizing them throughout the thesis.

**Communication** We developed simple distributed communication algorithms for crosstalk and neighbor detections. The neighbor detection algorithm is generic and can be used on most modular robots, while the crosstalk algorithm is ATRON specific.

**Scalability** We analyzed the scale-effects on module size and numbers. Specifically, we analyzed how the strength and speed of modules change with size and how the self-reconfiguration problem changes with the number of modules.

**Embodiment** We mapped the interdependence between morphology, scale and control for the task of snake-like locomotion. We concluded that none of these aspects can be studied in isolation.

**Organization** We proposed a hierarchical method to module organization, inspired by the anatomy of biological organisms. This is the first method that can guide the development of scalable fixed-topology modular robots.

**Emergent Meta-modules** We developed a set of measurements to analyze and compare meta-modules and presented a novel, distributed, emergent approach to control meta-modules. Further, we demonstrated how controllers automatically could be generated with evolution and we conducted some of the largest and longest physical 3D self-reconfiguration experiments to date (only on the M-TRAN larger and longer experiments has been demonstrated).
Learning Strategy We developed a simple, distributed, online learning strategy for modular robots, which can quickly learn effective locomotion gaits independent on the robot’s configuration and module type.

Overall, we believe that these contributions shows significant progress in the area of simple, distributed, practical control of modular self-reconfigurable robots. Several of the control algorithms are directly transferable to other system, and the developed concepts, ideas, and principles can inspire the development of the next generation of systems.

14.3 Publications

It is a requirement that this thesis should contain publishable material. Fortunately, several parts has already been published[37, 34, 31, 39, 35, 32, 33, 36]. Parts which are currently being prepared for publication or which are under review, include the results of Chapter 12 and 13. Some published work has only been partially included since it present the overview of several peoples work. includes work about ATRON versatility[12] and about the HYDRA project[124]. Published work which I have co-authored but which is not included since I am not the main contributor, include work on the design of the Odin robot[60, 166], on dedicated programming languages for modular robots[45, 149, 150], on the USSR physical simulator for modular robots[38] and of meta-modules to control 2D surfaces of ATRON modules[11].

14.4 Future Research

The individual chapters of this thesis contains specific suggestions for future work. Generally, we believe that future research should focus on to issues: Hardware and Integration.

Hardware is the basis of robots and the deciding factor for how software is designed. A sweet spot in the design space has been found with systems that all have similar actuated degrees of freedom: Polybot, CKbot, M-TRAN, YaMoR, and SuperBot. These systems have excellent ability to form various snakes and walkers without any serious weaknesses, perhaps except scalability. However, radically different system designs may open up for new possibilities and therefore systems such as Odin and Catom should be explored.

Integration of physical systems, software and tasks is another critical issue. Many control algorithms are developed in isolation, focusing on parts rather than wholes. We speculate that only by considering the integration of a robot in a physical task we will be able to develop the necessary control software. Software development should follow the progress of hardware. Preferably, wild ideas should first be tested in hardware not in software.
Otherwise, the software tends get a life of its own separate from any embodiment.
Chapter 14. Conclusion

Bibliography


Chapter 14. Conclusion


