Search Space Traversal in Co-Optimized Modular Robots

Mia-Katrin Kvalsund

Thesis submitted for the degree of
Master in Informatics: Robotics and Intelligent Systems
60 credits

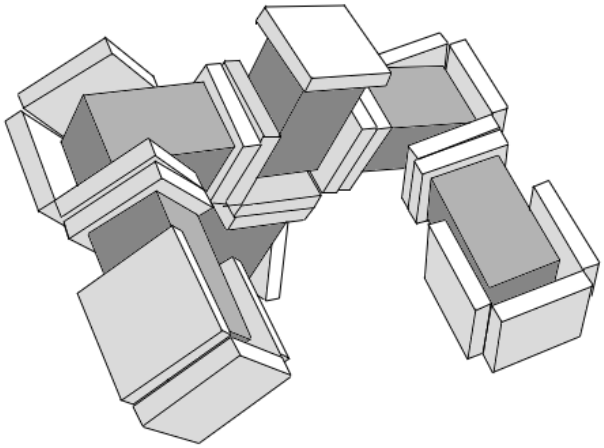
Department of Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2022

Search Space Traversal in Co-Optimized Modular Robots

Mia-Katrin Kvalsund



Search Space Traversal in Co-Optimized Modular Robots

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

In Evolutionary Robotics, Evolutionary Algorithms (EAs) are used to optimize robots. Research has shown that co-optimizing morphology and control can lead to innovative, animal-like behavior. Additionally, co-optimizing in Modular Robotics can be used to automatically produce robots for any task. Even so, it is a definite challenge to design a system that can evolve morphology and control simultaneously. A common issue is that of not being able to properly explore the space of possible robots, and thus not finding the globally best solutions to the task. This is reflected in the field struggling with early convergence of morphology, rugged search landscapes, and overall stagnation.

Here, we conduct two different experiments centered around the co-optimization of morphology and control in modular robots. The first investigates different controllers, comparing centralized and decentralized control strategies regarding their effect on morphology and performance. The second experiment investigates gradual encodings and their effect on smoothing the search space. In our gradual encodings, modules grow out gradually instead of being added with full size.

We found that a controller that duplicates control units across the robot body performs significantly better than other control approaches because it explores more of the search space. This indicates that decentralization with duplication can be useful, and possibly decrease early convergence of morphology, which helps confirm that compressing the search space is often beneficial. We therefore present this as an argument for duplication in controllers in general. In addition, we found that while the gradual encodings did smooth the search space, they led to no better or worse performance than the baseline. Even though we suggest some instances where it might still be advantageous, it largely implies that there is no benefit to these more gradual encodings in a standard EA. Overall, these two experiments corroborate other research findings that there is a trade-off between fine-tuning and coarsely exploring in a search, and that the latter will often be more helpful initially. We hope that future researchers will benefit from the suggested controller and encoding strategies and our further insights into their effect on search space traversal.

Acknowledgements

I would like to thank my two supervisors, Frank Veenstra and Kyrre Glette, for encouraging me to follow my interests and helping me to sow them into a cohesive thesis. You two have been a great support and an inspiration to work with.

A special thanks to my friends and fellow students, who were equal parts distractions and discussion partners. Without you guys, I might have been able to do more work, but been worse off for it.

Lastly, I would like to thank my family and my dog for the love and support.

This work was performed on the Fox supercomputer resource, owned by the University of Oslo Center for Information Technology.

Contents

1 Introduction **1**
- 1.1 Research Motivation . . . 1
- 1.2 Research Questions . . . 4
- 1.3 Contributions . . . 4
- 1.4 Thesis Outline . . . 5

2 Background **7**
- 2.1 Introduction . . . 7
- 2.2 Evolutionary Algorithms . . . 8
  - 2.2.1 Fitness . . . 9
- 2.3 Evolutionary Robotics . . . 10
  - 2.3.1 Co-optimizing morphology and control . . . 11
  - 2.3.2 Encodings . . . 13
  - 2.3.3 Evolvability . . . 15
  - 2.3.4 Control . . . 16
  - 2.3.5 CTRNNs . . . 18
  - 2.3.6 Material . . . 20
  - 2.3.7 Simulators and physics engines . . . 20
  - 2.3.8 Reality gap . . . 21
- 2.4 Modular Robotics . . . 23
  - 2.4.1 Classification . . . 24
  - 2.4.2 The modules . . . 26
  - 2.4.3 Control . . . 28
  - 2.4.4 The use of EA in modular robotics . . . 28
- 2.5 Concluding statement . . . 30

3 Implementation **31**
- 3.1 Overview of the system . . . 32
- 3.2 Software and tools . . . 32
  - 3.2.1 Unity . . . 32
  - 3.2.2 ML-Agents . . . 33
  - 3.2.3 The modules . . . 34

3.3 Implementation 36
3.3.1 Robot representation 36
3.3.2 Evolutionary algorithm 40
3.3.3 Fitness function 40

4 Experiment 1 **43**
4.1 The controllers 43
4.2 Parameter tuning 47
4.3 Results 50
4.3.1 Controller performance 50
4.3.2 Effect on morphology 51
4.3.3 Robustness 55
4.4 Analysis 58

5 Experiment 2 **61**
5.1 Variable scale of modules 62
5.2 The encodings 62
5.3 Parameter tuning 65
5.4 Results 67
5.4.1 Encoding performance 67
5.4.2 Smoothness of landscape 69
5.4.3 Landscape traversal 71
5.5 Analysis 72

6 Discussion **75**
6.1 Summary 75
6.2 Further discussion 77
6.3 Limitations and future work 80
6.4 Ethical considerations 82

7 Conclusion **83**

A For Conferences **97**

List of Figures

3.1 Overview of the system 31
3.2 The EMeRGE module 34
3.3 Examples of well-performing morphologies 35
3.4 Distributions of robot size in populations when varying initialization parameters 37
3.5 The distribution of number of modules in individuals in a random population 39
3.6 An evolved cartwheel in 7 frames 41

4.1 The controllers and how they could map to the modules in a small modular robot 44
4.2 The grids filled for the parameter tuning 48
4.3 The collapsed rows and columns for the sine controller grid search 48
4.4 The fitness progressions for all four controllers 50
4.5 The final fitnesses for all runs for each controller . . . 51
4.6 The progression of number of modules for all runs . . 52
4.7 Robot size plotted versus fitness 53
4.8 Number of morphology changes that led to better fit individuals in each generation interval 54
4.9 The explored morphology landscape, controllers . . . 55
4.10 Fitness preservation when applying module removals 57
4.11 Fitness preservation when applying control disabling 58

5.1 Examples of module sizes 61
5.2 Growing of a module illustrated 64
5.3 The grid search for all controllers with the variable scale encoding 65
5.4 The grid search for all controllers with the growing encoding 66
5.5 The grid search for all controllers with the gradual encoding 66
5.6 Fitness progressions for the encodings with controllers 68

5.7 The different mutations' effects 69
5.8 The morphology landscape exploration by each controller-encoding pair 73

6.1 Phenotypes in a fitness landscape 78
6.2 The different imagined samplings 78

List of Tables

2.1 A categorization of optimization-use in MR-papers . 29

3.1 Software versions 32
3.2 The morphology gene mutation rate values 39

4.1 The gene mutation rates for a CTRNN controller . . . 45
4.2 The mutation rate parameters chosen after the sweep 49
4.3 The p-values for the Mann-Whitney U tests, controllers 51

5.1 The morphology gene mutation rate values 63
5.2 The p-values for the Mann-Whitney U tests, encodings 68
5.3 Unique voxels filled for each controller-encoding pair 72

Chapter 1

Introduction

1.1 Research Motivation

Evolutionary Robotics offers an exciting view of future robotics: When generating robots automatically through Evolutionary Algorithms, innovative robots can be found for any task with a bit of computation and ingenuity. With a good model of the environment, we could automatically produce robots designed to traverse hazardous areas after natural disasters or walk the surface of an alien planet. If applied to self-reconfiguring robot bodies, a robot might even become to be seen as a diverse tool that can transform itself into any shape or function needed by a user.

Evolutionary Algorithms are optimization algorithms that are inspired by evolution in nature and works in a similar way to gradually improve a population of solutions. In Evolutionary Robotics, the algorithms keep a population of robot genomes. The genome encodes for the robot's morphology (its body) and its control. When evaluating a robot genome, it is built into its robot form, called a phenotype. The optimization progresses by creating variations on the genomes in the populations and selecting the better ones to continue optimizing.

With his influential paper from 1994, Karl Sims showed the first virtual creatures automatically evolved for complex tasks such as walking, swimming, and following a light source [1]. Their bodies and its control were evolved simultaneously through co-optimization, allowing natural behaviors to emerge from the interplay between them. In addition, strange creatures were found, the likes of which would likely not have been put together by a human. Sims' creatures garnered a lot of traction over the following decades, with inspired researchers continuing to develop

increasingly more sophisticated developmental stages [2, 3], control systems [4, 5], and pipelines for automatic robot creation [6–8].

Roughly at the same time as Sims' creatures were learning to walk, the first few papers on Modular Robotics were published [9–11]. Here, fully separable building blocks, or modules, containing the functions of a robot are assembled as robot bodies [12]. Because the modules are only loosely connected through magnets or latches, modular robots are uniquely suited for quick prototyping of virtual creatures and autonomous self-reconfiguration and self-assembly. Combining modular robotics with the co-optimizing of bodies and control, robots can be found for any task and assembled either by hand or autonomously. However, combining these approaches have only been investigated the last decade, and there are still many challenges to attend to before Sims' creatures hopefully will be surpassed.

One of the big problems in Evolutionary Robotics follows from artificial evolution's insufficiency at finding the optimal solution in a search space. The ideal artificial evolution would work like Darwinian evolution, where small, consecutive steps are taken to reach some good solution. The middle stages between complex structures would largely be beneficial [13], leading the search towards the solution. In reality, artificial evolution does not experience this smooth and gradual search. The search landscapes tend to be rugged, in that most mutations are deleterious and the few beneficial mutations are not necessarily easy to get to [7]. In addition, landscapes tend to be deceptive, where the road to the destination sometimes demands the search to traverse regions of lower fitness in the search space [14].

Meanwhile, mutation in nature is most often non-lethal to the individual, and populations display both drift and high diversification. Although there are many reasons for this, we highlight the concept of evolvability: A population's ability to produce useful solutions in the future by increasing non-lethal genotypic variation with latent potential for phenotypic diversity [15–18]. Due to their genotypic and phenotypic robustness, they will often undergo fewer phenotypic changes on the road to some ideal phenotype, and they are better able to avoid fitness minima [15, 16, 18]. This robustness and high degree of a populations' latent potential for increased phenotypic diversity makes it so that evolvable systems are better able to explore the search space and find optimal solutions [18].

Designing Evolutionary Algorithms with increased evolvability can be done, with for example different types of robot genotypes, encodings, having different effects on traversal [19]. Recent papers into varying module length and connection faces also suggests that module morphology itself has an effect on evolvability and ultimately performance [20, 21].

Related to this problem of landscape traversal, co-optimizing morphology and control often experiences premature convergence of morphology only. When co-optimizing, evolution will settle for a morphology early on, and continue to optimize the controller. Although the reason for this is not fully known, Cheney et al. suggests that because the controller interacts with the environment through the interface of the body, changes to the body will change the interface and essentially scramble the function of the controller [22]. When considering the evolvability of the system, it can be inferred that there is low morphological evolvability. However, it can also be that morphologies that are robust to control mutations are selected for and quickly dominate [23].

This implies that controllers that inherently are less disruptive when morphological changes are made could increase morphological evolvability and therefore diversity. Thus, in modular robots, for the controller to be robust to morphological changes, control in a new module would have to be functioning quickly. Removing a module would also have to reveal a functioning end-effector module. Perhaps having only a few control units and distributing them across the modules would lead to these units being optimized to work in many different modules. In that case, such a controller would increase morphological evolvability.

Morphological evolvability is very important for co-optimized modular robots because the promise of reconfiguration is dependent on useful bodies being able to be found. Therefore, with the goal of studying how and why morphological evolvability and ultimately performance can be increased, we conduct two experiments centered around the co-optimization of modular robots. These two experiments investigate respectively the controller and the morphology: One attempts to smooth the search landscape by implementing encodings with more gradual morphological mutations. The other will investigate different controllers and their effect on morphology, focusing on our hypothesis of how a controller can increase evolvability.

1.2 Research Questions

When co-optimizing morphology and control, morphology often converges prematurely while the controller continues optimizing. Our hypothesis is that a controller that duplicates control units across the robot body will lead to more morphological diversity and therefore increased fitness. In addition, when co-optimizing, the choice of centralized or decentralized control strategy is often complicated by numbers of inputs and outputs. We will investigate the effects of different controllers. In short, we will seek to answer these research questions:

1.1 What are the effects of a controller that duplicates control units across the morphology?

1.2 How can a controller best be designed when co-optimizing morphology and control?

Often the search landscape in co-optimizing modular robots is quite rugged, where mutational neighbors can have very varied performance. In addition, landscapes are deceptive, where the path to a good solution goes through regions of low fitness. Our hypothesis is that a more gradual encoding can smooth the landscape and lead to better performance. When decreasing the chance for highly detrimental mutations, we hope that we will achieve more evolvability and ultimately better fitness. From this hypothesis, we form these research questions:

2.1 How do encodings with different degrees of gradual mutation affect landscape traversal?

2.2 How do encodings with different degrees of gradual mutation affect performance?

1.3 Contributions

The main contributions of this thesis are 1) a controller that duplicates control units across the body (the copy controller), which causes higher performance and increased morphological exploration, and 2) a gradual encoding that we show smooth the landscape, although it does not lead to higher performance. This points to the importance of sampling, as exemplified by the coarse sampling of the copy controller and the granular sampling of the

gradual encoding. We conclude that coarser samplings lead to increased morphological exploration, with the caveat that it is less able to fine-tune. When prioritizing fine-tuning from the start, the optimization seems to gravitate towards local optima. In addition, the copy controller helps confirm that controllers that are more robust to morphological changes will have more morphological evolvability. This leads to a decrease in premature convergence of morphology, and therefore advocates for the use of duplication of control units in general in co-optimizing modular robots.

One article and one poster abstract were produced as a result of the work on this thesis. As such, they both reuse text and figures from the thesis. The article, titled "Centralized and Decentralized Control in Modular Robots and Their Effect on Morphology", was submitted and accepted for publishing in the proceedings and an oral presentation at the Conference of Artificial Life (ALIFE). The poster abstract, titled "Exploring the Effects of Centralized and Decentralized Control on Morphology and Performance", was submitted to the Advanced Course and Symposium on Artificial Intelligence and Neuroscience (ACAIN) and is at the time of writing under review. They are both included in the appendix.

1.4 Thesis Outline

This thesis consists of seven chapters: Introduction, Background, Implementation, Experiment 1, Experiment 2, Discussion, and Conclusion.

In chapter 2, Background, we present relevant topics that our work is based on. We focus on evolutionary algorithms, evolutionary robotics, and modular robotics.

In chapter 3, Implementation, an overview of the system will be presented. The details surrounding the general setup, like software and tools and the simulations, will be presented. The controllers and encodings to be investigated are omitted, included instead under the relevant experiments.

In chapter 4, Experiment 1, the controller experiments will be presented. This includes the controller's implementation, their tuning, all results, and lastly result analysis. This is also the outline of chapter 5, Experiment 2, concerning encodings.

In chapter 6, Discussion, the results of both experiments will be discussed as a whole, also with limitations, future work and ethical

considerations. Lastly, chapter 7, Conclusion, will conclude and summarize our findings.

Chapter 2

Background

2.1 Introduction

When co-optimizing morphology and control, search algorithms are used to develop both the body and brain of a robot to solve some problem. This can be used for single cases but could also be extended to create autonomous systems where robots are created to solve user-specified problems. For example, this could be utilized by construction workers to specify a robot to help with carrying loads, or rescuers who need a robot to go into a place too dangerous for a human. In both cases, the size and maneuverability needed would change from time to time, and so an adaptive robotic system could be more helpful than a general purpose one.

Following from Karl Sims work on virtual creatures [1], many researchers have continued developing co-optimized creatures with varied approaches. One popular approach is using evolutionary algorithms (EA) that uses a process similar to natural evolution. Here, a population of robots is evolved, and the most suitable robot is selected to solve the problem. These algorithms show potential to create solutions close to those seen in nature. For example, Sims showed that this method could evolve sinusoidal swimming motions similar to eels, and later researchers have evolved dog-like gaits like galloping [24].

While the robots created with this method show a lot of promise, there are still many issues to investigate. From the research done up until now it is clear that performance measures, gene encodings, and material all play a significant role in artificial evolution. There is also the issue of evolving centralized control when co-optimizing, which is generally solved by having distributed control instead.

Lastly, when creatures evolved in simulation transfer to reality,

both the simulator to reality gap and the unfeasible body plans of virtual creatures will pose problems. A proposed solution is found in modular robotics, where building blocks with fixed functionality are assembled into a robot. Modular robots also have the added benefit of being easily reconfigured, as their modules are separate mechatronic systems connected by magnets or similar mechanisms. Combined with EAs, modular robot assemblies can be evolved that are more easily transferable to reality.

The following is an overview of evolutionary algorithms and some key topics in evolutionary robotics, as well as the framework of modular robotics.

2.2 Evolutionary Algorithms

Evolutionary Algorithms (EA) are stochastic search and optimization algorithms inspired by natural evolution. They typically evolve a population of individuals, solutions to some optimization problem, over several generations by using operators such as crossover, mutation, and selection. Crossover combines parents into children, mutation makes a slight change to the child, and selection keeps part of the population for reproduction or survival. Populations are initiated with a random set of individuals or seeded with previously found/human designed solutions. Central to these algorithms are being able to formulate the goal of the optimization in a performance measure.

Any number of things can be optimized with EAs: Programs, neural networks, strings of numbers, and robot morphologies are some examples. In all of these, the operators usually must be balanced to both explore and exploit the search space. Some EAs have been well defined and tested but are usually suited most to some particular task: NeuroEvolution of Augmenting Topologies (NEAT) [25], for example, is used for evolving neural networks.

EA has several subsets of algorithms, like genetic algorithms (GA), evolutionary strategies (ES), neuroevolution, etc. These focus on different aspects, like evolutionary operators on strings of numbers or neural networks [26]. Evolutionary robotics, discussed below, can employ any of these branches of algorithms.

2.2.1 Fitness

The goal of evolution in EAs is usually expressed in some function we want to maximize that measures an individual's performance, or fitness, on some task:

$$fitness = f(i)$$

where i is the individual solution. The measured fitness can then be used to compete in parent and survival selection. For example, in evolving a robot to walk, the goal could be to locomote as far to the right as possible. Expressing it instead as a single value, we measure the displacement of the robot for a standard amount of time as it locomotes in an environment. This is a common fitness function for locomotion, found in e.g. [3, 6, 22, 27].

Common problems with the fitness function relate to the function landscape it produces, and especially how rugged or deceptive it is. A rugged landscape has low correlation between an individual's fitness and its landscape neighbor. A deceptive landscape will have its global optimum in a difficult location, perhaps in such a way that gradual improvement in the population is actually instead leading it away from the true solution [14]. In both cases, the normal operators will not be enough to mitigate these problems, and many have found a fix in maintaining diversity.

Many diversity maintenance approaches are inspired by niching in nature: Individuals in a species might compete mostly with their own species, within their age group, or within their own skillset. This is used to create local selection pressure with global speciation. Note that a measure of similarity between individuals must be specified, which can be measured on individual's genotypes or phenotypes. Though there are many methods for maintaining diversity, we will look at two that are widely used. These measure similarity on the phenotype, and are known as quality diversity methods:

Novelty search is one such approach that discards fitness, and only awards novelty of behavior through evaluating the sparseness of behavior space around an individual. What behavior means must be specified by the user, and in its introductory paper [14], behavior was in a biped locomotion experiment defined as the displacement of center of gravity measured each second. This allowed initially unstable gaits to withstand selection pressure and finally produce oscillatory gaits [14].

Multi-dimensional Archive of Phenotypic Elites (MAP-Elites)

[28] works in a similar way: A number N of discrete features must be set that describes an individual, and every creature will, according to these features, be placed in an N-dimensional map. All creatures in the map will have offspring that will compete with fitness for the cell their features map to, leaving a map of only the best individuals for each cell [28]. Nordmoen et al. shows that MAP-Elites is able to produce quality on the same level as a single-objective EA and a multi-objective EA (optimizing for diversity and fitness) when co-optimizing morphology and control. Additionally, MAP-Elites produced a higher diversity than the other two. This quality can make it more suited to avoid early convergence and local maxima [29].

2.3 Evolutionary Robotics

In Evolutionary Robotics (ER), EAs are applied to evolve robots, in order to automatically develop a robot for some task. This includes optimizing parameters of the robot, optimizing a brain for a human designed body, optimizing a body, or co-optimizing morphology and control. It is commonly done in simulation, even though this leads to a famous problem called the reality gap, discussed below.

A significant advantage of using EAs in robotics, is being able to create robots that are not limited by human ingenuity. In a lot of cases, researchers have found that EAs can create new, novel solutions to a problem that human engineers could not think of themselves, either outcompeting a designed solution as in [30] and [31], or figuring out a complete morphology and control from building blocks that were too complicated for researchers to put together, as in [22].

Additionally, a robot can be evolved for an environment or state that is not fully understood by a human engineer (given that its features can be accurately described in a simulation). Robots can be evolved to adapt to damage, and maybe even self-repair [31]. They can also evolve to adapt to different environments [32]. Given these features, evolved robots can be well suited to explore hazardous areas, where they can adapt to variable terrain and adapt to any damage they experience.

It is also a viable way to study biology through synthetic methodology. Researchers periodically comment on how robotics tends to borrow from biology, but also how ER (and other similar

subfields) is suited to study concepts from biology [33–35].

2.3.1 Co-optimizing morphology and control

Some researchers choose to optimize morphology and control at the same time, meaning that you usually start from only a set of parts, or a set of instructions. EAs will then shape those into some robot, demanding only that you can specify the parts, algorithm, and fitting objective function, therefore requiring comparatively little domain knowledge.

Motivation

When body and brain are allowed to co-evolve, synergies between the two can emerge. An example of this is morphological computation: A sort of pre- and postprocessing the body does that significantly simplifies what the brain must do. Some researchers believe that this can make much smarter designs, with bodies that are naturally easy to control. Paul, in her 2006 paper [36], introduces a tensegrity robot (described below) that shows a very high degree of dynamic coupling, yet proves to be easy to control through evolved controllers.

Additionally, morphological computation can extend the abilities of a control system: As an example, Paul also constructs a robot, the XOR bot, that can display more complex behavior through its body than its brain is capable of. In the XOR bot, the brain is a perceptron, which has been proven to not be able to perform XOR, yet the robot overall has XOR behavior through the dependencies of its mechanical body. In the same way, Bongard argues in his 2013 article [35] that computation can be outsourced to a body through smart morphology, which lessens the needed computation in the robot brain. In short, morphological computation extends or alters the abilities of the brain.

Another reason, though more philosophical, is the notion that such embodied design can let general AI emerge. Some researchers believe that intelligence can be created by continually more and more complex interactions between the environment and an embodied AI [34, 35]. This is grounded in the argument that an intelligent agent can be defined as an agent that is compliant with the rules of its environment (e.g the laws of physics), and is capable of exploiting those rules to diversify its actions [34].

Challenges

In Darwinian evolution, small, incremental steps towards a good solution is encouraged by more self-replicating (the organism lives to reproduce). This is what allows for complex structures to emerge, as initial attempts at something, such as the socket for an eye forming, is not punished but rather rewarded [13]. In our terms, this is a smooth search landscape, a much sought-after feature of artificial evolution [34].

Faina et al. illustrates briefly in their 2013 paper [7] that in co-optimizing morphology and control, a smooth landscape is seldom the case. Rather, they describe that their landscape tends to be deceptive and rugged, in that a mutational step will lead to drastic changes in performance, and not necessarily lead to an optimum.

An additional challenge is that of early convergence of morphology only. As described by Joachimczak et al. [2], and later explored further by Cheney et al. [22], the morphology will reach its almost final form relatively early. As such, the rest of an individual's evolutionary run will be spent on optimizing the controller. While Cheney et al. stresses that this can be attributed to the algorithm or encoding used, as previously discussed by other researchers, they also propose a hypothesis grounded in embodied cognition: Since the control policy interacts with the environment through the morphology, any change to this interface will radically change what action is occurring with a given brain. This is similar to how Paul's XOR bot, with a slightly different body, no longer performs XOR. In essence, variation in the morphology scrambles the control. This, in turn, leads to sudden drops in performance, contributing to the rugged landscape problem described by Faina et al. and Lehman and Stanley [14].

Lastly, researchers in this field express that progress is stagnating [22, 24, 27]. Since Karl Sims first started co-evolving morphology and control in the 90s [1], there has been no results that seem to significantly surpass his. Although there is steady improvement, co-optimized robots are not largely more scalable, complex, or more useful than they were 20 years ago [37].

Lifetime learning

An alternative approach to having to evolve both morphology and control is that of lifetime learning. Here, a robot is given time after its

creation to learn a suitable controller for its morphology, by-passing the problem of scrambled controllers. Some researchers claim this is necessary in all cases of co-optimizing morphology and control (e.g. [38]), but no consensus on this is observed.

Two main branches of lifetime learning are Baldwinian and Lamarckian. In the latter, learned controllers are inherited from parents. This provides a stepping-off point for the learning in the child, leading to faster increase in performance as shown by Jelisavcic et al [38]. In the Baldwinian case, learning is not explicitly inherited. Here, learning leads to better fitness for an individual, which leads to it having children with roughly the same aptitude for learning. Over time, individuals who can achieve higher fitness by learning will be selected for and dominate the population [39]. Gupta et al. demonstrate this in their recent paper [40], claiming that their Baldwinian learning optimizes for morphologies that are easier to control, shown by how later generations find good controllers faster.

2.3.2 Encodings

If you were to use a GA to find the value x that minimizes some function, let's say x squared, then your individuals' representation in your program, its genotype/chromosome, could simply be the value itself. A mutation could be to add or subtract one unit, and the evaluation would be done directly on the representation.

As the individuals that are optimized become more advanced, the need arises for a separation between the representation and the form to evaluate. Specifically, the genotype is transformed into its phenotype with some one-way set of rules called an encoding, which we can express as

$$phenotype = p(g)$$

where g is the genome and p is the process of converting the genotype into the phenotype. The phenotype is then what is evaluated for fitness, which we can express

$$fitness = f(p(g))$$

where f is the function that evaluates the phenotype. While encoding is a necessity, it will also influence the optimization in performance and diversity [27, 41].

In the following subsections, four common approaches to encoding are discussed.

Direct

Direct encoding is a set of one-to-one mappings for some robotic system so that one value in the genotype corresponds to one feature in the phenotype. This is a straightforward approach, which has led to good results in [6,7,42]. For example, Lipson and Pollack [6] used direct encoding in order to evolve simple creatures consisting of bars and actuated bars, with neural networks controlling the actuation. The genotype was a set of numbers corresponding to the parameters of the different modules.

Lipson and Pollack's approach is quite typical for direct encodings and shows a successful approach to it. However, it also illustrates its lack of scalability: When each part of your robot must explicitly be coded for, complexity has an upper bound that corresponds to your memory limitations, and a search landscape that is intractable. Following from this, we look at three encodings below that can be classified as artificial developmental encodings. Inspired by nature's own reuse of genes, they have a limited number of genes that through some system maps to a much more complex phenotype.

Rewriting systems

One such encoding is rewriting systems, or grammatical approaches. These are characterized by having to define some grammar of one-to-one mappings while employing this grammar in a system that allows for recursive re-use of entire parts of the genome. Some examples of this are Hornby et al. who implemented a grammar for an L-system [3], and Sims, who used a directed graph of nodes and connections [1]. Hornby et al. argues that their L-system representation captures "intrinsic properties of the design space" in a modular way. This led to mutation being more likely to lead to fitness increase compared to mutation on a direct encoding. Additionally, they found that an L-system representation took leaps and explored more of the search space [43], as was also found by Veenstra et al. [27].

Generative neural networks

Generative neural networks are neural networks that are evolved to map some input, for example coordinates as in [24], to some output, which can be the presence and parameters of a body part or weight and connection of a neuron. The genotype in this case

are then the parameters for the network itself, and the phenotype will be generated through queries to the network. One example of this is the evolution of Compositional Pattern Producing Networks (CPPNs) [44] through the neuroevolution algorithm NEAT [25] in HyperNEAT [45]. Another is cellular encoding, which grows a neural network node for node from a grammar tree of node construction commands, e.g., split node, cut connection, recurse, etc. Each newly constructed node gets its own copy of the grammar tree and a position to start executing construction commands from [46].

Cell chemistry approaches

Cell chemistry approaches are encodings that simulate the interactions between cells, and how they modulate development from a starting point to the complete phenotype. One such encoding is inspired by gene regulatory networks (GRN) in nature, as described by Joachimczak et al. [47] [48]. Their method has cells that perform cellular division and death through an abstraction of a GRN (a recurrent neural net) and communication between cells modeled after morphogens.

2.3.3 Evolvability

In order to explain evolvability, the definition from the introduction is repeated: Evolvability is a population's ability to produce useful solutions in the future by increasing non-lethal genotypic variation with latent potential for phenotypic diversity. To explain further, evolvability makes a population more evolvable, meaning it can explore more and often reach higher performance. This is accomplished through a population having genotypes that can quickly change to other functional phenotypes, avoiding pitfalls in the fitness landscape [15–18].

Often, genotypes can change in this way because of indirect encodings, where several genotypes can map to the same phenotype [19]. This increases the immediate phenotypic neighborhood of the population, increasing the possibility of a good phenotype being discovered in relatively few steps [15, 18]. Systems like this are said to be phenotypically robust, in that many changes can be applied to the genome without it influencing the phenotype. This allows for changes to the genome to build up and ultimately leads to larger jumps in phenotype space, and thus in the fitness landscape [18].

Another facet of evolvability is that of compartmentation, or modularization, of different co-evolving functions within an individual. For example, the compartmentation of organs in animals allow for local exploration within an individual without lethal failure of the global organization of the animal [16, 19]. In this way, an individual is robust to genotypic changes within itself, which improves evolvability by keeping the optimization of different functions separate [19]. Transferring this to the co-optimization of morphology and control in robots, we can infer that some robots will have morphologies that are robust to mutations in the controller [23], and vice versa.

2.3.4 Control

When co-evolving morphology and control, it is common to have distributed/decentralized control of the body (e.g. [3,7,22,27,38,49]), though this is not always the case (as in [4, 5]). Decentralized control is when a body part largely controls itself: A leg will for example have its own behavior that through just time, or with input, changes and affects the entire body's behavior. As opposed to this, centralized control takes in all sensor inputs and controls the entire body at once.

Centralized control

Centralized control allows for central processing of sensor information and synchronized action output. It is therefore well suited to task execution. It is also quite common to see centralized control in nature (animals are a good example), albeit with some decentralized control present as well (for example central pattern generators (CPGs) being responsible for many activities in animal behavior [33]). When co-optimizing morphology and control, it can be difficult to have centralized control because of the changing number of inputs and outputs. However, this does not mean it is not done.

Cheney et al. present one such co-optimized system in his work on electrophysiological soft robots [4]. Here, robots consist of voxels placed by a CPPN encoding. These voxels conduct electricity that is induced by one pacemaker voxel. Although this is simple and open-loop, it shows a form of centralization of control that does not use a neural network.

Auerbach and Bongard also use a CPPN encoding, that constructs a control network, a continuous time recurrent neural net-

work (CTRNN), along with the body [5]. Here, the CPPN encoding outputs body parts as well the input and output to control that part. In this way, the centralized control network can grow with the morphology.

In the Baldwinian lifetime learning of Gupta et al., the controller is erased every generation, and in essence only the morphology is optimized [40]. Here, they use a centralized control network with no difficulty because the morphology is finalized when the network is initialized. They then train the network with reinforcement learning and use the individuals final fitness from this in selection of the next generation of morphologies.

Distributed/decentralized control

Rodney Brooks was one of the first to employ a form of distributed control. In his 1986 paper, he introduced what he called a subsumption architecture, where he suggested thinking about the control system as a series of behaviors rather than a series of modules. For example, instead of leg 1, sensor 1, etc., he wanted to describe the control system in terms of object avoidance, walking, and so on. In this way, control is functional at each level, allowing for removal or addition of a level at any time. In his own words, "The control system can be viewed as a system of agents each busy with their own solipsist world" [50].

The later distributed controls would return to module-centered control in a sense, now in the form of appendages and modules controlling themselves. In the same way as Brooks, behavior was modularized and decentralized, though now the other functions of a creature were typically not fully preserved when adding or removing behavior. Sims' virtual creatures of 1994 is a famous example of this, where each appendage would have a set of neurons controlling itself [1]. Another person to do this was Stoy, in 2002, who created control for the modules of a modular robot that would work in any morphology: The behavior would change according to the number of connections to other modules [51].

Some recurring and current methods of distributed control are open-loop wave generators, CPGs and neural network based. They will be described below.

Open-loop wave generators

Open-loop wave generators are a way to produce repeating motion through cyclic functions, commonly a sine wave with evolved parameters [3, 7, 41]. They are open-loop in that their only input tends to be time passed, and the behavior is not affected by anything that happens to the robot (excluding the passage of time). Examples of this are Hornby et al in 2001, that produced complex virtual creatures with each actuated joint being controlled separately by a simple oscillator. Because it does not adapt to the environment, some researchers use it more as a simplification as they produce the rest of the system, rather than the final control system (e.g. [3,7]).

CPGs

Central Pattern Generator (CPG) is a neural circuit found in living organisms that produce rhythmic patterns [33]. For example, they are typically responsible for cyclic activities like walking and chewing. In some studied organisms, the CPGs appear to function independently, but can be modulated by neural or sensory input [33]. Because of this, CPGs can be seen as a more complex alternative to open-loop wave generators when modeled for robotic systems. In addition, Ijspeert identifies some beneficial properties of the CPG, amongst which are that it is well suited for distributed control of for example modular robots. This has been done in the Roombots [52]. Furthermore, the CPG is robust and can receive drive signals that have low dimensionality and produce high dimensional output [33].

Neural networks

Distributed control using neural networks will involve giving each part of some robot its own neural network, whose weights, topology, or activation functions can be evolved. Here, any type of network can be used, and we have many examples of these giving good and complex behaviors, as for example Sims' virtual creatures [1], the soft robots of Rieffel et al. [53], and the bar-robots of Lipson and Pollack [6].

2.3.5 CTRNNs

Continuous time-recurrent neural networks (CTRNNs) will be explained in further detail, because this is the network that will be

used for the controllers in this thesis. They can be used as both centralized and decentralized control, both of which we do in our experiments.

CTRNNs are simple neural network models that allow for memory through recurrent connections. They update based on time passed, sensor input, and internal state; and do this through a series of ordinary differential equations.

In particular, the change in neuron potential of neuron i, y_i, after neuron i's time constant τ_i is modeled as the differential equation

$$\tau_i \frac{dy_i}{dt} = -y_i + f_i\Big(\beta_i + \sum_{j \in A_i} w_{ij} y_j\Big)$$

where f_i is the activation function for neuron i, β_i is the bias of neuron i, A_i is the set of indices of all input neurons to neuron i, and w_{ij} is the weight for the connection between neuron i and neuron j. Each neuron potential is updated with the forward Euler method, which essentially means that the above calculated value will be added to the previous neuron potential for neuron i when going from time t to time $t + \tau_i$. This formula and its explanation is taken from neat-python's description of their CTRNN[1].

The CTRNN has several properties that makes it suited for control of biologically inspired robots. They allow for the possibility of dynamic temporal behavior [54] and can in theory exhibit any dynamic behavior [55]. They also have possible interpretations as analogies to biological neural networks [55, 56].

As mentioned, CTRNNs were successfully used by Auerbach and Bongard when co-optimizing morphology and control in their robots [5]. It was also used by Randall Beer and his various co-authors in a series of works related to the goal of investigating a theory on adaptive behavior [55, 56]. As they viewed CTRNNs and their own surrounding system through the lens of dynamical systems theory, it was also important to them that the CTRNN's behavior can be analyzed. In their work on novelty search, Lehman and Stanley used a CTRNN in their bipedal walker experiments [57]. They reasoned that it was suitable for locomotion because its non-linear dynamics was often found in gaits from nature.

[1] https://neat-python.readthedocs.io/en/latest/ctrnn.html

2.3.6 Material

Robots in ER have been built or simulated in many materials, which have different effects on the control and morphology. Probably most common is rigid, skeletal robots with actuation and movement in the joints only (e.g. [1, 3, 14, 27, 41]), which also has many examples of having been built in real life (e.g. [6, 7, 32, 42, 58]). These reflect what kind of robots have historically been built up until now. However, this limitation of material might be part of current observed stagnation in co-optimizing morphology and control [24]. Thus, we look at two alternatives below: Tensegrity and soft robots.

Tensegrity robots are built from tensegrity (tensile integrity) structures but include actuated pistons instead of cables and/or struts. These display a high degree of dynamic coupling, and can therefore do much morphological computation, as shown by Paul [36]. They are also robust and have a high strength to weight ratio. In Nasa's SUPERball tensegrity robot project, it is hypothesized that this quality will make it suitable for planetary exploration, as it can locomote over dangerous terrain, be robust to unexpected disturbances, and possibly need less landing equipment [59]. SUPERball is hand designed and uses a human designed controller. However, Paul argues and demonstrates that because of the high dynamic coupling, machine learning (and in her case, EAs) are well suited to find control for tensegrity robots.

In a very different vein, soft robots are constructed with soft materials. These can be entirely made of soft tissue as in Joachimczak's soft bodied animats [47, 48], or with both actuated, soft and hard tissue as in Cheney et al.'s voxel-based creatures [22, 24, 60]. In the 2013 Cheney et al. paper [24], the use of soft robots is motivated by similarity to nature, where organisms tend to have both rigid tissue like bone, and soft and/or actuated tissue like muscles and fat. These robotic systems are also very malleable and could prove themselves to be especially suitable for tight spaces and difficult terrain. However, controlling them is a definite challenge, as their soft bodies lead to "near-infinite degrees of freedom" according to Rieffel et al [53]. EAs are therefore very applicable to soft robots [53].

2.3.7 Simulators and physics engines

When choosing a simulator for your experiments, you must consider many things. Not all simulators can simulate the robot you want to

use, and not all sensors are supported. You also must consider if you want rigidbody dynamics, soft body dynamics, fluid dynamics, and so on. For some, the integration of ROS, Robot Operating System, is also quite important to have quick prototyping and deployment of robots in real life [61].

Some common physics engines are PhysX[2], ODE[3], Newton dynamics[4], and Bullet physics[5]. All provide rigidbody dynamics, while only PhysX and Bullet supports soft body dynamics [61].

A common simulator to use is CoppeliaSim, that is the successor to V-Rep. It supports programming in multiple languages and with ROS, and the use of multiple physics engines like Bullet Physics, ODE, Newton and Vortex Dynamics[6].

Mujoco is another much used simulator, recently acquired by DeepMind and used for their DeepMind Control Suite [62]. It does not allow for using different physics engines, however its own physics engine is capable, providing speed and accuracy[7]. It also compares favorably to physics engines like PhysX, ODE, Bullet and Havok when it comes to accuracy and speed on robotic tasks [63].

Perhaps the most used simulator for development of robotics is Gazebo [61]. It describes itself as a collection of libraries, and allow for a lot of customizability in terms of physics engines and other functions[8]. Especially Gazebo's ROS integration makes it a popular alternative [61].

2.3.8 Reality gap

Some ER experiments are done in simulation first, and then they are later transferred to reality. This often leads to a disparity in performance between the simulated and physical robot, which is called "the reality gap". This is caused by some difference between the simulated system and the real one, for example lack of noise or inaccurate physics. Consequently, when the optimization exploits features in the simulation, the solution fails in reality where those features are not present [35]. Samuelsen et al. shows how the gravity of this gap affects robots very differently, with performance of five

[2]https://developer.nvidia.com/physx-sdk
[3]https://www.ode.org/
[4]http://newtondynamics.com/
[5]https://pybullet.org/wordpress/
[6]https://www.coppeliarobotics.com/features
[7]https://mujoco.org/
[8]https://gazebosim.org/features

evolved robots either staying roughly the same or dropping as much as 90% [58].

Although the reality gap could be avoided by doing evolution in real life, doing so can be very time-consuming, not possible in cases like Jochimczak's developing animats, and/or induce much stress on the robotic platform [64]. In their paper on the physical robotic platform Dyret, Nygaard et al expresses that Dyret must be built very robust to withstand continued real-life trials. Easy maintenance must also be a focus, because it will eventually still break [32]. Veenstra et al. [30] is another example of a successful application of EAs in real life, where swimming in a robotic knifefish was found using an evolutionary strategy. However, they were still constrained by stress on the platform, causing them to opt for a faster optimization algorithm to minimize overheating of the servos.

Solutions

One early solution to the reality gap was the addition of noise to the simulated sensors in a robot. Jakobi et al. showed that a realistic level of noise would result in similar behavior between a simulated and real Khepera robot [65].

Another branch of solutions is concerned with sampling reality and using this in simulation. One method samples different readings for the real life robot, and uses it in a look-up table while optimizing. Though it works well, it scales poorly [66]. Yet another method co-evolves simulation parameters and the robot, focusing both on optimizing the robot and reducing the difference between the simulated and real performance. It does this by recording each best individual of a generation in real life and evolving the simulation to minimize the difference between recording and simulation [67].

Koos et al. saw evolving good behavior and crossing the reality gap successfully as opposing goals [68]. They therefore proposed a pareto-based multi-objective evolution of robots, where a robot was measured on the objective goal, but also on a simulation-to-reality disparity measure (STR). The STR measure is a model that can predict the STR disparity based on previous recordings of controllers on the physical robot [68].

Feasibility

Another approach to crossing the reality gap is designing for feasibility. This means to choose methods and algorithms that have

been shown to cross over better.

Faina et al., among others, suggest modular robots for having feasible manufacture of co-optimizing morphology and control: If you feed your EA parts that are already feasible, the following design and control will likely also be doable [7]. Additionally, modular robots allow for easier testing of several different solutions because they can be reconfigured. Because of this, versions that cross over to reality better can be found and used.

2.4 Modular Robotics

Modular Robotics (MR) concerns robots built from separable modules or units that encapsulate some function of a robotic system. This is as opposed to an integrated design with no clear sectional modularity. The modules contain actuation, computation, energy, and sensing as needed, as well as some mechanism to connect and transmit to other modules [69].

There are three key reasons to choose a modular robotic system: Low cost, robustness, and versatility. These are presented below.

The simplest reason is low cost: Because a system is usually made of only a few types of modules, they can be mass-produced leading to low cost per module. However, modules usually have redundant parts, so it is not self-evident that this benefit will be present in real systems.

These systems can also show robustness because they typically have high redundancy. Additionally, the promise of self-repair could mean that a robot can replace or expel a faulty module on its own [69], or with the help of a human. However, modular robots run the risk of lacking robustness if their control and between-module communication depends on a chain of working modules [11].

Finally, modular robots can be versatile, meaning that they can take many forms to best adapt to a task. Ideally, this would be done on their own through self-reconfiguration. Ease of reconfiguration by hand also allows for quick prototyping and reality gap experimentation.

In their 2000 paper on the modular robot PolyBot, Yim et al. presents these three benefits, and claims PolyBot proves modular robots to be versatile [11]. 19 years later, in a review written by among others Yim, they write that this is still the only claim shown to be true of modular robots, as modules built by researchers tend to

be expensive and lack robustness [69].

2.4.1 Classification

When a modular robot is presented, these are the classifications usually mentioned: Architecture, system abilities, and module homogeneity. These will be explained below to have a better understanding of how people describe modular robots, as well as some of the capabilities usually associated with MR.

Architecture

Modular robots can be connected in a lattice-, chain-type, or a hybrid of these two, as well as mobile and truss architecture. Chain-type modules typically have less connection sites, commonly two as in the CONRO [51]. This makes their bodies chain- or tree-like, and often not space-filling. Lattice-types have more connection sites than two, and often more than one degree of freedom, and make up a 3D configuration. Hybrids have features of both, for example the M-TRAN which can be configured in both chain- and lattice-type structures with its six connection sites per module [70, 71]. A mobile type can fully disconnect from each other and move around autonomously, as well as form chain- and lattice-type robots, and possibly smaller sub-robots. Lastly, a truss architecture consists of members and nodes [69].

System abilities

Modular robots show promise to be highly autonomous, with researchers theorizing about self-assembly, self-repair, self- reconfiguration, and even self-reproduction [72]. Because of the ease of mass-manufacture and connecting and disconnecting modules, modular robots are uniquely suited for these functions.

The most implemented ability is self-reconfiguration [71, 72], where a modular robot is able to change its configuration without human intervention, as in the M-TRAN [70] and PolyBot [11]. In his paper on the ATRON self-reconfigurable modular robot, Christensen explains that focusing on achieving a specific shape can sometimes be impossible for modules such as the ATRON. Mechanical constraints in the modules make reaching some positions not feasible. He proposes instead a system based on attractors, where the

modules will configure into a target function rather than a target shape [31].

Self-repair means to be able to identify and fix a faulty part of a system, be it a faulty module or other damage. In Christensen's ATRON, self-repair comes about as a consequence of self-reconfiguration: The modules have distributed control that seeks to inhibit attractor states, which lead to them forming a configuration. If modules are removed, simulating damage, the attractor states are no longer inhibited and surrounding modules flood in to repair the damage autonomously [31].

Self-assembly is for the modular robot to either have such properties that it can assemble itself when put together, or otherwise is assembled by an autonomous system. M-Blocks show an ability to locomote independently using an internal fly-wheel and has algorithms for finding and assembling at a site [73].

Self-reproduction is for a modular robot to be able to replicate itself, or to produce another robot. Among the system abilities mentioned, self-reproduction in particular is the least demonstrated [72], with the minimal example of Zykov et al. [74] being one of the few we've seen.

Homogeneity

When it comes to module homogeneity, a modular robot system is said to be homogeneous if all its modules are the same, and heterogeneous, or n-modular [11], if there are more than one type of module present. For example, M-TRAN is a homogeneous modular system which has one part, two semi-cylindrical boxes connected by a hinge [70]. For a heterogeneous example, the Microtub system features a total of five modules, which are a camera, rotation, support, extension, and helicoidal module [75].

The key difference between homogeneous and heterogeneous modular systems is redundancy. Because homogeneous systems have only one type of module, that module must contain the sensing, actuation, and computation needed for the entire robot, leading to high redundancy. This is a strength, in that it can easily self-reconfigure, be repaired or self-repair from a supply of that module. It is also a liability, because every functionality is typically not used everywhere in the body, causing the maker to have to pay for parts that are not used [34]. Heterogeneous modular robots do not have this problem, as modules can be designed with only one

functionality and used as needed, leading to no redundancy if that is wanted.

2.4.2 The modules

The modules in real life modular systems tend to be small, compact, and feature clever mechanical solutions. They pose an interesting engineering challenge: As a part of a whole robot, they should supply actuation, sensing, and computation while remaining as compact as possible. In addition, they need capabilities to attach and stay connected to other modules. However, this need for many functionalities can often lead to feature creep and larger modules. Below, these key implementation features and challenges are presented.

Actuators

Modules often contain a type of actuation, with the exceptions possibly being modules from heterogeneous systems, like a passive base, and remote actuation sources like electrical fields [69]. Among homogeneous robots, a popular choice is DC motors [71] providing rotation between two parts of the module, as in the M-TRAN [70] and Roombots [52]. Other choices are stepper- and servomotors, pneumatic actuators, as well as several novel actuation methods, like the electromagnetic inertia actuation in the M-Blocks.

Sensors and communication

There can be internal and external sensors in a module. Internal sensors are used to monitor the internal state, like the rotation of the actuators, e.g., with Hall-effect sensors, or the module orientation through accelerometers. Sensors to help with docking can also be beneficial, like LEDs and photosensors, as the inverse kinematics involved in self-reconfiguration can be inaccurate. External sensors are used to sense other modules and the environment, the latter of which is paramount for performing tasks [69].

Communication between modules can be facilitated through a bus over the connection plate, though this requires a connection. Wireless options allow for communication without connection, which can allow for more autonomy in the system, and possibly more distributed control [69]. Among wireless communication is infrared, Bluetooth, and Wi-Fi [71].

Connection mechanism

The connection mechanism, or docking element, is the part of a module that allows it to connect to another module. Many modules use a concept of gender, where a female element can only connect to a male element and vice versa, as with polarities of magnets. When gender is not used, the genderless docking element has the added benefit of allowing more configurations [71].

Connection by magnets is widely used as opposed to other methods [71], and can be divided into subcategories of permanent and electromagnetic magnets.

In the case of permanent magnets, they hold the advantage that a connection does not consume energy. This also means that there must be a means to disconnect two elements autonomously if the system is to self-reconfigure/repair/assemble. One solution to this are the M-Blocks system simply using the force it moves by to break away. It also avoids gendered elements, as magnets usually call for, by allowing its magnets to rotate [73].

Electromagnets function in much the same way, except that power is needed to maintain connection. Advantages of this is easy disconnecting and being able to change the polarity/gender of the magnet, receiving the same benefits as genderless elements.

Latches, or hooks and holes, are also used. These have gendered docking elements, with hooks that come out, driven by a motor, and latches onto holes on the female connection plate [71]. Latches have the drawback that they are often weak to misalignment [73].

Challenges

A goal in modular robotics is having smaller, more compact modules, both because some envision robots built from cell-like modules (e.g. [9, 34, 76]), and because it gives better general applicability to different task spaces [69]. The challenges here are several, the top of which is the current technology in actuators and power sources, which are the main determiners of size [71]. Another issue is that the smaller modules get, the amount you need to fill the same space is cubed. As modules increase, stiffness and strength of the overall robot goes down, since every link can be a potential weakness. Solutions for this include designing for parallel connections, as this provides strength that a chain type robot will not have [69].

For homogeneous robots especially, there is the issue of feature creep. This happens when too many functionalities are added to a module, leading to expensive modules with a lack of reliability. Possible solutions to this is to better think of the congregated robot as a whole when equipping modules, and not adding more functionalities than is needed for the function of the ensemble [69].

2.4.3 Control

As with ER, modular robots can be controlled in any of the centralized or decentralized ways listed above. Within homogeneous modular systems, all modules tend to carry computational elements, which means that the amount of computation available scales with the number of modules. This can make control difficult, but a lot of researchers have found a solution in distributed control [69].

Nordmoen et al. [29] and Faina et al. [7] both use open-loop wave generators. Sproewitz et al. uses CPGs in every module to make their Roombots locomote [52]. Christensen uses modules with distributed evolved artificial neural networks (ANNs) for the ATRON, that automatically congregate in specified configurations [31].

Centralized control might be a more practical approach when it comes to tasks like object manipulation, or the combination of several behaviors [69]. Self-reconfiguration has for example been demonstrated to work well when orchestrated by a centralized control such as for the M-TRAN [70]. Here, the robot was able to reach a specified configuration through controlled maneuvers, which contrasts the imperfect swarm-like behavior of Christensen's ATRON-controller.

For a combined example, the Microtub modular robot employs a semi-distributed approach. Here, an overseer centralized system processes the sensory input of the modules and sends out corresponding commands. At the same time, the modules can react autonomously to stimuli like overheating. These behaviors function independently of the centralized control [75], very similar to Brooks subsumption architecture [50] explained above.

2.4.4 The use of EA in modular robotics

EAs can be a powerful tool to find control algorithms and module configurations for modular robots. As module numbers grow, both

Sys. name	Paper	Year	Ctrl	Config	Co-opt.
CEBOT	Kawauchi et al. [9]	1992	EA		
Fracta	Murata et al. [10]	1994			
PolyBot	Yim et al. [11]	2000			
Crystalline	Rus et al. [77]	2001			
CONRO	Stoy et al. [51]	2002			
M-TRAN	Murata et al. [70]	2002			
Genobot	Hornby et al. [43]	2003	EA	EA	x
Adam	Marbach et al. [78]	2004	EA	EA	x
ATRON	Christensen [31]	2006	GA		
Swarmbot	Groß et al. [79]	2006	SI & EC		
RoomBot	Sproewitz et al. [52]	2009			
SMORES	Davey et al. [80]	2012			
Microtub	Brunete et al. [75]	2012			
M-Blocks	Romanishin et al. [73]	2013			
EDHMoR	Faina et al. [7]	2013	EA	EA	x
	Zappetti et al. [76]	2017			
EMeRGE	Liu et al. [20]	2017	EA	EA	x
Sambot II	Tan et al. [81]	2018			
SMORES	Tosun et al. [82]	2018			
RoboGen	Jelisavcic et al. [38]	2019	EA	EA	
	Nordmoen et al. [29]	2020	EA	EA	x
EMeRGE	Moreno et al. [21]	2020	EA	EA	x

Table 2.1: **A categorization of optimization-use in MR-papers.** System name is either the name of the modular robot used, or the system for developing modular robots. Ctrl: evolved control, Config: evolved configuration, Co-opt.: Co-optimized control and configuration.

of these tasks can be daunting for a human designer, and even in manageable systems a solution found by EA can outperform a human-designed controller, as in [31].

In order to get an overview of the use of EA in MR, notable papers frequently mentioned in recent papers and MR surveys ([69,71,72]), as well as papers from this reference list have been chronologically categorized in table 1. The latest surveys are from 2019, meaning the development from 2019 to 2022 is likely not covered well.

As we can see from the table, early modular robots tended to be fully hand-designed, although controllers were sometimes optimized. This is still a trend. Hornby et al. provided the first example of co-evolving simple, fixed modules of rods and actuators [43], while Marbach et al. co-evolved a more typical modular system a year after [78].

Later researchers have followed up on this work, however there still has not been much work done on evolving modules. Liu et

al. and Moreno et al. both found that different variations on the module morphology had a significant impact on the evolvability and performance of evolved configuration and control [20, 21]. This shows that evolving modules could further aid in the design of modular robots.

2.5 Concluding statement

The above has been an overview of various topics related to co-optimizing modular robot systems.

During ER and MRs short lifetime, a lot has been accomplished. The idea to co-optimize and the first modular systems came to be in the 90s. Thereafter, many researchers expanded on these ideas with larger projects, more elaborate developmental stages, as well as combining ER and MR. The problem of the reality gap came to be more researched, and researchers started to worry about diversity maintenance. In the 2010s, quality diversity algorithms were popularized. Co-optimization in general, but also specifically applied to MR, gained more traction.

At this point in time, the field is experiencing some stagnation. Many are investigating what role materials, encodings, and algorithms could play in this. Whatever the solution might be, hopefully we will work through this stagnation, and something as impressive as Sims' 94 work will soon be presented in these fields.

Chapter 3

Implementation

Here the implementation of the system we will use to test the hypotheses is presented. We exclude the implementations of the controllers and encodings, as these will be discussed in the chapters about these experiments respectively.

First, we give a general overview of the system, then we present the software and tools, which are the frameworks we will use. Lastly, we present the further implementation details, like robot representation, evolutionary algorithms, and fitness function.

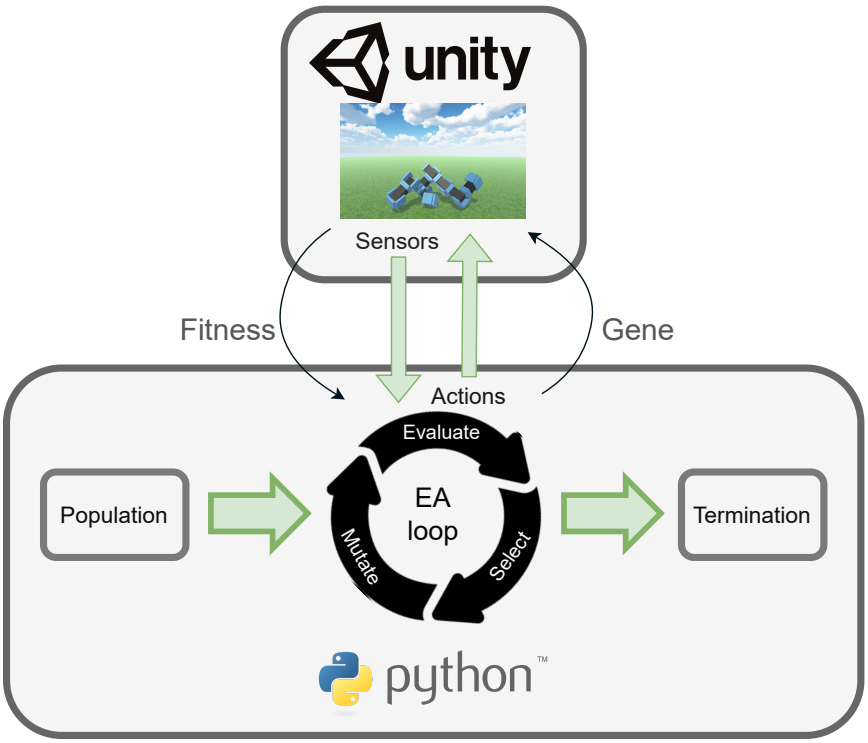


Figure 3.1: **Overview of the system.** The bottom box represents the Python program, hosting the evolutionary loop, and the top box represents the Unity executable, responsible for the simulation.

3.1 Overview of the system

Our system consists of modular robots simulated in a flat ground environment and being measured on the task of locomotion. To co-optimize the morphology and control, an evolutionary algorithm is used.

The general setup with the simulator can be seen in Figure 3.1. The simulator used is a Unity executable, which is launched and communicated with through Python scripts. In order to do this, the Unity ML-Agents Toolkit is used, explained below.

The Python side keeps a population of genes, which are optimized in an evolutionary loop. When it comes to evaluation, each robot gene builds a blueprint of its body, which is sent to the Unity executable. Here, the blueprint is built into its phenotype, a modular robot configuration using the EMeRGE modules (Figure 3.2). Then, during the evaluation, the Unity executable continually sends the robot observations to the Python side, whereupon Python responds with the controller actions. Lastly, Unity sends the final fitness of the individual, and the evaluation section is over.

The software versions used can be seen in Table 3.1, and the code for this thesis can be found on GitHub[1].

Software	Version
Unity ML-Agents	1.0.7
mlagent_envs	0.27.0

Table 3.1: **The software versions used.**

3.2 Software and tools

3.2.1 Unity

Unity is a widely used and general-purpose game development platform. It provides easy development through its graphical user interface called the Unity Editor. It comes with the Nvidia PhysX physics engine, but the Havok, MuJoCo, and Bullet physics engines can also be used. In this project, PhysX is used. It runs in real-time, allowing changes to be made to the environment and objects in runtime. Because of its inbuilt realistic rendering engine and other available sensory data, as well as its available game functionality, it

[1] https://github.com/UiO-Robotics-and-Intelligent-Systems/master_mkkvalsu

can be used for a wide range of machine learning tasks defined by the developer. To use the Unity Editor to develop environments, the Unity ML-Agents Toolkit is used [83].

3.2.2 ML-Agents

The Unity ML-Agents Toolkit[2] is an open-source project that enables easy integration of artificial intelligence in Unity. It is equally aimed at game developers and AI researchers, with many features to support reinforcement learning, neuroevolution, and other machine learning algorithms.

The ML-Agents Toolkit provides a Python API through the two packages mlagents_envs and mlagents-learn. Because we only used the mlagents_envs package, we will explain this one further. It contains the class UnityEnvironment, which allows you to load and use your Unity executable learning environment in Python scripts. This class supplies a typical gym interface, with methods to reset and get observations, and a step method that will move the simulation forward. The step frequency was set to step 5 times a second, meaning sensor info and control information was exchanged every 0.2 seconds. The environment can be run in headless mode, which may cause a decrease in simulation time due to not having to render, or the desired speed up can be specified [83]. However, when speeding up our simulations, we lost determinism in evaluations on the same robot. Because of this, we regrettably ran our experiments with no speed up. We also experienced no significant decrease in simulation time from running in headless mode. In addition to the UnityEnvironment, the side channel classes EnvironmentParametersChannel and SideChannel, with accompanying functionalities and classes, allowed for further communication with the Unity executable. In this way, we were able to send our robot gene to Unity through a custom defined side channel.

In order to design an environment using ML-Agents in the Unity Editor, a scene has to include an instance of the Academy class and an object with an Agent script. The Agent script will receive actions and rewards from the Python API [83]. Because ML-Agents is not specifically meant for co-optimizing morphology and control in robots, we did have to make some adjustments to the way certain classes were meant to be used. Specifically, the Agent class is meant

[2]https://github.com/Unity-Technologies/ml-agents/tree/release_17_docs

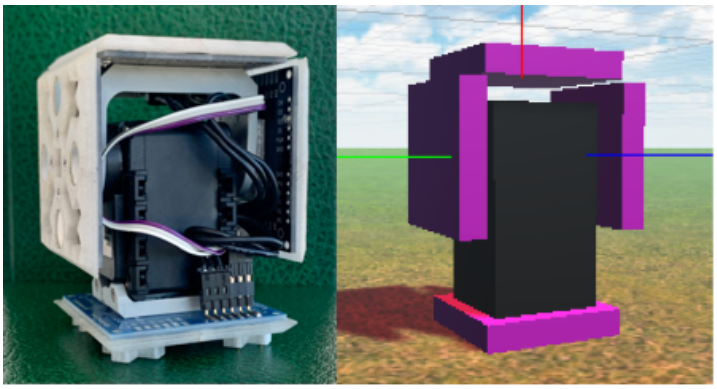


Figure 3.2: **The EMeRGE module.** The left image is an example of a real EMeRGE module, and the right is our Unity simplification.

to represent the agent object, but instead we used the Agent class as a spawner and communicator for our modular robots. We found that this allowed us a lot of freedom over the robot morphology, as we could spawn the entire robot and adjust it during runtime while still having deterministic physics. We were also allowed freedom over the physics, being able to freeze and restart it, as well as adjust the physics parameters at any point. The physics updates every 0.02 seconds, but this can be adjusted.

3.2.3 The modules

We are using the EMeRGE module [84], see Figure 3.2. It is a simple module built around one servo motor, so that each module works like a hinge, and is 8.1 cm tall and circa 6 cm in width and length. We limited the movement of the servo to be +/-90 degrees from the neutral position depicted in the figure. It has four connection faces, one male at the base, and three female on the top and sides. The male connection face can only connect to the female ones and vice versa, meaning the robot will have a root module with three possible child modules, growing outwards in a tree-like structure. The modules can be rotated at any angle relative to its parent, and so morphologies can grow in 3 dimensions. Example morphologies can be seen in Figure 3.3.

In Unity, the modules were modeled as two separate compound colliders, with the three female plates being one collider, and the male plate and servo box being the other collider. Each collider weighed 5 kilos, and together the module weighed 10 kilos and took up circa 1 cubic meter. All measurements were scaled up

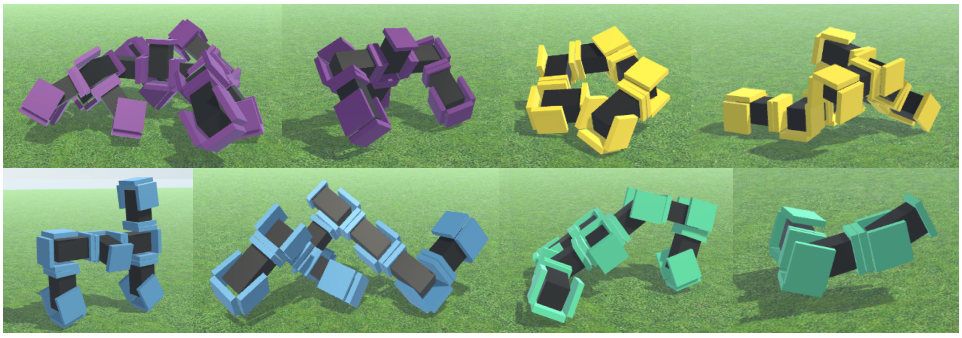


Figure 3.3: **Examples of well-performing morphologies.** Purple and top left: Copy controller. Yellow and top right: Sine controller. Blue and bottom left: Decentralized CTRNN controller. Teal and bottom right: Centralized CTRNN controller.

because this is common among users when using small physics objects in Unity. On many forums such as Unity Answers and Unity Forum, users report strange physics behavior in small objects, likely due to accumulating floating point errors as Unity still uses 32-bit accuracy as opposed to 64-bit accuracy. While we should have tested the system in proper scale and made a comparison, due to time constraints we opted for scaling them up from the start.

The colliders were both physics Rigidbodies and were connected through two ConfigurableJoint components configured to work like a hinge. To control the joint target position, the corresponding JointDrive had to be configured, which meant some manual tuning as the robot would either have too much force, or not be able to move at all. Finally, the JointDrive spring value was set to 200, while the damper was set to 5, and the maximum force was left to its default of the max 32-bit floating point number, 3.402823e+38. Lastly, the angular drag of the female connection plate collider was set from 0.05 to 0 to not slow its rotation. Other values were left at default.

The original design for the modules includes infrared proximity sensors on each face. This was also implemented in our abstraction of the module, although the workings of the exact sensor model were not replicated. Instead, sensors here register all distances through a ray cast, meaning the distances it can register are not capped and could get very high. For not registering a distance, for example from being angled towards the sky, a sensor returns -1.

If a module occupies the connection site, the sensor will pick up the small distance towards the module. This was kept because the modules will move a little in relation to each other, which a controller

might use to detect the presence and behavior of child modules. In theory, this will allow controllers to change their behavior based on sensing if they are a leaf module or not. In a system where connections between modules can break, this could also be used by the controller to avoid losing modules, but this is not done in this system.

3.3 Implementation

In this section, general implementation details are given for the system. In the case of controllers and the experimental encodings, these are presented in respectively experiment 1 and 2, where they are relevant. The different encodings will not be used in experiment 1, but all controllers will be used in experiment 2, meaning they should be read sequentially.

Below, we go through the robot representation and construction, evolutionary algorithm, and fitness function.

3.3.1 Robot representation

Initialization

The robots are represented as directed trees of nodes in a direct encoding. Nodes correspond directly to modules in the phenotype. Constructing a random robot consists of adding child nodes with a diminishing probability based on the depth. Specifically, the presence of a child node with current depth d and overall max depth D is determined by the Boolean expression

$$\begin{aligned} \text{Add Node} = \text{normal}(\mu, \sigma) &< \frac{D-d-1}{D} \\ = \frac{\exp^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}}{\sigma\sqrt{2\pi}} &< \frac{D-d-1}{D} \end{aligned}$$

where σ and μ are parameters that determine the normal distribution, and x is a random number from a uniform distribution between 0 and 1. As d gets higher, the probability of adding a node gets smaller.

Note that the μ and σ does not correspond directly to the mean and standard deviation of the distribution of robot size, however they give some control over the mean and spread. As seen in

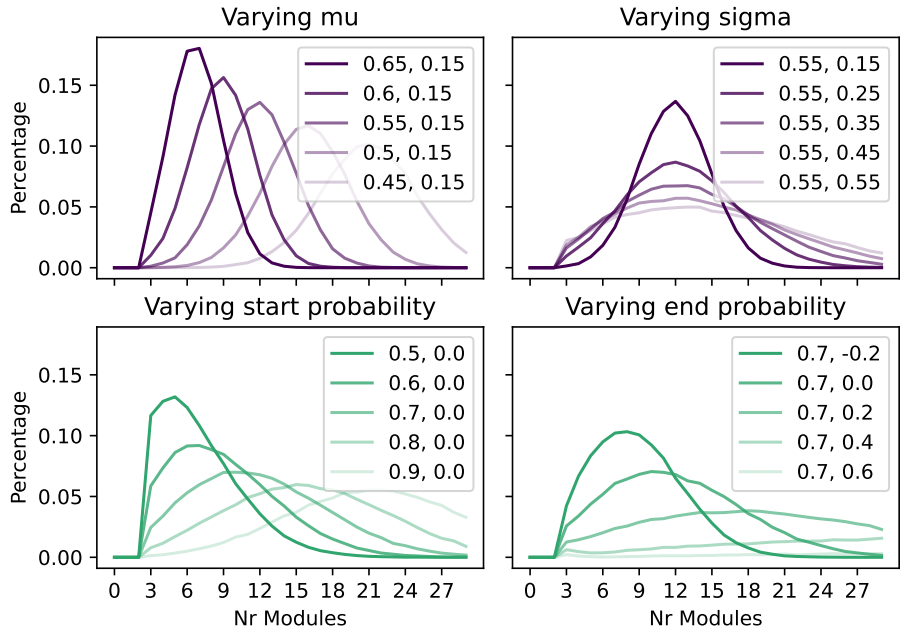


Figure 3.4: **Distributions of robot size in populations when varying initialization parameters.** Robot size is measured as number of nodes, and the y-axis shows the percentage of robots in a population with this number of nodes. Top row shows the non-linear decrease using a normal distribution, bottom row shows the linear decrease.

Figure 3.4, varying the μ gives control of the mean, while also affecting the spread, and varying σ only affects spread. Additionally, at initialization a robot is forced to have 3 nodes or more. If it has less, the robot is reinitialized.

This method of initializing robots gives a similar distribution of number of nodes in individuals when compared to a linear decrease in addition probability until the max depth is reached. An example of this could be the boolean function

$$\text{Add Module} = x < s - (s - e)\frac{d}{D - 1}$$

where x is a random number from a uniform distribution between 0 and 1, s is the start probability, e is the end probability, and d is depth and D is the overall depth. In short, this function linearly interpolates between the start and end probability using the current depth.

These two approaches, with different parameters for μ and σ and s and e, gives distributions as seen in Figure 3.4. While both approaches could have been used, the non-linear decrease using a normal distribution gives control over the mean and spread while

having an even, normal distribution, thus that was chosen. Different values for μ and σ were tested, and in the end the distribution seen in Figure 3.5 was chosen, with $\mu = 0.75$ and $\sigma = 0.35$.

Mutation

A direct encoding was chosen in our system when keeping in mind our experiments. Although good indirect encodings offer many benefits, like increased search space exploration [27, 43], we considered it appropriate to use a direct encoding. This makes it easier to manipulate mutation to be more gradual, as we will later do in experiment 2. In addition, a part of the goal in experiment 1 is to test the effects of duplicating control, and so our baseline controllers should not also duplicate through an indirect encoding.

Because of this use of a direct encoding, mutation is done on each node in the directed tree using a breadth first approach. Each node has four different mutation possibilities:

1. Angle: Randomly switch the angle of the module relative to its parent. Possibilities are 0, 90, 180, and 270 degrees.
2. Remove module: Remove one random child node. This can remove an entire branch.
3. Add module: Add one child node in a random connection site at a random angle. The child will as far as possible be initialized with its parent's control parameters.
4. Copy branch: If there are both occupied and unoccupied connection sites, one child branch is copied over to another connection site. This mutation was chosen to facilitate symmetry and large jumps in the search landscape, but is quite volatile because it can change the robot solution very drastically.

Each node's mutation probability p is given by the function

$$p = P/size$$

where P is the global morphology mutation rate, and *size* is the genome's number of nodes. Dividing the probability is done to not have larger creatures mutate more than smaller ones. If they did, larger creatures would be unstable solutions, quickly changing and disappearing from the search space. When dividing it, the morphology mutation rate corresponds to the chance of a creature mutating and ensures all creatures will mutate the same amount.

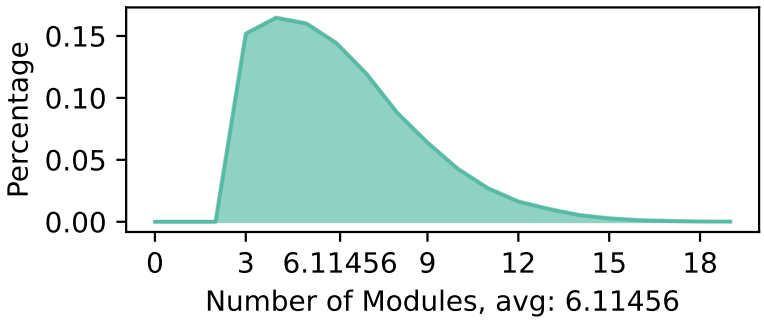


Figure 3.5: **The distribution of number of modules in individuals in a random population.**

We wanted to encourage robots to grow; with more modules, robots can have diverse solutions with more actuation and power, meaning they in theory can get further than a smaller creature. As such, the probabilities between the different mutation options are slightly biased in favor of adding modules.

The probabilities seen in Table 3.2 are multiplied with the node mutation rate, leading to an even lower per-gene mutation rate. For every node, each gene mutation can occur with the probability of the individual per-gene mutation rate. Therefore, multiple gene mutations can occur in one node, and multiple node mutations can occur across the morphology when the mutation operator is called.

Phenotype

Each directed tree is sent to the Unity executable in a string representation, where it is decoded and constructed. The built phenotype's modules correspond directly to the nodes in the directed tree.

Before starting the evaluation, the robot is pruned breadth-first of any branches that collide with the robot or the environment. This means that the genome might have unexpressed modules, that

	Mutation rate values
Angle	0.2
Remove module	0.25
Add module	0.3
Copy branch	0.25

Table 3.2: **The morphology gene mutation rate values.** These are further scaled by the global morphology mutation rate.

with further mutation may come to be expressed. In addition, unexpressed modules can cause bloating of the genome, as many modules are added that never physically change the performance. In this case, bloating of the genome will cause the per-module mutation rate to drop, meaning robots are stabilized from mutating if the bloating occurs. It was not a common occurrence in the experiments, but did occur in some individuals.

3.3.2 Evolutionary algorithm

The evolutionary algorithm used has tournament selection, with a tournament size of 4, and generational replacement. It is implemented using the DEAP framework [85].

Generational replacement was chosen because it can sometimes dislodge a population from early convergence. This happens because the best genotype will rarely be kept when the population is mutated, and no elites are kept. Other parameters of the algorithm are also chosen to keep diversity. Most notably, the tournament selection size is kept small to increase selection pressure on the elites, while still being large enough to avoid a noisy evolutionary progression.

The elite from each generation was saved as a file for later use. Therefore, when considering the fitness of a run at generation X, we will not use the elite at generation X, but rather the best elite found up until generation X. This is done because when conducting a search for a robot, realistically you would be interested in the most fit individual and not the last.

Morphology and controller have separate mutation rates that were found by grid search, further explained in the experiment sections. The controllers all have separate mutation powers, where controller parameters are mutated with Gaussian distributions based on the power.

3.3.3 Fitness function

The task that we measure the modular robots on is locomotion away from the origin during a set amount of time (100 steps of 0.2 seconds, roughly equaling 20 seconds in real time). Because this often leads to robots discovering the immediate optima of the somersault (a well-known problem [86]) or simply falling over, the robots are given 2 seconds to fall before the evaluation starts. This period is intended

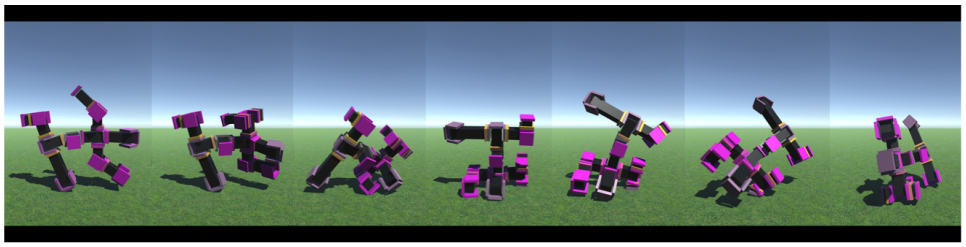


Figure 3.6: **An evolved cartwheel in 7 frames.** The robot initially balances on its "leg" and "arm" before it gains actuation, whereupon it throws itself on its "head" and tumbles over.

for the robot to release the potential energy of its spawn position and come to rest, however the final runs would show that some morphologies evolved to prolong its falling time with for example features such as two limbs to balance on (Figure 3.6). In the start period, the controller is not given input and will not give output.

The fitness function is then

$$Fitness = \sqrt{(x_{end} - x_{start})^2 + (y_{end} - y_{start})^2}$$

where x_{start} is the x-position after 2 seconds of simulation, and x_{end} is the x-position when the simulation ends. Likewise for y. The position of the robot is the position of the root module, always spawned at the world coordinate origin, which discourages the misdemeanor of growing tall and falling. The evaluation will stop early after 4 seconds from start if the robot has not moved in the last 2 seconds.

Due to the scaling of the robots, this fitness should not be interpreted as meters or centimeters, because one unit corresponds to the length of one module. The length of one module is circa 6 cm, and so a score of for example 30 should be interpreted as a score of 1.8 meters.

Chapter 4

Experiment 1

In experiment 1, we will answer these research questions:

1.1 What are the effects of a controller that duplicates control units across the morphology?

1.2 How can a controller best be designed when co-optimizing morphology and control?

In order to address both research questions, we will implement four different controllers. We will have a baseline controller, a centralized controller, a decentralized controller, and lastly a decentralized controller that reuses controls in many different parts of the body. These will be tested simultaneously in one experiment. We analyze the controllers' performance and convergence, as well as their morphological change during evolution, and robustness in the face of random perturbations.

First, the controllers will be presented, after which their mutation rate parameters will be tuned, and then they will be tested on the task of locomotion. Lastly, we analyze the results.

4.1 The controllers

There are four controllers implemented in this system (Figure 4.1), each described below. For all of them, the output produced is the desired angle of a module's servo.

Three of the controllers use a continuous time recurrent neural network (CTRNN), which we explain in more detail in the background. This network was chosen to have the possibility of dynamic temporal behavior [54], as it has memory through recurrent connections and can exhibit any dynamic behavior.

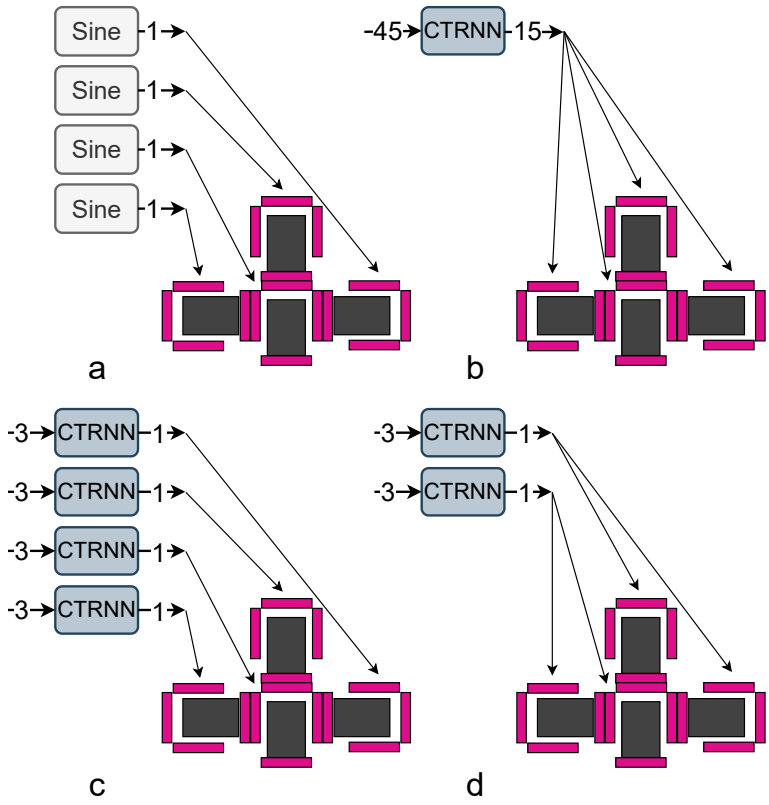


Figure 4.1: **The controllers and how they could map to the modules in a small modular robot.** The arrows with numbers represent inputs (left side of boxes) and outputs (right side of boxes). a) The sine controller, b) The centralized CTRNN controller, c) The decentralized CTRNN controller, d) The copy controller.

We used the CTRNN implementation from the neat-python library[1]. neat-python's CTRNN mutation operators can adjust weights and biases, change activation and aggregation functions, and cut away/disable or add/enable connections and nodes. Here, each gene mutation rate is as seen in Table 4.1, with a mutation power of 0.2, and this is equal for all our CTRNN controllers. These rates are then scaled by the controller's control mutation rate.

In the neat-python library, the CTRNN is implemented as a pure Python class hierarchy. This is good to avoid package dependencies and is easy to manipulate from outside but might affect speed. With large networks, we experienced a considerable increase in simulation time.

[1]https://neat-python.readthedocs.io/en/latest/

Mutation	Prob.	Options/limits
Connection add	0.2	
Connection delete	0.2	
Connection enable	0.01	**True** False
Node add	0.2	
Node delete	0.2	
Bias	0.8	[-1.0, 1.0]
Activation	0.2	sin **tanh** sigmoid
Aggregation	0.1	**sum** min max mean median product maxabs
Weight	0.8	[-1.0, 1.0]
Response	0.2	[-1.5, 1.5]
Bias replace	0.1	
Response replace	0.1	
Weight replace	0.1	

Table 4.1: **The gene mutation rates for a CTRNN controller.** These are scaled by the per-controller mutation rate. Default values are **marked**.

Open-loop sine wave generator

The open-loop sine wave generator is a decentralized controller that performs well because it produces periodic movement through sine waves, although it has no sensor input. In our case, the sine wave controller is used to provide a baseline to compare the other controllers to. The controller is given by the function

$$y(t) = A * \sin(w * t + p) + o$$

where A is the amplitude, w is the frequency, t is time, p is phase, and o is offset. $y(t)$ is the controller output at time t that is directly fed to the servo's desired angle. The amplitude, phase, and offset mutate with a Gaussian distribution with a standard deviation of 0.5.

To enable easier synchronization between the modules, the frequency is fixed in all sine wave controllers and is not allowed to mutate. We noticed that without this, the sine controller is susceptible to choose local optima solutions.

The sine wave controller has 3 parameters for each joint, meaning an average robot of 6 modules will have 18 parameters to optimize for the controller. When a module is added to the morphology, it is instantiated with the control parameters of the parent module.

Centralized CTRNN

A straightforward approach to using a CTRNN for a modular robot is to simply gather all sensor outputs and feed them into one big CTRNN, that then outputs all controller actions. This leads to a fixed size CTRNN controller.

In initial experiments with the centralized CTRNN controller, different numbers of inputs and outputs were tested. Initially, because the system at large is meant for max 50 modules, 50 modules were allowed in the network. Because this led to very large networks and very few modules used, the number of modules were gradually reduced after viewing the trends in size and performance. Because there was a clear tendency to have a small number of modules, we chose 15 to be the number of modules that would be controlled. This allowed the network to be as small as possible while still accommodating larger creatures at initialization, however it was very rare to see larger creatures with this controller. When a modular robot is smaller than 15 modules, the rest of the inputs to the network is set to 0.

The centralized CTRNN controller will have 45 inputs and 15 outputs. It is initialized with 45 hidden nodes but is only 20% connected. The order of mapping modules to input and output follows a depth first ordering of the modules, so that the first three inputs and first output goes to the root, the second three inputs and second output goes to its child, and so on.

The centralized CTRNN controller ends up having circa 600 connections and 60 nodes, each with respectively 3 and 5 parameters to tune, for a total of 2100 parameters. Because of this size, and the neat-python implementation, the evolution time of robots with this controller was the bottleneck that decided the number of samples to use.

Decentralized CTRNN

Foregoing the benefits of a centralized brain, a decentralized approach leads to less parameters and a less complex optimization. The decentralized approach consists of each module getting its own CTRNN controller. It is implemented in the same way as the sine controller, except CTRNNs are slotted in where previously there were sine controllers. So likewise, new modules are instantiated with the control parameters of the parent module.

Here, we were able to use a small compact network of 3 inputs for each sensor and 1 output for the action. With 3 hidden nodes and enabling all connections from the start, we get 4 nodes and 16 connections, totaling 68 parameters. For an average sized robot with one of these controllers in every module, this leads to about 400 parameters.

Copy Decentralized CTRNN

The copy decentralized CTRNN controller, or the copy controller for short, is our alternative to the decentralized approach. It functions by having each robot keep a list of two CTRNN networks, which maps to different modules. The networks are the same as in the decentralized CTRNN controller. At initialization, these networks are clones, but as the optimization progresses, they will mutate separately. The modules will then mutate which network they use for control, theoretically allowing specialization. When a module is added to the morphology, it will use the same network as its parent module.

At the start of an evaluation, the networks are copied into their corresponding modules, hence the name. They then function independently of each other, and because of different sensor input such as detecting ground or the presence of child modules, they will likely behave differently. Nevertheless, it is reasonable to assume it will not achieve the level of specialization that the decentralized CTRNN controller can. While this might be a trade-off, we assume the copy controller will be quicker to achieve a good fitness, and possibly not be as dependent on number of modules.

This controller, like the centralized CTRNN controller, is the same no matter the size of the robot. This gives us two controllers with 68 parameters, for a total of 136 parameters. Additionally, each module can change which controller they use, giving us a further average of 6 parameters.

4.2 Parameter tuning

For each controller, a set of 8 suitable controller mutation rate parameters were chosen to test along the 8 morphology mutation rate values. This resulted in a total of 64 pairs to check for each controller, as seen in Figure 4.2. For the body and the sine and decentralized CTRNN controllers, higher values were chosen than

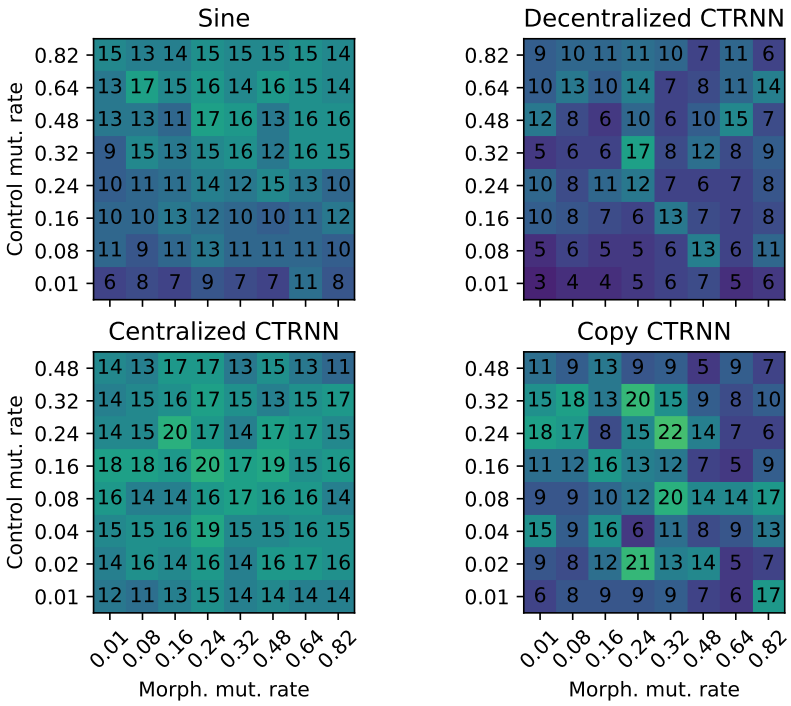


Figure 4.2: **The grids filled for the parameter tuning**, values are rounded average fitness for each mutation rate pair.

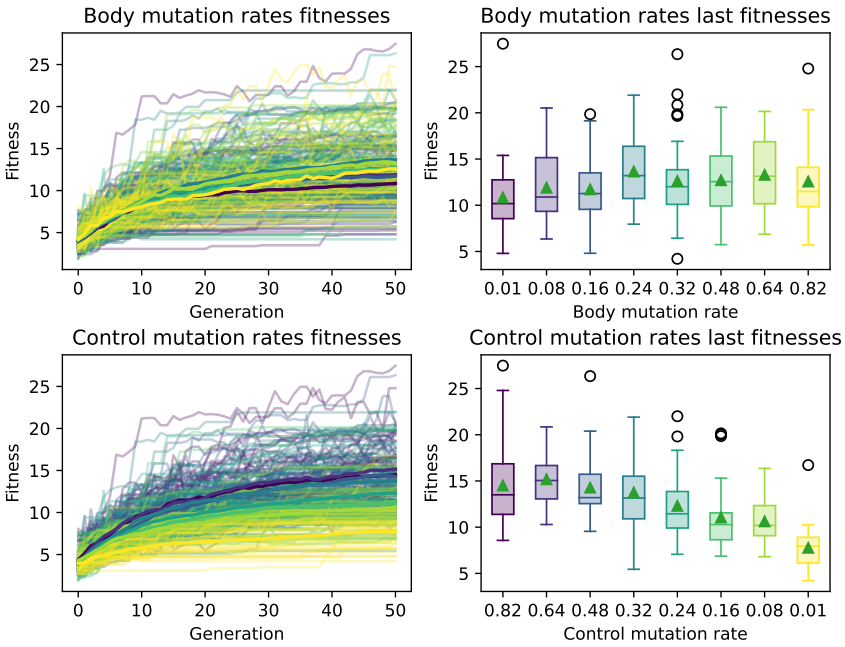


Figure 4.3: **The collapsed rows (controller mutation rates) and columns (morphology mutation rates) for the sine controller grid search.**

for the copy and centralized CTRNN controllers: This was because the former three divide the mutation rate internally by the number of modules, while the latter two do not. If it were not divided, a larger creature would mutate more than a smaller one. This would make it an unstable solution that quickly changed and disappeared from the search space. Since the average number of modules is 6 (Figure 3.5), the per-module mutation rate is 0.14 for the largest sweep value of 0.82. In the morphology mutation, this is further divided by circa 1/4th for each gene in the module, exact values can be seen in Table 3.2

Each mutation rate pair was run for 50 generations with a population size of 50, which totaled 2500 evaluations. This was done 4 times for each of the 64 pairs.

As can be seen in Figure 4.2, the results from the sweep were often quite even. Only the copy and decentralized CTRNN controllers saw huge differences between different pairs. Because we suspected there were a lot of variation in the data, simply picking a well performing square was likely not a good strategy. As such, the data in the columns and rows were collapsed to give an idea of each parameters performance while varying the other, as seen in Figure 4.3. This gives us a more robust measure of performance because we have more data for each parameter value and are less dependent on the other mutation parameter.

Even after collapsing the data, it was not immediately clear which values to choose. For example, in Figure 4.3 we can tell the controller mutation rate should be somewhere in the interval 0.48-0.82, but the differences between these values are not large. Therefore, we settled on choosing a few of the ones that were contenders after the initial sweep and run a few more evolutionary runs on those (circa 10 more runs). A winner would then often be clearer. The final parameters for all controllers can be seen in Table 4.2.

Controller	Morph. rate	Controller rate
Sine wave	0.32	0.64
Centralized CTRNN	0.24	0.16
Decentralized CTRNN	0.24	0.48
Copy CTRNN	0.32	0.08

Table 4.2: **The mutation rate parameters chosen after the sweep.** Note that for the sine wave and decentralized CTRNN controller, as well as the morphology, the working mutation rate on one module is divided by the number of modules in the robot.

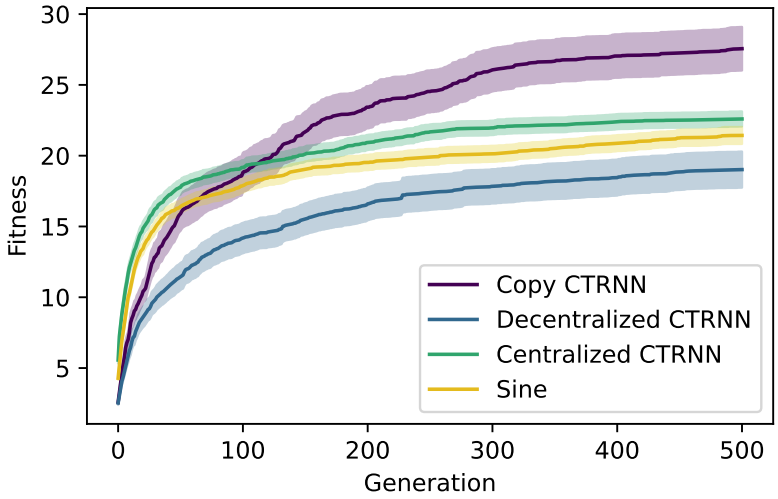


Figure 4.4: **The fitness progressions for all four controllers.** The solid lines are averages, and the shaded areas represent the standard error. Distributions can be seen in Figure 4.5.

4.3 Results

4.3.1 Controller performance

The final runs were done on populations of 50 individuals for 500 generations, for a total of 25 000 evaluations. This was done for all controllers 64 times, and the resulting performances can be seen in Figure 4.4. Additionally, Figure 4.5 shows the distribution of performances for the different controllers.

In order to get an overview of all significant differences, 6 two-sided Mann-Whitney U test were performed between all controllers at generations 50 and 500, a total of 12 tests. The null hypothesis is that the final performances came from the same distribution, while the alternative is that they did not, that they are different. An alpha level of 0.05 was chosen. Because we were conducting multiple comparisons, Bonferroni correction was used. This gives us an adjusted alpha level of 0.05 / 12 = 0.00416. The test results can be seen in Table 4.3.

At generation 50, the sine and centralized CTRNN controllers were significantly different from the decentralized CTRNN controller (both $p < 0.0001$). There was no significant difference between the sine and copy ($p > 0.2$), sine and centralized CTRNN ($p > 0.07$), copy and centralized CTRNN ($p > 0.06$), and the copy and decentralized CTRNN controllers ($p > 0.01$).

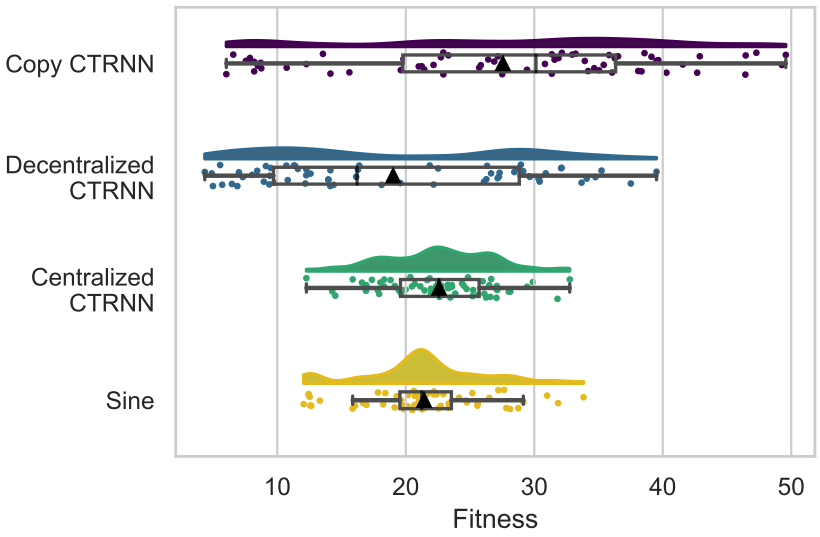


Figure 4.5: **The final fitnesses for all runs for each controller.** The distributions are showed with the underlying points scattered below. A boxplot is placed over the scattered values, with the mean marked with a triangle.

At generation 500, the copy controller was significantly different from the centralized CTRNN, the sine, and the decentralized CTRNN controllers (respective p-values $p < 0.003$, $p < 0.0004$, $p < 0.0002$). There was no significant difference between the sine controller and the centralized and decentralized CTRNN controllers (both $p > 0.1$), or between the centralized and the decentralized CTRNN controllers ($p > 0.04$).

4.3.2 Effect on morphology

In Figure 4.6, the progressions for number of modules in morphologies are plotted for all the controllers. Here, we can see that the sine

	vs.	Gen. 50	Gen. 500
Copy CTRNN	Sine	0.22886	**0.00037**
Copy CTRNN	Cen. CTRNN	0.06142	**0.0025**
Copy CTRNN	Dec. CTRNN	0.01642	**0.00012**
Cen. CTRNN	Sine	0.07431	0.10264
Cen. CTRNN	Dec. CTRNN	**1e-07**	0.04559
Dec. CTRNN	Sine	**1e-06**	0.11747

Table 4.3: **The p-values for the Mann-Whitney U tests performed between all the controllers' best achieved fitnesses.** Significant p-values are marked in bold.

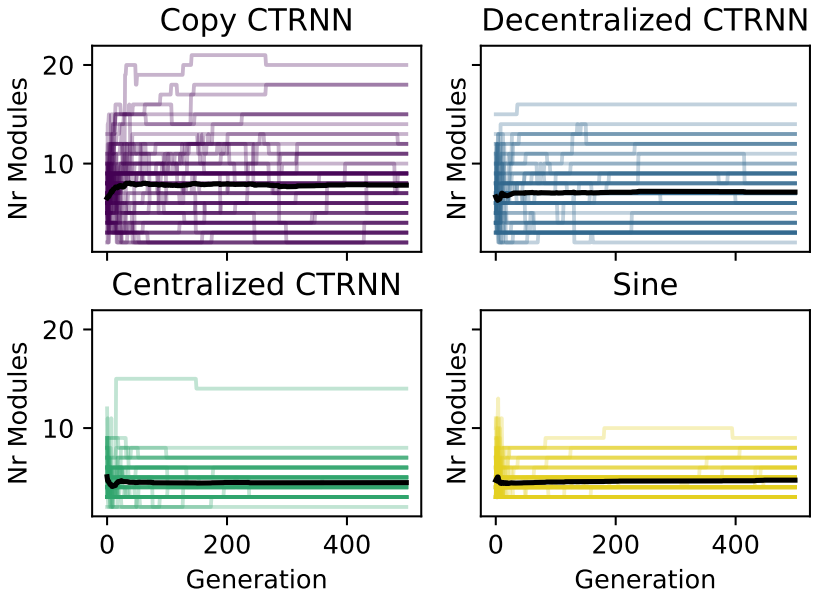


Figure 4.6: **The progression of number of modules for all runs.** Only changes in number of modules that led to a more fit individual is shown.

and centralized CTRNN controllers both end up at a lower average number of modules than the copy and decentralized CTRNN controllers. To confirm if this was significant, as previously 6 two-sided Mann-Whitney U tests were performed with Bonferroni correction between the different count distributions. An alpha level of 0.05 was used, which means that with correction we consider p-values below 0.05 / 6 = 0.0083 as significant.

Here we found that the copy and decentralized CTRNN controllers had no significant difference between them ($p > 0.4$) and the sine and centralized CTRNN controllers likewise had no difference ($p > 0.2$). However, the copy and decentralized CTRNN controllers were both different from the sine and centralized CTRNN controllers (all $p < 0.0001$).

Qualitatively, we recognize this from looking at the robots. The sine and centralized controllers had small, effective strategies while the copy and decentralized CTRNN controller both tended towards larger morphologies. The sine and centralized CTRNN controllers would do large, powerful movements, with some modules in the morphology not moving at all. As opposed to this, the copy and decentralized CTRNN controllers favored small, rapid movements. Here, the copy moved most its modules, while the decentralized CTRNN controller sometimes had unmoving modules. Example

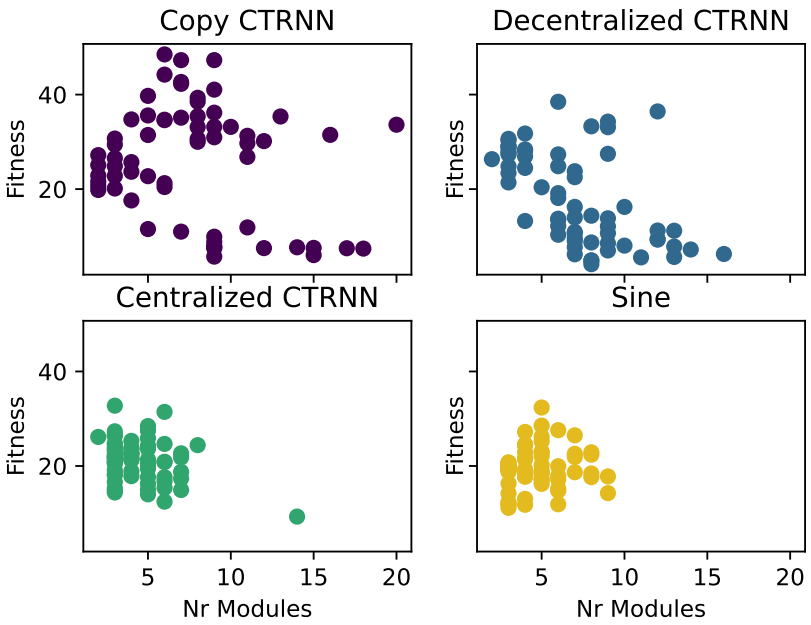


Figure 4.7: **The expressed modules of all generation 500 elites plotted against their fitness.**

behaviors can be seen in the accompanying video[2].

In Figure 4.7, we see that there seems to be a divide between larger creatures that get high fitness, and those that get low fitness. Presumably, this is because falling strategies tend to be larger in order to get further, but it is also interesting to see that some larger solutions did quite well. Especially the copy controller did well with larger morphologies, managing to produce a fitness on par with the centralized CTRNN and sine controllers with upwards of 10 expressed modules.

When looking into the issue of early convergence of morphology only, Figure 4.8 was made. It shows shaded areas between the 25 and 75 percentiles of number of beneficial morphology changes in a run in each interval of 50 generations. It shows that the centralized CTRNN controller mutates its morphology the least from the very start and stops mutating much at circa generation 200. The sine and decentralized CTRNN controllers mutate a little more, especially at the start, but follow a similar pattern. Compared to this, the copy controller experiences more beneficial morphology mutations at the start than the others, and continually throughout the runs. This is congruent with the slow convergence seen in the fitness progression

[2]https://www.mn.uio.no/ifi/english/research/groups/robin/research-projects/cocomo/media/kvalsund_master_thesis.mp4

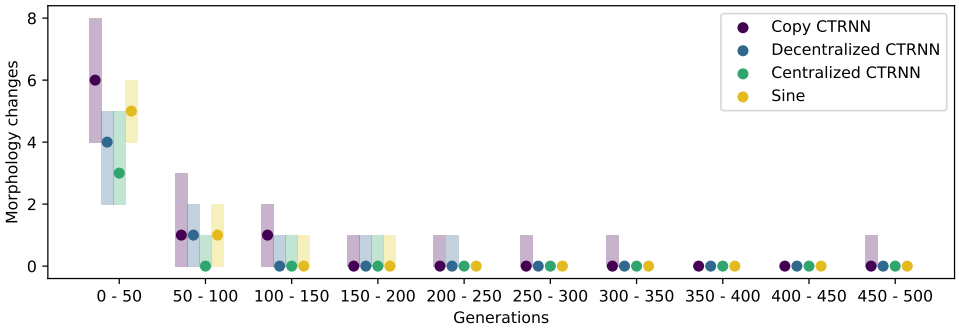


Figure 4.8: **Number of morphology changes that led to better fit individuals in each generation interval**. The shaded areas are between the 25 and 75 percentiles, and the dots are the medians.

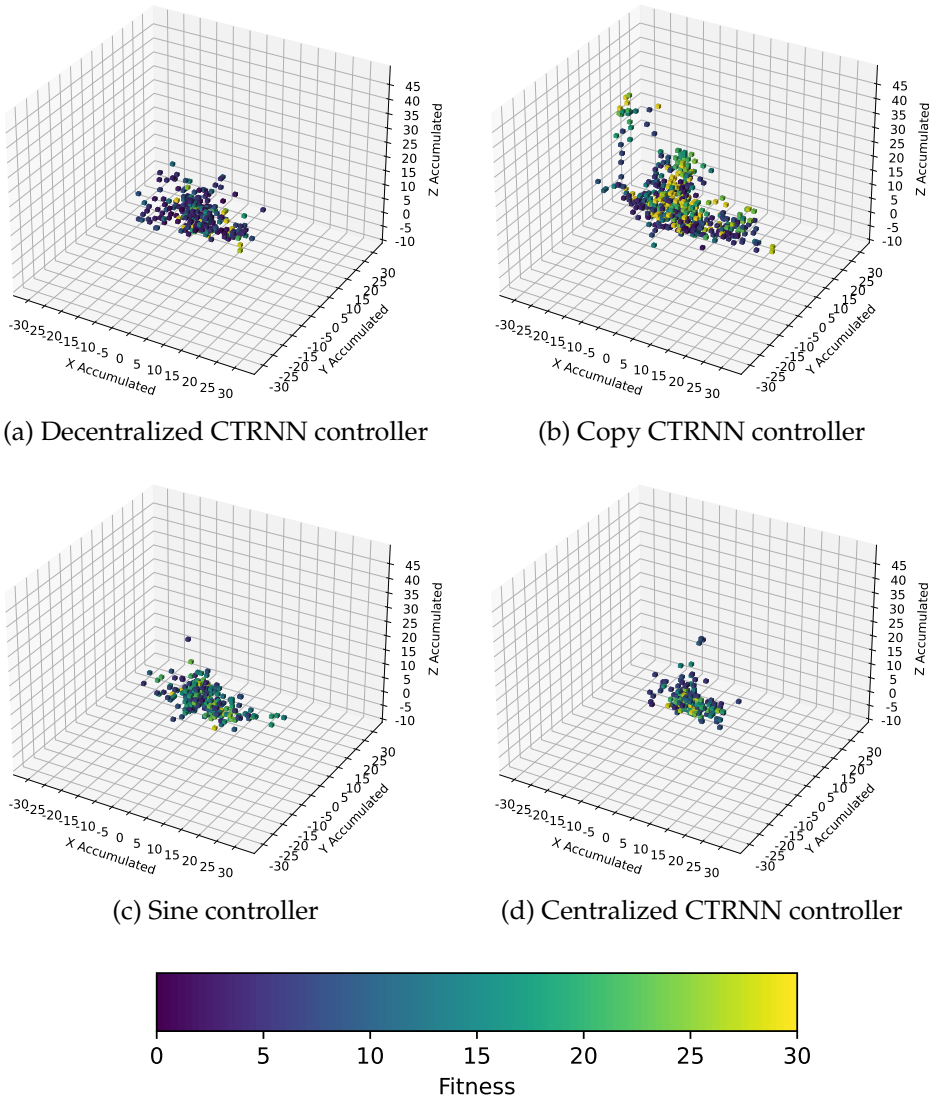
figure, Figure 4.4.

To support this, Figure 4.9 was made to show the different controller's traversal of the search space. Only the best individuals in each generation were used to give a sense of how a controller traversed the morphology space. Here, each module coordinate in a robot was added together to create a new coordinate representing a morphology after the following formula:

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} + \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} + ... + \begin{bmatrix} x_n \\ y_n \\ z_n \end{bmatrix} = \begin{bmatrix} x_a \\ y_a \\ z_a \end{bmatrix} \tag{4.1}$$

where (x_i, y_i, z_i) is the position of module i at initialization, n is the number of modules, and (x_a, y_a, z_a) is the new abstract coordinate. Note that different morphologies can map to the same coordinate, and that the abstract coordinate was rounded to the nearest integer coordinate in the figure.

These morphology landscape traversal plots suggests that the copy controller explores more of the landscape and has more different morphologies than the other controllers. The sine and decentralized CTRNN controllers also appear to cover a bit more of the search space. Compared to these, the centralized CTRNN controller explores a smaller portion of the morphology landscape. This is especially noticeable because the sine and centralized CTRNN has the same mean number of modules. After rounding to the nearest integer coordinate, the copy controller fills 489 unique voxels, the decentralized CTRNN controller fills 234 voxels, the sine controller fills 214 voxels, and the centralized fills 135. We are more interested in the



(a) Decentralized CTRNN controller

(b) Copy CTRNN controller

(c) Sine controller

(d) Centralized CTRNN controller

Figure 4.9: **The explored morphology landscape by the elites from each generation for all runs**. Each plot has all 64*500=32000 individuals' abstract morphology position rounded to a voxel coordinate. The axes are the same to show the relative size of each cluster.

amount of space traversed, rather than the granularity at which a space was searched, however there is little difference between number of unique coordinates and number of unique voxels.

4.3.3 Robustness

To test robustness, we considered two typical malfunctions in modular robots: Modules falling off and modules losing power and

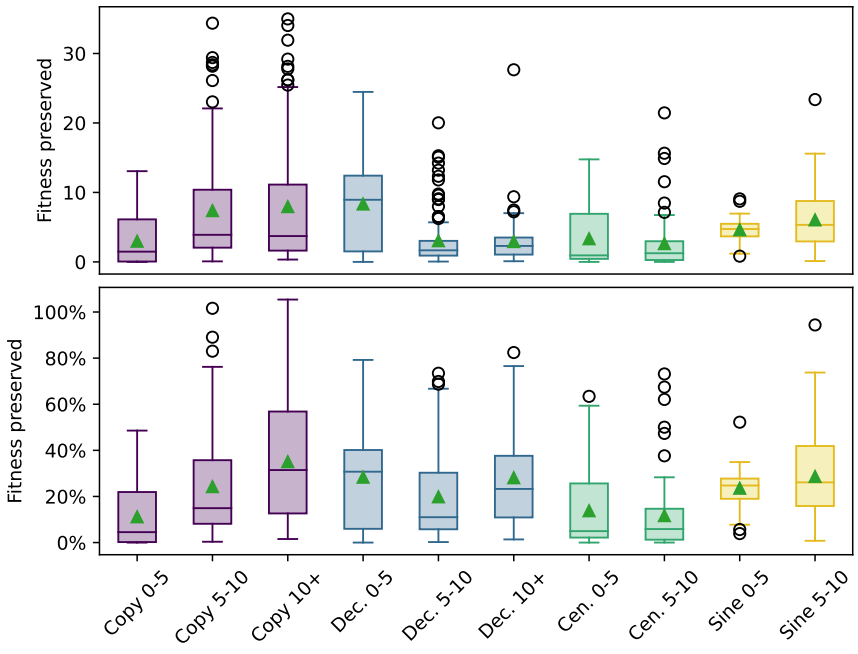
becoming passive. The first malfunction will be simulated in one kinder test, and one quite difficult: The kinder test removes only one leaf node at a time and notes the fitness change, while the more difficult test systematically removes each module. To simulate losing power and becoming passive, each module in turn has its control disabled.

These three tests are done for each elite from all 256 runs. In addition, we differentiate between the size intervals 0-5 modules (small size), 5-10 modules (medium size), and 10+ (larger size). This is done because larger creatures could respond better to perturbations, given that one module makes up less of their total mass.

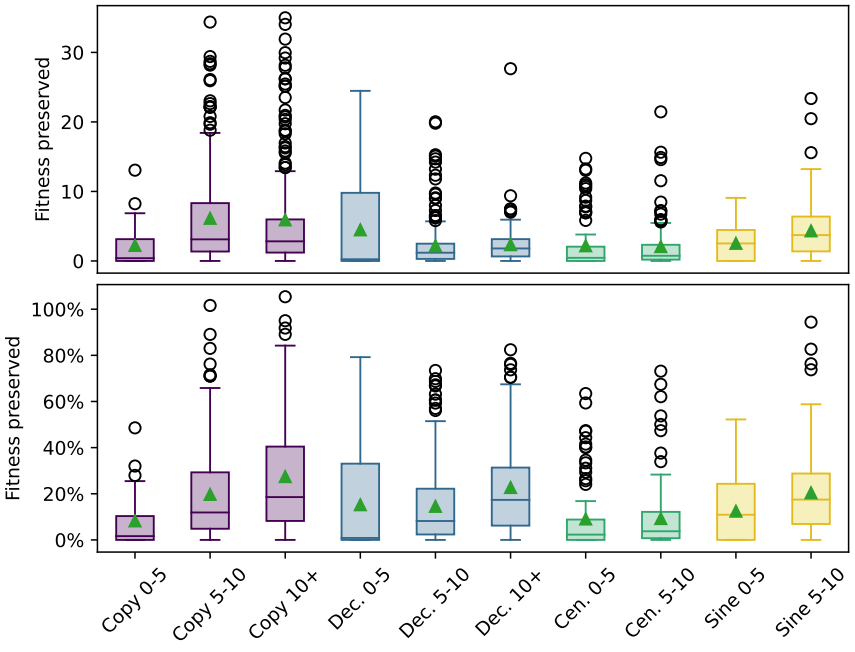
The results from removing only leaf nodes can be seen in Figure 4.10a. The copy and sine controllers preserve progressively more of their fitness the larger the creature, although the sine controller never reaches the 10+ size range. The decentralized and centralized CTRNN controllers both appear to preserve roughly the same fitness percentage at each size range. Especially the small sized decentralized CTRNN robots and the medium sized sine robots preserve a lot of their fitness.

The test of removing each module in turn is more volatile than the last, and the results can be seen in Figure 4.10b. The results are quite similar to the test above, although now most of the percentages are lower. Here, we can see that most of the controllers are rendered unusable after this, except for the medium and large sized copy controller and the medium sine controller. We see this because the preserved fitnesses of the others are so low they are likely simply falling over.

Lastly, the test of disabling each module's control in turn yielded slightly more fitness preservation than the other two tests. The results can be seen in Figure 4.11, and it appears that disabling control is far less volatile than losing modules. Again, the copy controller shows more robustness the larger it gets, and notably so does the rest of the controllers too. The copy and decentralized CTRNN controllers are by far the most resistant to this malfunction given their larger sizes, however they both show much worse fitness preservation than the other two controllers at the small size range.



(a) Removing only leaf modules



(b) Removing modules

Figure 4.10: **Fitness preservation when applying module removals.** All 256 elites had one remove perturbation applied, specified in each subfigure, and their performances measured. Bottom shows the percentage of original fitness that was preserved, and the top shows the total preserved fitness.

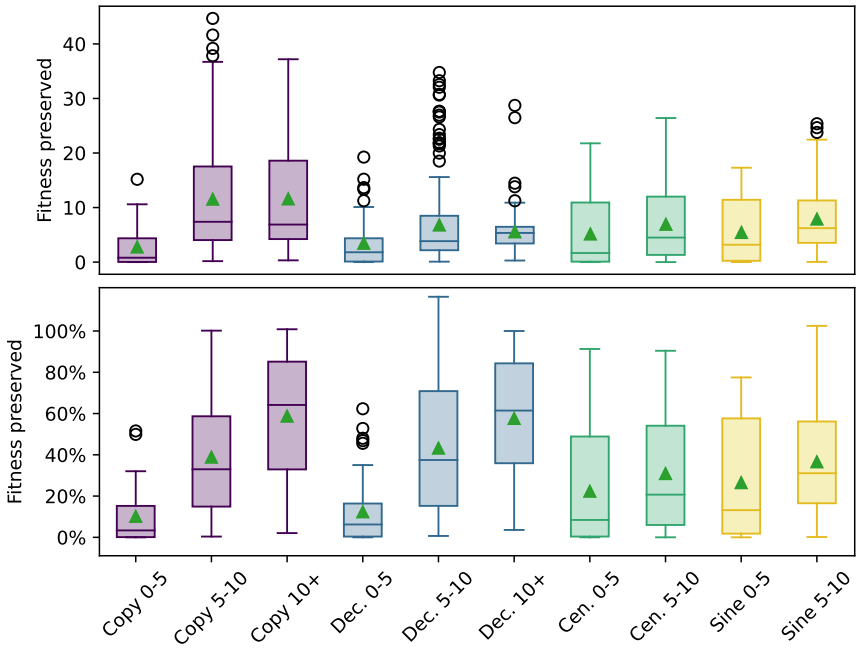


Figure 4.11: **Fitness preservation when applying control disabling.**

4.4 Analysis

Our results have shown that the copy controller performs significantly better than other controllers when co-optimizing the morphology and control of modular robots. Since it duplicates behaviors, modules are more likely to be synchronized. Moreover, when a new module is added, a working control unit can be inherited that is already potentially useful. Even though the sine and decentralized CTRNN controllers had a similar feature of inheriting the parent module's control, the copy controller is more likely to be useful since it is already evolved to work in many different parts of the robot. In addition, because a control mutation affects multiple modules, it has an overall larger effect on the behavior of the robot compared to a mutation in the other controllers. Because of this, the copy CTRNN approach is less able to fine-tune a single controller compared to the other approaches and therefore may rely on morphological change to see a performance increase. This feature would thereby promote continued morphological diversification compared to the other controllers, as seen in Figure 4.8.

In terms of controller robustness, we have shown that the copy controller displays a pattern of reliably getting more robust to perturbations the larger its morphology is. Because the copy controller only has two networks, it is possible that the controller

is using its sensors to detect the presence of children and in that way modify the networks' behavior to have more specialization. In that case, when losing modules, the copy controller will adapt its behavior to the new morphology. This feature is useful when there are perturbations to the system, making it robust. This could also be one of the reasons the copy controller frequently gains high fitness with large morphologies: The controller is simply robust to morphology mutations. It is likely therefore we see increased robustness in the larger morphologies compared to the smaller ones: The same controllers that would be beneficial when adding modules would also be beneficial when removing modules. However, growing larger is favored because it adds force to the robot, allowing it to get further.

From the fitness progressions, we can see that the sine and centralized CTRNN controllers converged rather fast compared to the other two. They also showed a pattern of quickly finding a final morphology of relatively small size, and then optimizing the controller. Meanwhile, the other two controllers spent time developing both and thus converged slower. Because having more modules means the robot has more potential force, allowing for more movement and higher fitness overall, the sine and centralized CTRNN controllers were then at a disadvantage. These results confirm that there is a trade-off between fine-tuning controllers and getting a good fitness with a small morphology while losing the potential of getting a higher fitness and a large morphology.

The distributions of solutions for the controllers vary wildly, as seen in Figure 4.5. While the sine and centralized CTRNN controllers had a more solidly high performance, the decentralized CTRNN controllers both had very flat distributions, stretching from the worst to the best performances recorded. The lower fitnesses can be accounted for as robots that grow tall and fall in one direction, as some of these have been visually confirmed to be. The higher values of the copy, decentralized, and centralized CTRNN controllers often had rapid module movements that either led to small jumps or shuffling behaviors. Because of fixing the sine controller's frequency, this strategy was not available to the sine controller, and so its worse performance must at least be partially attributed to that. Even though it could have rivalled the others by growing larger, the CTRNN controllers' behavior was likely less complicated to evolve. Still, the sine and centralized CTRNN much more often arrived at

very similar local optima, which tended to have the same fitness. Here, the centralized CTRNN controller had an advantage over the sine controller because it could optimize further by adding non-periodic movements.

Because we had the same morphology mutation rate for the sine and copy controllers, we could expect similar morphological diversity from these. However, it is clear from our results that this is not the case. When keeping in mind that some of the more scalable strategies available to the CTRNN controllers were not available to the sine controller, it could simply be that there were less available good morphologies for the sine controller. Even so, the lack of exploration of morphologies suggest that the sine controller could be improved. Perhaps using a decentralized copy approach like the copy controller would have enabled more morphological diversification during evolution.

Every controller shows increased robustness with larger sizes when faced with controller perturbation. It is also the least detrimental perturbation of the ones tested. It is likely that as opposed to losing mass and actuation, having a dead module is simply less disruptive. However, it can be argued that because the morphologies are largely decided early in the runs and survive through many controller mutations, the morphologies are far more robust to controller changes than the controllers are to morphology changes. This further emphasizes the challenge of co-optimizing morphology and control, where morphologies are side-lined in the optimization, while also possibly becoming the weak link in response to damage.

Chapter 5

Experiment 2

In experiment 2, we address these research questions:

2.1 How do encodings with different degrees of gradual mutation affect landscape traversal?

2.2 How do encodings with different degrees of gradual mutation affect performance?

In order to answer these questions, direct encodings with varying degrees of gradual mutation will be tested on the task of locomotion in order to compare their performances. All the encodings will be tested with all the controllers from experiment 1 to understand under what circumstances the encodings are useful. In addition, we will assess the smoothness of mutation in the encodings.

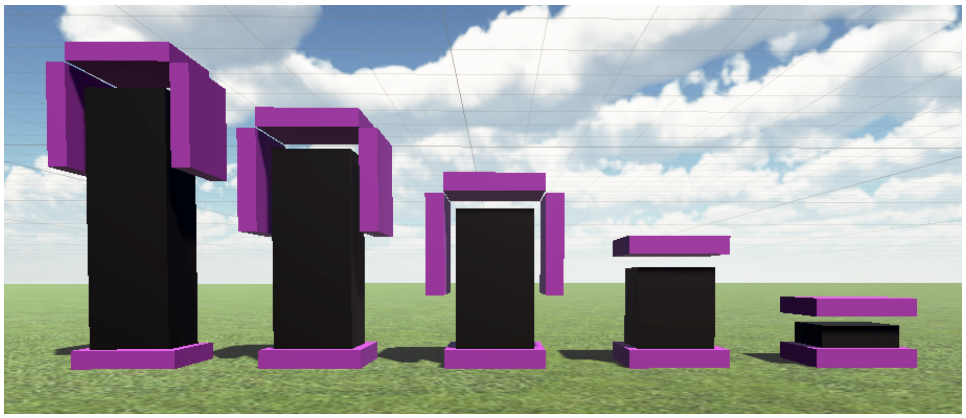


Figure 5.1: **Examples of module sizes.** These are respectively, from left to right, 1.9, 1.45, 1.0, 0.55, 0.1.

5.1 Variable scale of modules

In order to have more gradual mutation, we introduce being able to scale the module length, as seen in Figure 5.1. Only the middle box is scaled in length, and can be any scale above 0.05, although for this experiment scale is capped at 3.0. When the three female connection plate are rotated by the servo motor, they rotate relative to the top of the box no matter the scale.

When the scale of a module is below 1.0, its actuation is removed and the motion between the top connection site and box goes from being limited to being locked. This is done because a servo motor would not realistically be able to fit in a shorter module. In addition, the connection sites at the sides are removed because they would collide with the parent module when the module is short. In this case, the sensors on these connection sites return -1.

Because we are working with variable sizes, the mass change also had to be modeled. We wanted the mutation of adding length to be as gradual as possible, and therefore we modeled the mass change as a linear increase:

$$m = 1 + 4s$$

where m is the mass in kilos, and s is the scale of the module. At scale 1.0, the weight is exactly 5, making the three top connection sites and the bottom box and connection site together have a mass of 10.

5.2 The encodings

Here, three alterations to the normal direct encoding used in experiment 1 is presented. They all use the variable scale of modules as described above, and so the mutation probabilities between mutation options are now as described in Table 5.1. In addition, a mutation power had to be introduced for scale, meaning scale mutates with a normal distribution with a standard deviation of 0.5.

When a robot is initialized, all modules in all encodings have scale 1.0. This is done so that all encodings will develop from the same basis and let us more easily compare their performances. However, the length of modules that gets added through mutation depends on the encoding used.

Although these encodings are constructed to be gradual, they all are still allowed the copy branch mutation, which can be very

disruptive. This was done because being able to make large jumps in the search space can introduce more variation. As our algorithm does not have crossover operators to explore, and struggled with local optima, we hoped the copy branch mutation could provide this variation. With these encodings, we hope to instead enable the robots to climb hills more easily in the landscape, while also letting them make large alterations to the genome.

It is important to note that if a module with a child mutates to be shorter than 1.0 in scale, its side children will be removed, and its top child will remain. This can lead to quite vast changes to the genome, as a non-leaf node scaling to below 1.0 can cut away many nodes at once. With the same reasoning as was used for keeping the copy mutation, this behavior was kept in all the encodings.

Variable scale encoding

The variable scale encoding adds only two aspects to the normal direct encoding: Modules can now mutate their scale, and added modules can have any scale between 0.05 and 2.05.

This encoding does not make mutation more gradual, but it is intended to control for the effect of allowing variable scale. It might be that simply allowing modules to be longer will have the effect of walking further because creatures are larger. If this is the case, we expect to see the variable scale encoding outperform the normal direct encoding, and we must then compare the more gradual encoding performances to this one. However, it might also be that modules become shorter, as this leads to less weight per servo.

Mutation	Fixed scale	Variable scale
Angle	0.2	0.15
Remove module	0.25	0.2
Add module	0.3	0.25
Copy branch	0.25	0.2
Scale	0.0	0.2

Table 5.1: **The morphology gene mutation rate values for fixed scale and variable scale encodings.** These are further scaled by the global morphology mutation rate.

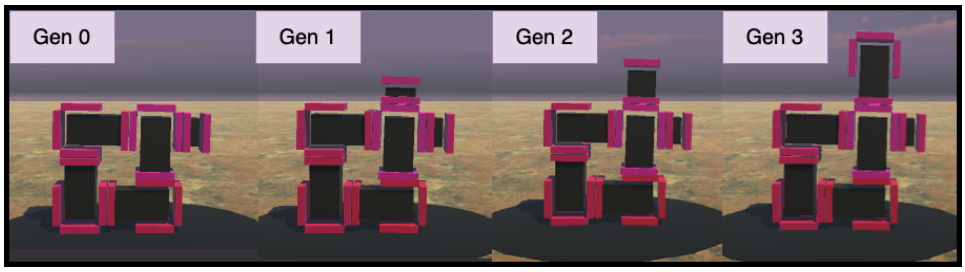


Figure 5.2: **Growing of a module illustrated.**

Growing encoding

Further adding on the variable scale encoding, the growing encoding changes only one thing: Added modules are now 0.05 in scale. They will be allowed to grow by mutating scale. In an ideal case, a module will slowly reach full length and gain actuation through consecutive mutation as seen in Figure 5.2.

This encoding is intended to make the add module mutation less disruptive: In a small robot, an add module mutation will add a large amount of mass and movement without giving the robot any development to adjust. In theory, this mutation slowly introduces the mass and size until a scale of 1.0 is reached, whereupon it will gain actuation. It might be the case that this slow change encourages the addition of modules, and that we will see more modules in robots of this type of encoding.

Gradual encoding

Finally, the gradual encoding is the most gradual encoding in this experiment. It builds on the growing encoding by adding several things:

1. When a module is not fully grown, it is not allowed to add a child module at the available connection site. Instead, when an add module mutation occurs, the scale of the module is instead increased.

2. Only leaf nodes can be removed.

3. When the leaf node to be removed has a scale of over 0.25, it will not be removed. Instead, its scale will be decreased.

This encoding was made with the intention of reducing the disruption of both the add module and remove module mutation.

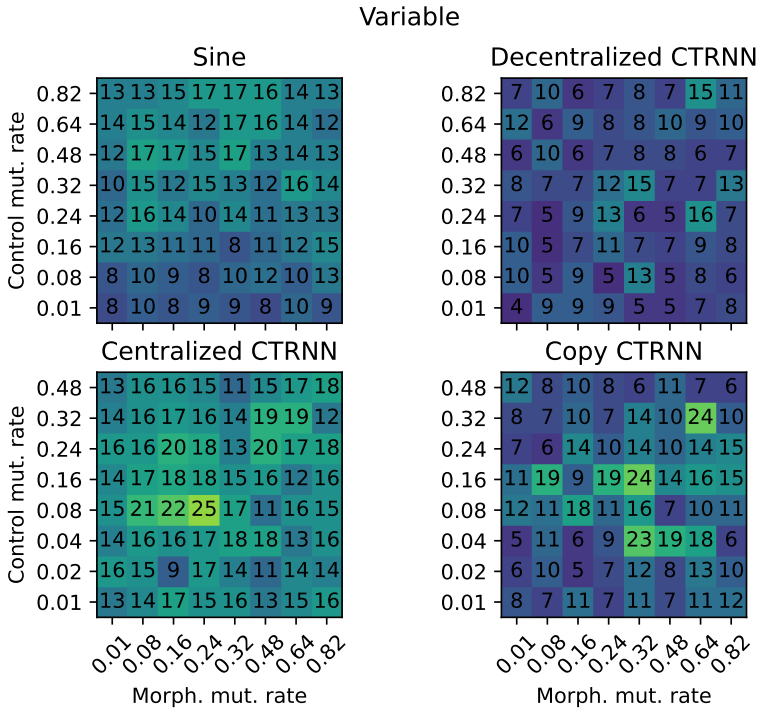


Figure 5.3: **The grid search for all controllers with the variable scale encoding.**

It also further encourages modules to grow out: Compared to the growing encoding, there is a strong bias towards scaling up as both the 0.25 chance of adding a module and the 0.2 chance of scaling can lead to a larger module.

5.3 Parameter tuning

As in experiment 1, a grid search will be conducted for each of the 12 new combinations of controllers and encodings. In total there will be 16 combinations, but the data from experiment 1 will be reused. Again, 50 generations with a population size of 50 was run 4 times for each of the 64 mutation rate pairs. All the encodings were tested for the same morphology mutation rates. The results for each encoding can be seen in Figure 5.3, Figure 5.4, and Figure 5.5.

After collapsing the rows and columns as was also done in the experiment 1 tuning, it was clear that there were no larger differences between the normal and the variable scale encodings' mutation rate performances. In fact, it appeared that the ideal parameters for each encoding and controller combination was close or identical to the normal encoding's parameters for that controller.

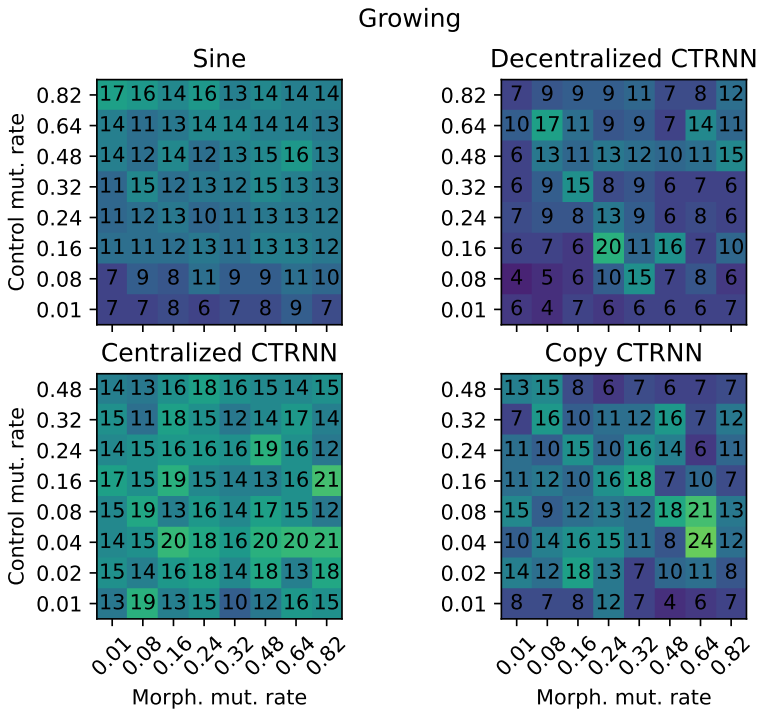


Figure 5.4: **The grid search for all controllers with the growing encoding.**

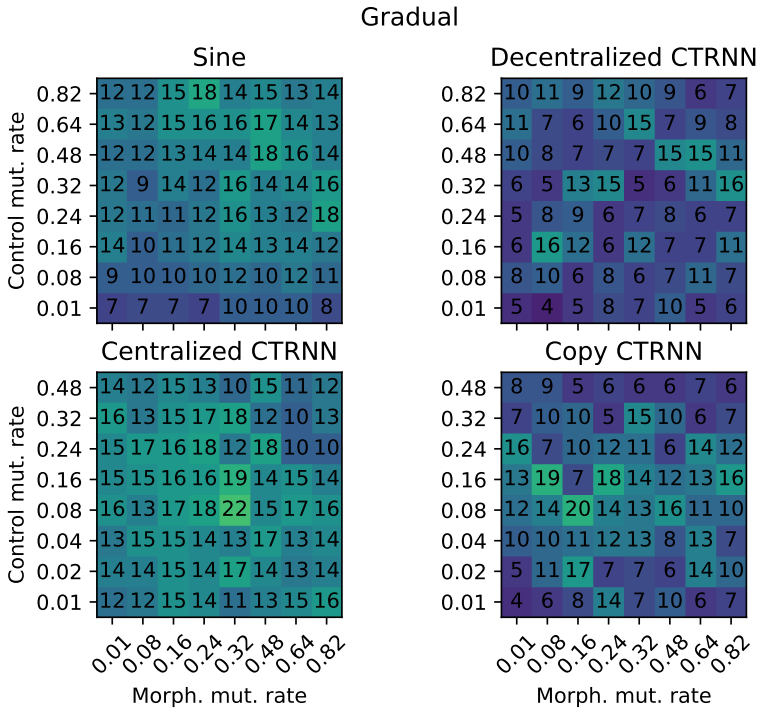


Figure 5.5: **The grid search for all controllers with the gradual encoding.**

While running a few more runs on parameter contenders could find the ideal parameters for each encoding, we feared that random variation could cause a bad set of parameters to be chosen on accident. If the encoding approaches have no difference between them as the tuning data suggests, we would then end up comparing a bad to a good tuning, and not actually investigating the hypothesis. This is especially a concern compared to in experiment 1 because the tuning data for the different encodings are so similar amongst controllers.

With this in mind, we chose to reuse the mutation rates for each controller from experiment 1, as seen in Table 4.2. We do run the risk of an encoding having the potential to perform better with a different set of parameters, however we believe that doing this will facilitate better comparison.

5.4 Results

5.4.1 Encoding performance

The final runs were done on populations of 50 individuals for 500 generations, for a total of 25 000 evaluations. This was the same as in experiment 1, which allowed us to reuse the normal encoding performances for all the controllers. In order to match the sample size of the new data, which was 32 runs for each controller-encoding pair, 32 samples were picked from the experiment 1 data for each controller. The resulting performances of all 512 runs can be seen Figure 5.6. Examples of evolved phenotypes can be seen in the accompanying video[1].

Although there appears to be few differences at a glance, the copy and decentralized CTRNN controllers do display some differences in fitness. However, these controllers also have a lot of variation in performance, as we saw in experiment 1.

To determine which of the apparent differences were significant, six two-sided Mann-Whitney U tests were performed between each encoding with one controller. This results in a total of 24 tests. The null hypothesis is that there is no difference between the encodings we test between, and the alternative is that there is. Like in experiment 1, an alpha level of 0.05 is chosen, that with Bonferroni correction results in an adjusted alpha level of 0.05 / 24 = 0.00208.

[1] https://www.mn.uio.no/ifi/english/research/groups/robin/research-projects/cocomo/media/kvalsund_master_thesis2.mp4

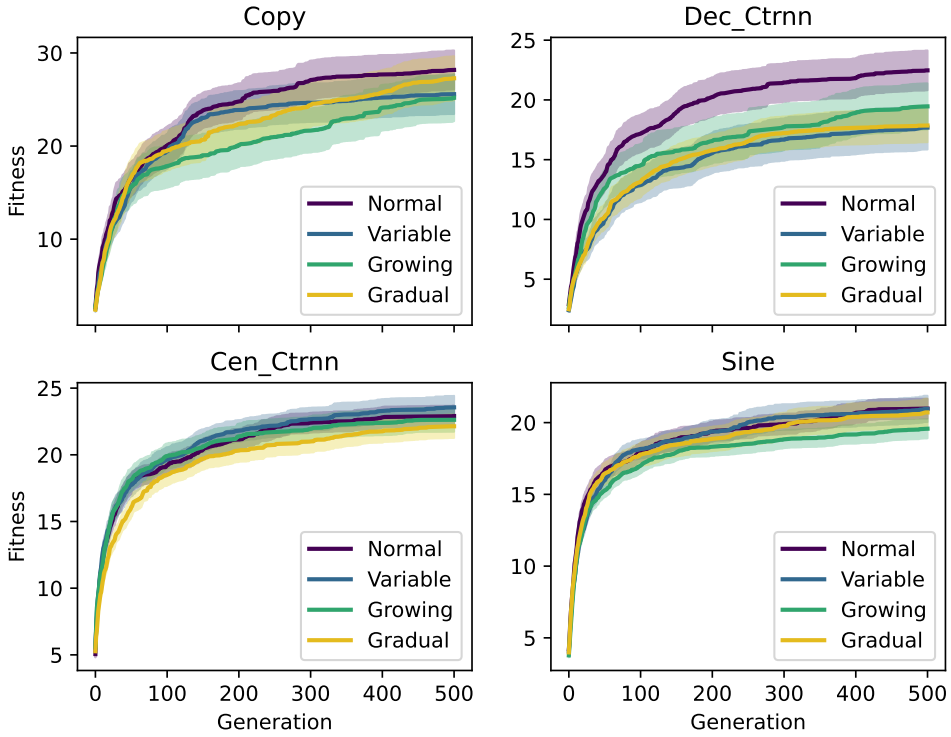
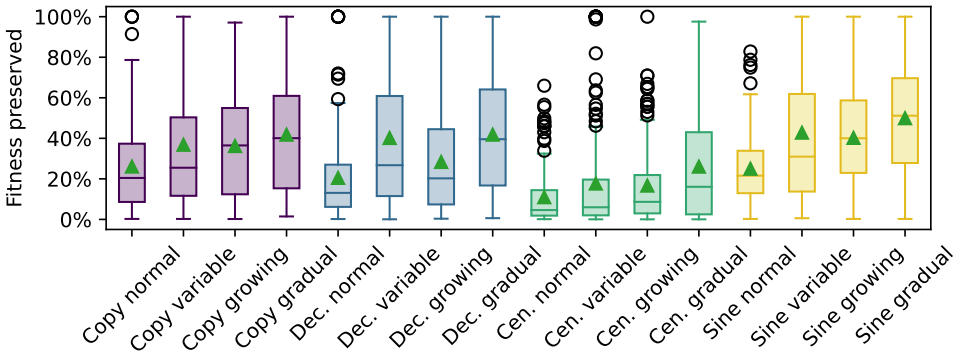


Figure 5.6: **Fitness progressions for the four different encodings with the four different controllers.**

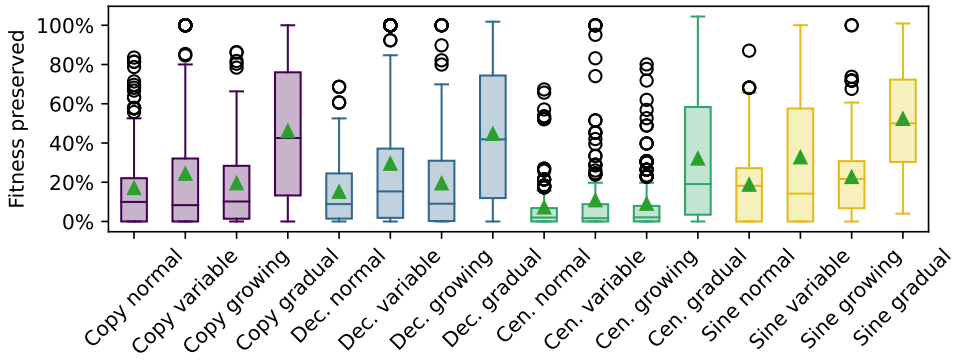
The tests showed that there were no significant differences between any of the encodings, as can be seen in the p-values in Table 5.2. Even with a less conservative correction method than Bonferroni, it does not seem that we would have found significant results. Although some of the p-values are lower, like between the normal and gradual and variable encodings for the decentralized CTRNN controller, they are not nearly low enough to reject the null hypothesis. In these cases, we would need to conduct the experiment again with a larger sample size to say anything with certainty.

	Copy	Dec. CTRNN	Cen. CTRNN	Sine
Normal vs gradual	0.877	0.054	0.481	0.587
Normal vs variable	0.605	0.069	0.742	0.577
Normal vs growing	0.365	0.311	0.783	0.056
Gradual vs variable	0.692	0.663	0.292	0.888
Gradual vs growing	0.489	0.814	0.712	0.386
Variable vs growing	0.763	0.394	0.489	0.280

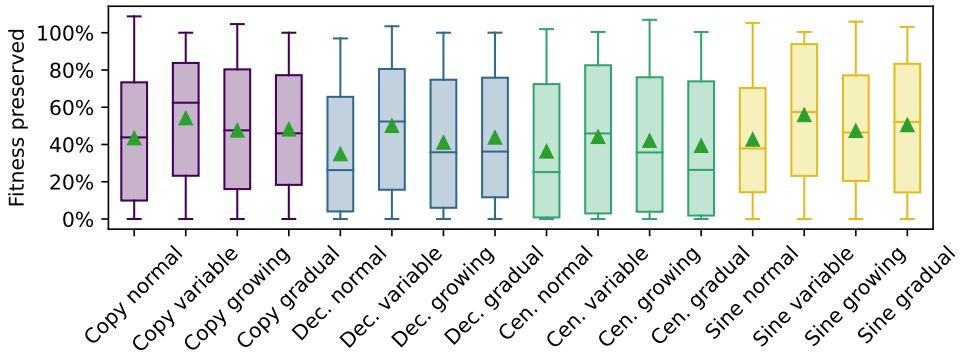
Table 5.2: **The p-values for the Mann-Whitney U tests performed for all the controllers between the different encodings.**



(a) The add mutation's effect



(b) The remove mutation's effect



(c) The scale mutation's effect

Figure 5.7: **The different mutations' effects on all individuals at generation 499.**

5.4.2 Smoothness of landscape

In the EDHMoR article of Faina et al., the rugged landscape problem was visualized by varying one evolved robot's morphology in 80 different ways and plotting the corresponding fitnesses [7]. This method of checking smoothness when varying one parameter is especially applicable to our experiment, because we can then directly check each aspect of the encodings and their effect on the fitness

landscape.

We do not, however, have any clear sense of adjacency of mutations in our system. Although different scales of the same module are adjacent in terms of morphology, it is difficult to determine the adjacency of adding, removing, or scaling different modules in a robot. Smoothness of the landscape is therefore determined here as the degree of fitness change when applying mutations. We measure this relative to the normal encoding baseline and end up with an idea of the relative landscape smoothness when conducting one mutational step.

In our case, we are interested to see if there is increased smoothness between encodings when adding and removing a module. The smoothness of scaling is also relevant to test because our encodings are based on the assumption that this will not be a disruptive mutation.

In addition, instead of checking just one creature, all the final generation individuals from all 512 runs were mutated five times with respectively only the add module, remove module, and scale mutation. This way, we can see the average fitness difference between neighbors. The scale mutation was applied to the normal encoding individuals to get a baseline for how disruptive it was for a creature that was evolved without the scale mutation. This will show us how disruptive it is with no evolution, and if there is a difference after evolution.

It also needs to be noted that because we are mutating the most evolved individuals, it is not likely that any of the mutations will be beneficial. These individuals are results of stabilized runs, and most of the mutations will have been tried before. We therefore focus on how much the fitness is preserved after mutation: The ideal would be that most of the fitness is preserved because that would mean the change was minimal and gradual. We also consider the percentage of preserved fitness, because if we just considered the preserved fitness directly, the high-performing controllers would unfairly register much more fitness loss.

The results of applying the add module mutation to each individual from each encoding and controller pair can be seen in Figure 5.7a. It appears that the variable encodings on average preserve more of the original individual's fitness compared to the normal encoding. In all the controllers, the gradual encoding preserves more than the others, averaging at 40-50% percent.

Interestingly enough, the variable encoding also preserves more than the normal encoding, with about 10% in each controller. Lastly, the centralized CTRNN controller in general do not respond well to the add module mutation, as discussed in experiment 1. There is an increase in preserved fitness for the gradual encoding, but this is likely because the gradual mutation sometimes grows the module instead of adding a new one.

On the opposite end, the remove module mutation was tested and plotted in the same way in Figure 5.7b. Here we see again that the most noticeable difference is between the gradual encoding and the others, which still averages at 40-50% fitness preservation. It appears the remove module mutation is quite detrimental and not much difference can be seen between the encodings apart from this.

Lastly, the scale mutation was tested in the same way in Figure 5.7c. The averages within controllers appear quite even, with only the variable encoding consistently preserving the most fitness. On average, around 50% is preserved in most of these encoding-controller pairs, and it appears that the scale encoding is indeed the least detrimental one of those investigated here. No obvious difference can be seen between the encodings evolved with and without the scale mutation, implying that there is no favoring of individuals that perform well even after scaling.

5.4.3 Landscape traversal

Like in experiment 1, a plot was made to depict the different controller-encoding pairs' exploration of the morphology landscape. The elite from each generation for all generations and runs were used for each controller-encoding pair. Using the same method as in experiment 1, with equation (4.1), all module positions in a robot was added together to create a new abstract coordinate representing the morphology. In addition, due to the low degree of information in the y-axis comparatively, the up direction, and mostly for illustrative purposes, the y-axis was collapsed in the plot. All 3 dimensions were used in the accompanying table, Table 5.3, counting unique voxels.

In the table, it is clear that the encodings led to some percent more morphological diversity than their normal encoding counterparts. Notably, less so for the copy controller than the others. Although this is to be expected because modules now can be any length, it is interesting that there is even an increase in the growing and gradual encodings compared to the variable encoding. Interestingly, the

Controller	Encoding	Voxels	% of normal
Copy	normal	208	
	variable	288	+38 %
	growing	309	+49 %
	gradual	304	+46 %
Dec CTRNN	normal	124	
	variable	216	+74 %
	growing	223	+80 %
	gradual	222	+79 %
Cen CTRNN	normal	85	
	variable	119	+40 %
	growing	139	+64 %
	gradual	172	+102 %
Sine	normal	117	
	variable	177	+51 %
	growing	218	+86 %
	gradual	207	+77 %

Table 5.3: **The unique voxels filled for each controller-encoding pair.** The rightmost column displays how many more percent voxels were filled for that encoding compared to the controller's normal encoding.

centralized CTRNN saw a 100% increase in voxels filled with the gradual encoding, and has a larger cluster in Figure 5.8.

Figure 5.8 shows the exploration spatially. Here we can see that in the copy and decentralized CTRNN controllers, the variable encoding seemed to explore quite far outside of the normal cluster. The decentralized CTRNN controller with the growing encoding also shows a similar pattern. Because coordinates are added together, we can tell that these morphologies were quite large. From observation, this coincides with many large and falling morphologies being observed with the variable encoding, and the poorer average performance. It still appears that some of these robots get a higher fitness, which either indicates effective falling or ineffective locomotion.

5.5 Analysis

Our findings show that there is no fitness increase with the different gradual mutations, but that there is likely smoothing of the fitness landscape. Returning to our hypothesis, we are therefore unable to confirm that a more gradual mutation might smooth the landscape and increase fitness. In fact, because performances

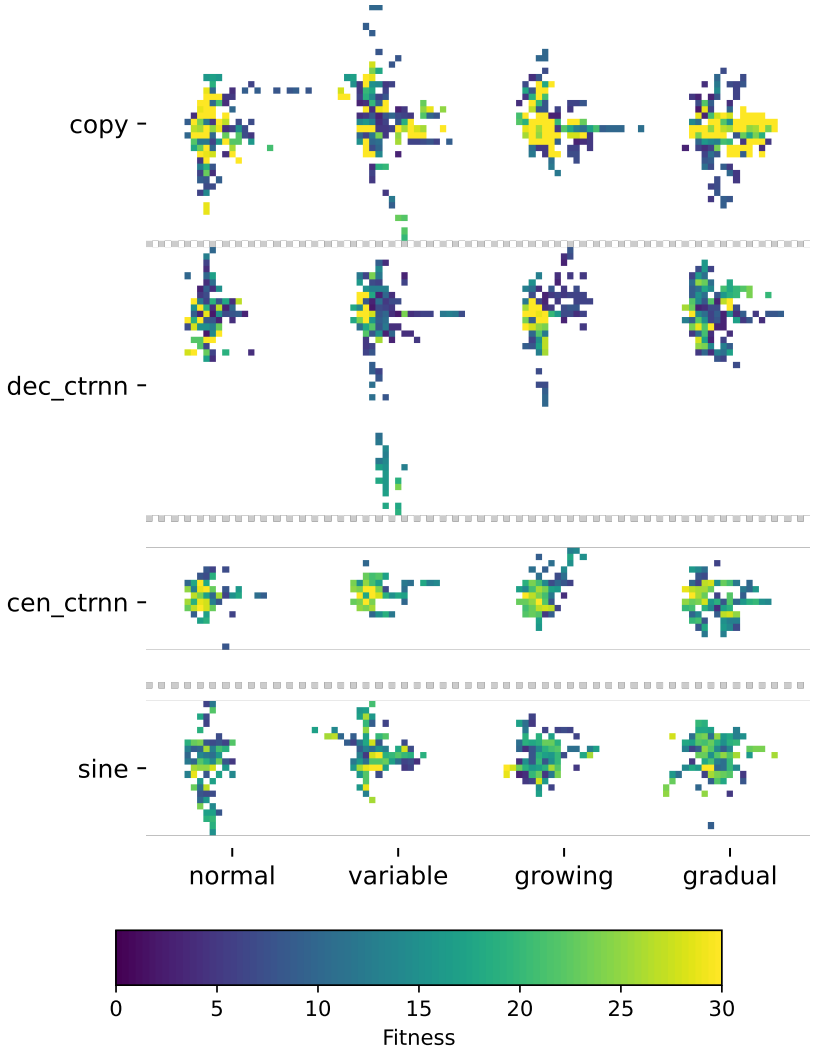


Figure 5.8: **The morphology landscape exploration by each controller-encoding pair.** The y-axis is collapsed into the xz-plane. The dotted gray line differentiate between the different controllers.

between encodings both appeared to be and were statistically indifferent from each other, these findings suggest that there is no benefit to these versions of gradual encodings. While some p-values were low enough to warrant further investigation, especially in the decentralized CTRNN controller, these trended toward decreasing fitness instead of increasing it.

Even so, there is a discernible difference in the convergence of the different encodings in the copy controller (Figure 5.6). While ending up at roughly the same fitnesses at generation 500, the growing and gradual encodings both seem to not have converged. This is especially interesting considering that the copy controller itself is

quite slow to converge. If allowed to run for longer, they might have indeed gained a fitness above the normal encoding. Regarding why we are only seeing this pattern in the copy controller, there are two immediate explanations following from experiment 1. The first is that there is simply a lot of variation in this controller, and had we performed a statistical test at generation 250, we would have found no difference. The second explanation is that the copy controller displays a lot more beneficial morphology mutations, leading to the gradual morphology encoding being allowed more mutations and thus more room to have an effect.

From experiment 1, we learned that the amount of beneficial morphology mutations was quite low. This is also a well-known trend in co-optimizing morphology and control: The morphology is rarely the focus of the optimization, as it converges early. When considering that we found no difference between the encodings, it appears that the morphology encoding simply did not have much effect. Especially the sine and centralized CTRNN controllers showed the least beneficial morphology mutations in experiment 1, and perhaps the most similar performances between encodings in experiment 2. It can therefore be argued that the effect of the encodings might not appear because they were reliant on a certain number of morphology mutations to be selected in order to be effective. In particular, the growing and gradual encodings relied on at least 3 beneficial mutations to gain a module with full actuation. In a system that already did not encourage morphology mutations, this could lead to stagnation as modules never grew out and the robot could not change size.

From the results in landscape traversal however, it appears that some of the encodings exacerbated tendencies of falling. Especially the variable encoding suffered from this in the copy and decentralized CTRNN controller, leading to large morphologies with effective falling behaviors. A possible explanation for this could be that in the beginning, when most controllers are not yet usable, the ability to increase size and mass and thus falling speed and length is selected for. This tendency was already present in all the controllers, and it could be that the encodings heightened this tendency, leading to worse average performance. Instead of disproving gradual encodings overall, this instead points to a different system perhaps being more suited to study these effects.

Chapter 6

Discussion

To sum up our findings, we have two experiments that investigate controllers and encodings when considering their search space traversal and effect on performance. This led us to conclude with the effectiveness of a control strategy using duplication and the ineffectiveness of gradual encodings in our system. Starting out, we had two hypotheses, each with two corresponding research questions. Below, these will be repeated and the answers to them summarized. Then, we will move onto a general discussion of our results, their limitations, possible future work, and lastly some ethical considerations.

6.1 Summary

Hypothesis for experiment 1: A controller that duplicates control units across the robot body will lead to more morphological diversity and therefore increased fitness. In addition, we wanted to investigate different control strategies.

The first research question connected to this is "What are the effects of a controller that duplicates control units across the morphology?". Our results show that there was a higher performance from the copy controller, that duplicated two control units across the entire body. This simple feature led to a controller that could not fine tune its behavior and could not rely on controller placement to decide actions. Instead, it had to rely on morphological development to optimize, leading to more morphological diversity found in this controller. It is likely that the evolved controllers either adopted general strategies that could work in most modules or had strategies with different expression based on detection of child modules and therefore placement in the morphology. With the

larger robot sizes and more morphological development, ultimately we saw a significant increase in performance compared to the other strategies.

The main takeaway from the above point is not simply to use copy controllers above other controllers, but it is instead an argument for controllers with duplication in general. With duplication, we believe the controller evolving to be robust to morphological changes can happen in other systems as well. Along with this, duplicated controllers also lead to fewer parameters to optimize, and a coarser sampling of the fitness landscape that makes the search space appear smaller. This in turn leads to more exploration. So far, and to the best of our knowledge, we have not seen a simple study into controllers with duplication ever be made. This can be considered an isolated case argument for why copying and reusing parts of the controller in many parts of the robot can lead to increased fitness.

The second research question was "How can a controller best be designed when co-optimizing morphology and control?". This allowed us an open exploration of different strategies of centralized and decentralized control. As mentioned above, the control strategy that led to the best performance was the copy controller, but the centralized CTRNN and the decentralized sine controller approaches both led to good results with more reliability, as they had less variation. However, we explore their lack of morphological diversity, and suggest that the reason for their quick convergence was in fact premature convergence of morphology. The naive decentralized CTRNN approach performed the worst, as it both was slow to converge and ended up at no better fitness than the sine and centralized CTRNN controllers. Because the copy controller is a simple modification to the decentralized CTRNN controller, we show that decentralized control on its own is perhaps ineffective, but that it quickly can become a strategy that competes with other control strategies. Our hope is that these findings will be of use to researchers designing their control strategies.

Hypothesis for experiment 2: A more gradual encoding can smooth the landscape and lead to better performance.

The first research question connected to this hypothesis is "How do encodings with different degrees of gradual mutation affect landscape traversal?". When applying the different add, remove, and scale module mutations in a controlled manner on the most

evolved individuals, it was found that the more gradual encodings did for the most part lead to less fitness decrease than the normal direct encoding. The gradual encoding preserved the absolute most in both remove and add node mutation. The scale mutation was not disruptive in any of the encodings. Preserving more fitness indicates to us that the fitness landscape is smoother, where mutations are less disruptive and do not lead to large fitness drops. Overall, it appears that the gradual encoding led to smoother traversal of the landscape.

The second research question is "How do encodings with different degrees of gradual mutation affect performance?" After implementing and testing several encodings with varying degrees of gradual morphology mutation, we found no significant difference between them in performance when compared to our baseline direct encoding. We hypothesize that overall, due to the low degree of morphological development in the system, the different encodings did not have any effect. These findings are somewhat inconclusive due to there being some doubt as to whether all controller-encoding pairs ran to convergence. Consequently, this research question would benefit from being revisited with a system with more morphological development and that makes sure evolution plateaus.

6.2 Further discussion

In general, these two experiments present very different methods for traversing the fitness landscape, which can explain why they did or did not increase performance. In short, the controller experiments highlighted the link between coarse sampling and more exploration, and the encoding experiments showed pitfalls of smoother sampling and local optima. This will further be explored below.

In Figure 6.1 and Figure 6.2, a comparison is made between the imagined different samplings of the fitness function,

$$fitness = f(p(g))$$

where g is the robot genotype, p is the function to generate a phenotype from a genotype, and f is the function by which we evaluate it. In Figure 6.1, we imagine that the x-axis consists of several more or less unordered phenotypes, as the actual space of $p(g)$ is so large that it cannot be illustrated. In this specific example, we imagine an instance where we start in phenotype A, with controller and morphology A. Close to us in the fitness landscape,

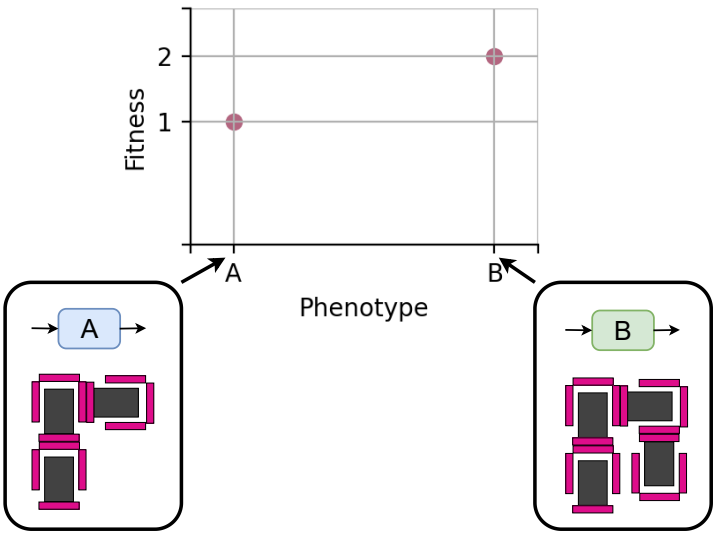
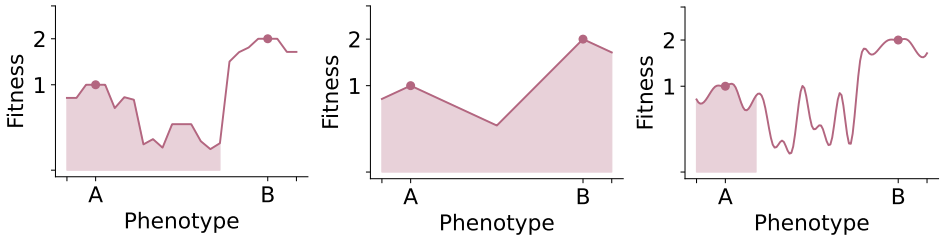


Figure 6.1: **An imagined scenario of two phenotypes in a fitness landscape.**



(a) **Baseline** (b) **Copy controller** (c) **Gradual encoding**

Figure 6.2: **The different imagined samplings.** The shaded area shows the reachable space with the mutation rate and granularity of sampling.

there is a more fit phenotype B, with controller and morphology B. The difference between the morphologies is just the addition of one module, and the difference in controller is one that can control morphology A and one that can control morphology B. This is an isolated case of needing a morphology mutation and needing the controller to follow suit.

With a normal direct encoding and a normal controller, perhaps a decentralized sine controller or our centralized CTRNN controller, the landscape will be sampled like in Figure 6.2a. Between phenotype A and B, there are some phenotypes that perform quite poorly, as either the controller is maladjusted to the morphology or vice versa. On the opposite side of this fitness minimum, there are phenotypes where the controller and morphology cooperate and perform better than phenotype A. However, because a search is

unlikely to accept traversing this fitness minimum, such a sampling is ineffective in exploring the space and finding higher fitness.

With the copy controller, the controller space is vastly truncated because of the relatively few parameters (Figure 6.2b). The sampling becomes coarser, and the reachable samples through mutation with the given mutation rate becomes larger. Between morphology A and B, there is now only a few maladjusted phenotypes, and phenotype B is reachable within a few generations. In some cases, because the controller is optimized to work in many different controllers, controller A is equal to controller B, and phenotype B can be reached with a simple add node mutation.

On the opposite end, the gradual encoding increases the granularity of the morphology space sampling. Because modules can be of any length, there are an infinite number of morphologies between A and B (when ignoring that floating-point representation is limited). This has two main effects: 1) Morphologies that lead to only a minor detriment to performance with the given controller exists within reach, and 2) the number of steps needed to get to phenotype B vastly increases. We imagine this encoding as being able to reveal stepping stones in the performance valley between phenotype A and B (Figure 6.2c) that the search can use to cross, again lessening the deceptive nature of the search. However, as the starting position might have more solid neighboring phenotypes, the few solutions attempting to cross the gap might find themselves in unstable territory, with most of their mutational offspring dying off to unusable changes. Because of this, it might simply be that revealing stepping stones at the cost of increasing the distance will never work. After assessing this, the gradual encoding approach seems to have the same issues as the normal encoding in crossing deceptive landscapes, and it seems not unreasonable that they would gain the same performance. Even so, because of the smoother sampling, the gradual encoding should be more able to climb local optima.

Although this example is imagined, it illustrates the importance of sampling when considering a search's traversal. When constructing a robotic system to co-optimize, you are defining a morphology and controller space, that together create the phenotype space. This will define what robots you can achieve with this system. Further, the granularity will have a lot to say about the search, and from our findings it might appear that a coarser sampling will allow better fitness over all because it explores better. This is congruent with the

general notion that fewer parameters to optimize will be easier to optimize. Still, it seems reasonable that once the search arrives at a good solution, the granularity can be increased in order to locally explore and find the absolute optimum [27]. This means that it can be beneficial to switch from the normal encoding to the gradual in order to climb a hill locally, and to switch from the copy controller to the naive decentralized CTRNN controller in order to specialize each controller with the given morphology.

6.3 Limitations and future work

Although the results are promising, they are of course limited by generalizability: We have tested these concepts in our system, which itself has many features which may muddle the conclusions. Additionally, some aspects of how the experiments were conducted could have been improved. All limitations, and how they could be amended, as well as other future work will be presented here.

Even though we were able to get a good amount of sample runs, the number of generations were limited by resources. After viewing the lack of convergence in the initial experiments, number of generations were increased from 200 to 500 in the final runs. Even so, we do not observe full convergence in some of the controller-encoding pairs. Most notably the copy and decentralized CTRNN controllers lack convergence in both experiments, with beneficial changes still occurring and the average fitness still rising.

In experiment 1, this is mostly consequential for the conclusion that the decentralized CTRNN is statistically on par with the sine and centralized CTRNN controllers. It might be that, given more generations, it would prove to rise above them in performance. Still, from the level of convergence seen, it is unlikely that any of the conclusions reached would change. It is also worth noting that the copy controller's lack of convergence offers the possibility of gaining even higher performance if the needed computational resources are available. However, in experiment 2, the lack of convergence in the copy's gradual and growing encodings implies that the conclusion might have changed given more generations. It might be that this controller in conjunction with the gradual encodings could produce even higher fitness because of its increased amount of morphological development. We recognize that this is a limitation of the experiment, and that further research into these

encodings might indeed reveal a statistical difference between them.

As in any such experiment conducted only on one system, these results would benefit from being verified on a different robotic system. The EMeRGE modules seemed to be prone to local optima: Its square corners and large flat surfaces provided pushing, grabbing, and balancing in a way other modules likely would not. The balancing especially led to some interesting strategies in order to still exploit the falling optima (as in the cartwheel showed in Figure 3.6). In addition, although it was deemed usable for the purposes of these experiments, the modules were perhaps too strong in simulation. This created some less realistic jumping behaviors, especially in the sine controller. This could be fixed with further tweaking of the weight to force ratio of the modules, perhaps by using the real module values. All in all, implementing the same strategies on a different robotic system would help confirm these findings and support their transferability to other systems. An option for this could be the RoboGrammar modules [87], that have far less angular modules that could provide a very different set of behaviors.

The use of sensors was not the main focus in these experiments, however further incorporating them and repeating the controller experiment seems a worthwhile endeavor. Here, more realistic sensors closer to those described in the original EMeRGE paper could be implemented [84]. Because the copy controller lacks the ability to specialize module networks, it might instead have specialization of modules through detection of placement through sensors. This would cause the different modules to adopt different roles, making the system potentially very robust to damage. In an investigation centered around this controller and its use of sensors, it could be determined whether this actually occurs. The copy controller in a modular robot could also be examined through the view of swarm robotics and determine if it can achieve the benefits usually associated with swarms.

For more future work, we could study the controllers and encodings when using a quality diversity algorithm instead of a standard EA. When considering the landscape traversal discussed above, we notice that especially the problems discussed in relation to the gradual encoding would diminish with an algorithm like MAP-elites [28]. Because it has bigger potential to discover morphological niches, and because MAP-elites keeps niches no matter how unstable

their champion solution are, this algorithm might be what the gradual encoding needs to excel. In relation to the controllers, it would be useful to see if the exploration by the copy controller still dominates when using an algorithm that encourages exploration so heavily.

6.4 Ethical considerations

In this thesis and the work surrounding it, we hope to contribute to a future where robotics is a normal part of everyday life. Although our work is exploring the foundation for which more commercially available robotics can be built upon, we still feel that because our aim is tied to this vision, we would be remiss if we did not address some ethical issues this might lead to.

In particular, a normal concern when it comes to robotics is how it can replace human workers and lead to unemployment, and thus economic decline. As tools gain autonomy, human workers can be replaced with machines. Although the machines will not have human expertise at first, progress in technology will soon leave even fairly complicated intellectual work in the hands of a robot. An example of this happening already is AI automatically producing code on par with humans [88]. Machines require far less pay and other goods, and so employers will be encouraged to choose this option. If this is not handled thoughtfully, it will lead to economic decline [89] and perhaps economic inequality [90].

Although there are many arguments as to why robotics might in the future lead to economic decline, there are also theories of how it can be managed or not happen at all [89, 90]. It is also important to keep in mind that robots will fill roles we cannot or do not want humans to have. For example, robots can replace dangerous human jobs, like rescue work in dangerous areas, deep sea exploration and maintenance, and so on. And more commonly, robots can be used in the home to take care of sick or ageing people, allowing these people an autonomy and mobility they currently cannot have.

In short, with the rise of robotics we are entering a time of rapid technological change, and it is difficult to see where it might go from here. It is still important to keep in mind that proper legislation and other measures might have to be used preemptively to avoid the new issues presented with automation. Hopefully, robotics will soon become so advanced that we do have to think about these problems.

Chapter 7

Conclusion

Search space traversal and morphological evolvability are still largely unresolved challenges in co-optimizing morphology and control. Due to premature convergence of morphology and other difficulties in properly exploring the search space, the field is currently experiencing stagnation. Therefore, this thesis had the goal of determining how and why morphological evolvability and performance could be increased. In particular, we designed experiment 1 with the goal of investigating controllers' performance and traversal of the morphological landscape, as well as suggesting the copy controller strategy. We designed experiment 2 with the goal of investigating how the landscape could be smoothed using gradual encodings, and if this would lead to a performance increase.

In experiment 1, we found that the copy controller performed significantly better than the other approaches. Because it only had two networks to use for all the modules' control, these networks had to work in many modules and thus ended up being very robust to morphological changes. This led to it experiencing more morphological change and exploring more of the morphological search space. This suggests that a decentralized approach using duplication of control units is more suited for control in systems like ours. While we therefore present the copy controller as a viable control strategy with beneficial properties, we also suggest that this indicates the benefits of duplication in controllers at large.

In experiment 2, we found that there was statistically no advantage in performance to encodings with different degrees of gradual mutation. While they did smooth the landscape, this was not reflected in their performance, as we could find no difference between the baseline encoding and the gradual encodings. We hypothesize that because there were few beneficial morphology mutations in the

system, the encodings had little opportunity to have any effect on the individual.

Overall, this thesis contributes both the copy controller and the gradual encoding to the research community. The copy controller exhibits continued morphological diversification and higher performance, while also making a case for reuse of control units in controllers at large. The gradual encoding has no significant performance increase, but because it does smooth the landscape, it has potential uses in future work. For example, because of the increased granularity, it has more opportunity to discover niches, which could be useful in a quality diversity algorithm like MAP-elites. However, there would have to be more morphological development in the system, because if the body is simply not optimized, this morphology encoding will not be different from a naive morphology encoding.

In relation to early convergence of morphology, our findings help confirm that controllers that are more robust to morphological changes will lead to more morphological evolvability. This further suggests that designing for such robustness, with duplication or for example through treating the modules as homogeneous swarm agents, might be one way to decrease premature convergence of morphology.

Our findings also highlight the importance of sampling. As we have seen, coarser samplings lead to increased exploration and less fine-tuning, while a more granular sampling, from the gradual encoding or centralized CTRNN controller, causes the search to get stuck on local optima. As this is congruent with the findings of other works, we further emphasize that it seems to be more important to be able to explore than to fine-tune, and that in designing a system you often must choose between them. However, if a beneficial behavior is reached, there is potential to switch to fine-tuning by more gradual search or even switch to a different optimization strategy like reinforcement learning.

With these conclusions, we hope to contribute to future research in co-optimized modular robots. For modular robots to live up their promise of being versatile, reconfigurable tools, it is especially important that morphology is not sidelined in optimizations, since reconfigurability is their main draw. Therefore, it is important that optimization gets better at traversing the fitness landscape, so that we will see evolvable modular robots with more morphological

diversity. As we continue to work on landscape traversal, we hope to see even more scalable and complex modular robots. It may be that with the progress of nanorobotics, the groundwork being done on modular robots now will see its use in small, cell-like modules that create increasingly large and life-like robotics. Perhaps then, co-optimized modular robotics will enter mainstream robotics, or simply be another explored branch of science that we can hopefully learn something from.

Bibliography

[1] K. Sims, "Evolving virtual creatures," in *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1994*, pp. 15–22, 1994.

[2] M. Joachimczak, R. Suzuki, and T. Arita, "Artificial Metamorphosis: Evolutionary Design of Transforming, Soft-Bodied Robots," *Artificial Life*, vol. 22, no. 3, 2016.

[3] G. S. Hornby and J. B. Pollack, "Evolving L-systems to generate virtual creatures," *Computers and Graphics (Pergamon)*, vol. 25, no. 6, 2001.

[4] N. Cheney, R. MacCurdy, J. Clune, and H. Lipson, "Unshackling evolution: Evolving soft robots with multiple materials and a powerful generative encoding," *SIGEVOlution*, vol. 7, p. 11–23, aug 2014.

[5] J. E. Auerbach and J. C. Bongard, "Evolving complete robots with cppn-neat: the utility of recurrent connections," in *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pp. 1475–1482, 2011.

[6] H. Lipson and J. B. Pollack, "Automatic design and manufacture of robotic lifeforms," *Nature*, vol. 406, no. 6799, 2000.

[7] A. Faíña, F. Bellas, F. López-Peña, and R. J. Duro, "EDHMoR: Evolutionary designer of heterogeneous modular robots," *Engineering Applications of Artificial Intelligence*, 2013.

[8] M. Hale, E. Buchanan Berumen, A. Winfield, J. Timmis, E. Hart, G. Eiben, W. Li, and A. Tyrrell, "The are robot fabricator: How to (re) produce robots that can evolve in the real world," in *International Society for Artificial Life: ALIFE2019*, pp. 95–102, York, 2019.

[9] Y. Kawauchi, T. Fukuda, and M. Inaba, “"a strategy of self-organization for cellular robotic system (cebot)",” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3, pp. 1558–1565, IEEE, 1992.

[10] S. Murata, H. Kurokawa, and S. Kokaji, “Self-assembling machine,” in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pp. 441–448, IEEE, 1994.

[11] M. Yim, D. G. Duff, and K. D. Roufas, “PolyBot: a modular reconfigurable robot,” in *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 1, 2000.

[12] K. Stoy, D. Brandt, D. J. Christensen, and D. Brandt, *Self-reconfigurable robots: an introduction*. Mit Press Cambridge, 2010.

[13] R. Dawkins, *The Blind Wathcmaker. Why the Evidence of Evolution Reveals a Universe without Design*. 1996.

[14] J. Lehman and K. O. Stanley, “Evolving a diversity of creatures through novelty search and local competition,” in *Genetic and Evolutionary Computation Conference, GECCO'11*, 2011.

[15] A. Wagner, “The role of robustness in phenotypic adaptation and innovation,” *Proceedings of the Royal Society B: Biological Sciences*, vol. 279, no. 1732, pp. 1249–1258, 2012.

[16] M. Kirschner and J. Gerhart, “Evolvability,” *Proceedings of the National Academy of Sciences*, vol. 95, no. 15, pp. 8420–8427, 1998.

[17] F. Veenstra, P. G. de Prado Salas, K. Stoy, J. Bongard, and S. Risi, “Death and progress: How evolvability is influenced by intrinsic mortality,” *Artificial life*, vol. 26, no. 1, pp. 90–111, 2020.

[18] J. Lehman and K. O. Stanley, “Evolvability is inevitable: Increasing evolvability without the pressure to adapt,” *PloS one*, vol. 8, no. 4, p. e62186, 2013.

[19] G. P. Wagner and L. Altenberg, “Perspective: complex adaptations and the evolution of evolvability,” *Evolution*, vol. 50, no. 3, pp. 967–976, 1996.

[20] C. Liu, J. Liu, R. Moreno, F. Veenstra, and A. Faina, “The impact of module morphologies on modular robots,” in *2017 18th International Conference on Advanced Robotics, ICAR 2017*, 2017.

[21] R. Moreno and A. Faina, "Using evolution to design modular robots: An empirical approach to select module designs," in *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*, pp. 276–290, Springer, 2020.

[22] N. Cheney, J. Bongard, V. SunSpiral, and H. Lipson, "On the difficulty of co-optimizing morphology and control in evolved virtual creatures," in *Proceedings of the Artificial Life Conference 2016, ALIFE 2016*, 2016.

[23] S. Kriegman, N. Cheney, and J. Bongard, "How morphological development can guide evolution," *Scientific reports*, vol. 8, no. 1, pp. 1–10, 2018.

[24] N. Cheney, R. MacCurdy, J. Clune, and H. Lipson, "Unshackling evolution: Evolving soft robots with multiple materials and a powerful generative encoding," in *GECCO 2013 - Proceedings of the 2013 Genetic and Evolutionary Computation Conference*, 2013.

[25] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[26] A. E. Eiben, J. E. Smith, *et al.*, *Introduction to evolutionary computing*, vol. 53. Springer, 2003.

[27] F. Veenstra and K. Glette, "How Different Encodings Affect Performance and Diversification when Evolving the Morphology and Control of 2D Virtual Creatures," in *The 2020 Conference on Artificial Life*, 2020.

[28] J.-B. Mouret and J. Clune, "Illuminating search spaces by mapping elites," *arXiv preprint arXiv:1504.04909*, 2015.

[29] J. Nordmoen, F. Veenstra, K. O. Ellefsen, and K. Glette, "Quality and Diversity in Evolutionary Modular Robotics," in *2020 IEEE Symposium Series on Computational Intelligence, SSCI 2020*, 2020.

[30] F. Veenstra, J. Jørgensen, and S. Risi, "Evolution of fin undulation on a physical knifefish-inspired soft robot," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 157–164, 2018.

[31] D. J. Christensen, "Evolution of shape-changing and self-repairing control for the atron self-reconfigurable robot," in

Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006., pp. 2539–2545, IEEE, 2006.

[32] T. F. Nygaard, J. Nordmoen, K. O. Ellefsen, C. P. Martin, J. Tørresen, and K. Glette, "Experiences from real-world evolution with DYRET: Dynamic robot for embodied testing," in *Communications in Computer and Information Science*, vol. 1056 CCIS, 2019.

[33] A. J. Ijspeert, "Central pattern generators for locomotion control in animals and robots: A review," *Neural Networks*, 2008.

[34] R. Pfeifer and J. Bongard, *How the body shapes the way we think: a new view of intelligence*. MIT press, 2006.

[35] J. C. Bongard, "Evolutionary Robotics: Taking a biologically inspired approach to the design of autonomous, adaptive machines," 2013.

[36] C. Paul, "Morphological computation. A basis for the analysis of morphology and control requirements," *Robotics and Autonomous Systems*, vol. 54, no. 8, 2006.

[37] S. Kriegman, "Why virtual creatures matter," *Nature Machine Intelligence*, vol. 1, no. 10, pp. 492–492, 2019.

[38] M. Jelisavcic, K. Glette, E. Haasdijk, and A. E. Eiben, "Lamarckian Evolution of Simulated Modular Robots," *Frontiers in Robotics and AI*, 2019.

[39] G. HINTON, "How learning can guide evolution," *Complex Systems*, vol. 1, 1987.

[40] A. Gupta, S. Savarese, S. Ganguli, and L. Fei-Fei, "Embodied intelligence via learning and evolution," *arXiv preprint arXiv:2102.02202*, 2021.

[41] F. Veenstra, A. Faina, S. Risi, and K. Stoy, "Evolution and morphogenesis of simulated modular robots: A comparison between a direct and generative encoding," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10199 LNCS, 2017.

[42] G. Lan, M. De Carlo, F. van Diggelen, J. M. Tomczak, D. M. Roijers, and A. E. Eiben, "Learning directed locomotion in modular robots with evolvable morphologies," 2020.

[43] G. S. Hornby, H. Lipson, and J. B. Pollack, "Generative representations for the automated design of modular physical robots," *IEEE transactions on Robotics and Automation*, vol. 19, no. 4, pp. 703–719, 2003.

[44] K. O. Stanley, "Compositional pattern producing networks: A novel abstraction of development," *Genetic Programming and Evolvable Machines*, vol. 8, no. 2, 2007.

[45] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci, "A hypercube-based encoding for evolving large-scale neural networks," *Artificial Life*, vol. 15, no. 2, 2009.

[46] F. Gruau and U. C. B.-l. I, "Thesis Neural Network Synthesis Using Cellular Encoding and the Genetic Algorithm," *Synthesis*, 1994.

[47] M. Joachimczak, R. Suzuki, and T. Arita, "Fine grained artificial development for body-controller coevolution of soft-bodied animats," in *Artificial Life 14 - Proceedings of the 14th International Conference on the Synthesis and Simulation of Living Systems, ALIFE 2014*, 2014.

[48] M. Joachimczak, R. Suzuki, and T. Arita, "Improving evolvability of morphologies and controllers of developmental soft-bodied robots with novelty search," *Frontiers Robotics AI*, 2015.

[49] E. Samuelsen, K. Glette, and J. Torresen, "A hox gene inspired generative approach to evolving robot morphology," in *GECCO 2013 - Proceedings of the 2013 Genetic and Evolutionary Computation Conference*, 2013.

[50] R. A. Brooks, "A Robust Layered Control System For A Mobile Robot," *IEEE Journal on Robotics and Automation*, vol. 2, no. 1, 1986.

[51] K. Støy, W. M. Shen, and P. M. Will, "Using role-based control to produce locomotion in chain-type self-reconfigurable robots," *IEEE/ASME Transactions on Mechatronics*, vol. 7, no. 4, 2002.

[52] A. Sproewitz, A. Billard, P. Dillenbourg, and A. J. Ijspeert, "Roombots-mechanical design of self-reconfiguring modular robots for adaptive furniture," in *2009 IEEE international conference on robotics and automation*, pp. 4259–4264, IEEE, 2009.

[53] J. Rieffel, F. Saunders, S. Nadimpalli, H. Zhou, S. Hassoun, J. Rife, and B. Trimmer, "Evolving soft robotic locomotion in physx," in *Proceedings of the 11th annual conference companion on genetic and evolutionary computation conference: Late breaking papers*, pp. 2499–2504, 2009.

[54] R. D. Beer, "On the dynamics of small continuous-time recurrent neural networks," *Adaptive Behavior*, vol. 3, no. 4, pp. 469–509, 1995.

[55] R. D. Beer, "The dynamics of adaptive behavior: A research program," *Robotics and Autonomous Systems*, vol. 20, no. 2-4, pp. 257–289, 1997.

[56] R. D. Beer, "The dynamics of brain–body–environment systems: A status report," *Handbook of Cognitive Science*, pp. 99–120, 2008.

[57] J. Lehman and K. O. Stanley, "Improving evolvability through novelty search and self-adaptation," in *2011 IEEE congress of evolutionary computation (CEC)*, pp. 2693–2700, IEEE, 2011.

[58] E. Samuelsen and K. Glette, "Real-world reproduction of evolved robot morphologies: Automated categorization and evaluation," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9028, 2015.

[59] A. P. Sabelhaus, J. Bruce, K. Caluwaerts, P. Manovi, R. F. Firoozi, S. Dobi, A. M. Agogino, and V. SunSpiral, "System design and locomotion of superball, an untethered tensegrity robot," in *2015 IEEE international conference on robotics and automation (ICRA)*, pp. 2867–2873, IEEE, 2015.

[60] N. Cheney, J. Clune, and H. Lipson, "Evolved electrophysiological soft robots," in *Artificial Life 14 - Proceedings of the 14th International Conference on the Synthesis and Simulation of Living Systems, ALIFE 2014*, 2014.

[61] J. Collins, S. Chand, A. Vanderkop, and D. Howard, "A review of physics simulators for robotic applications," *IEEE Access*, 2021.

[62] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. d. L. Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, *et al.*, "Deepmind control suite," *arXiv preprint arXiv:1801.00690*, 2018.

[63] T. Erez, Y. Tassa, and E. Todorov, "Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx," in *2015 IEEE international conference on robotics and automation (ICRA)*, pp. 4397–4404, IEEE, 2015.

[64] S. Nolfi, J. Bongard, P. Husbands, and D. Floreano, "Evolutionary robotics," in *Springer Handbook of Robotics*, pp. 2035–2068, Springer, 2016.

[65] N. Jakobi, P. Husbands, and I. Harvey, "Noise and the reality gap: The use of simulation in evolutionary robotics," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 929, 1995.

[66] O. Miglino, H. H. Lund, and S. Nolfi, "Evolving mobile robots in simulated and real environments," *Artificial life*, vol. 2, no. 4, pp. 417–434, 1995.

[67] J. C. Bongard and H. Lipson, "Nonlinear system identification using coevolution of models and tests," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 4, pp. 361–384, 2005.

[68] S. Koos, J. B. Mouret, and S. Doncieux, "Crossing the reality gap in evolutionary robotics by promoting transferable controllers," in *Proceedings of the 12th Annual Genetic and Evolutionary Computation Conference, GECCO '10*, 2010.

[69] J. Seo, J. Paik, and M. Yim, "Modular Reconfigurable Robotics," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 2, no. 1, 2019.

[70] S. Murata, E. Yoshida, A. Kamimura, H. Kurokawa, K. Tomita, and S. Kokaji, "M-TRAN: Self-reconfigurable modular robotic system," *IEEE/ASME Transactions on Mechatronics*, vol. 7, no. 4, 2002.

[71] A. Brunete, A. Ranganath, S. Segovia, J. P. de Frutos, M. Hernando, and E. Gambao, "Current trends in reconfigurable modular robots design," 2017.

[72] R. J. Alattas, S. Patel, and T. M. Sobh, "Evolutionary Modular Robotics: Survey and Analysis," *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 95, no. 3-4, 2019.

[73] J. W. Romanishin, K. Gilpin, and D. Rus, "M-blocks: Momentum-driven, magnetic modular robots," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4288–4295, IEEE, 2013.

[74] V. Zykov, E. Mytilinaios, B. Adams, and H. Lipson, "Self-reproducing machines," *Nature*, vol. 435, no. 7039, pp. 163–164, 2005.

[75] A. Brunete, M. Hernando, E. Gambao, and J. E. Torres, "A behaviour-based control architecture for heterogeneous modular, multi-configurable, chained micro-robots," *Robotics and Autonomous Systems*, vol. 60, no. 12, pp. 1607–1624, 2012.

[76] D. Zappetti, S. Mintchev, J. Shintake, and D. Floreano, "Bio-inspired tensegrity soft modular robots," in *Conference on Biomimetic and Biohybrid Systems*, pp. 497–508, Springer, 2017.

[77] D. Rus and M. Vona, "Crystalline robots: Self-reconfiguration with compressible unit modules," *Autonomous Robots*, vol. 10, no. 1, pp. 107–124, 2001.

[78] D. Marbach and A. J. Ijspeert, "Co-evolution of configuration and control for homogenous modular robots," in *Proceedings of the eighth conference on intelligent autonomous systems (IAS8)*, pp. 712–719, IOS Press, 2004.

[79] R. Groß, M. Bonani, F. Mondada, and M. Dorigo, "Autonomous self-assembly in swarm-bots," *IEEE transactions on robotics*, vol. 22, no. 6, pp. 1115–1130, 2006.

[80] J. Davey, N. Kwok, and M. Yim, "Emulating self-reconfigurable robots-design of the smores system," in *2012 IEEE/RSJ international conference on intelligent robots and systems*, pp. 4464–4469, IEEE, 2012.

[81] W. Tan, H. Wei, and B. Yang, "Sambotii: a new self-assembly modular robot platform based on sambot," *Applied Sciences*, vol. 8, no. 10, p. 1719, 2018.

[82] T. Tosun, G. Jing, H. Kress-Gazit, and M. Yim, "Computer-aided compositional design and verification for modular robots," in *Robotics Research*, pp. 237–252, Springer, 2018.

[83] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, *et al.*, "Unity: A general platform for intelligent agents," *arXiv preprint arXiv:1809.02627*, 2018.

[84] R. Moreno, C. Liu, A. Faina, H. Hernandez, and J. Gomez, "The emerge modular robot, an open platform for quick testing of evolved robot morphologies," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 71–72, 2017.

[85] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, jul 2012.

[86] J. Lehman, J. Clune, D. Misevic, C. Adami, L. Altenberg, J. Beaulieu, P. J. Bentley, S. Bernard, G. Beslon, D. M. Bryson, *et al.*, "The surprising creativity of digital evolution: A collection of anecdotes from the evolutionary computation and artificial life research communities," *Artificial life*, vol. 26, no. 2, pp. 274–306, 2020.

[87] A. Zhao, J. Xu, M. Konaković Luković, J. Hughes, A. Speilberg, D. Rus, and W. Matusik, "Robogrammar: Graph grammar for terrain-optimized robot design," *ACM Transactions on Graphics (TOG)*, vol. 39, no. 6, pp. 1–16, 2020.

[88] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. D. Lago, *et al.*, "Competition-level code generation with alphacode," *arXiv preprint arXiv:2203.07814*, 2022.

[89] S. G. Benzell, L. J. Kotlikoff, G. LaGarda, and J. D. Sachs, "Robots are us: Some economics of human replacement," tech. rep., National Bureau of Economic Research, 2015.

[90] A. Berg, E. F. Buffie, and L.-F. Zanna, "Robots, growth, and inequality," *Finance & Development*, vol. 53, no. 3, pp. 10–13, 2016.

Appendix A

For Conferences

Appended are an article and a poster abstract that was created as a result of this thesis.

Article for the Conference of Artificial Life (ALIFE): "Centralized and Decentralized Control in Modular Robots and Their Effect on Morphology". At the time of writing, it is accepted to be published in the proceedings and for virtual oral presentation. It will change before publication because the corrections suggested by the peer reviewers are not fully implemented at the time of thesis delivery.

Poster abstract for the Advanced Course and Symposium on Artificial Intelligence and Neuroscience (ACAIN): "Exploring the Effects of Centralized and Decentralized Control on Morphology and Performance". At the time of writing, it is under review.

Centralized and Decentralized Control in Modular Robots and Their Effect on Morphology

Mia-Katrin Kvalsund[1], Kyrre Glette[1,2] and Frank Veenstra[1]

[1]Department of Informatics, University of Oslo, Norway
[2]RITMO, University of Oslo, Norway
mia.kvalsund@gmail.com

Abstract

In Evolutionary Robotics, evolutionary algorithms are used to co-optimize morphology and control. However, co-optimizing leads to different challenges: How do you optimize a controller for a body that often changes its number of inputs and outputs? Researchers must then make some choice between centralized or decentralized control. In this article, we study the effects of centralized and decentralized controllers on modular robot performance and morphologies. This is done by implementing one centralized and two decentralized continuous time recurrent neural network controllers, as well as a sine wave controller for a baseline. We found that a decentralized approach that was more independent of morphology size performed significantly better than the other approaches. It also worked well in a larger variety of morphology sizes. In addition, we highlighted the difficulties of implementing centralized control for a changing morphology, and saw that our centralized controller struggled more with early convergence than the other approaches. Our findings indicate that duplicated decentralized networks are beneficial when evolving both the morphology and control of modular robots. Overall, if these findings translate to other robot systems, our results and issues encountered can help future researchers make a choice of control method when co-optimizing morphology and control.

Introduction

When co-optimizing morphology and control of modular robots, how do we optimize a controller for a robot that often changes its number of actuators and sensors? If a centralized approach is chosen, we must select a method to deal with disappearing actuators or the addition of new ones. Furthermore, although distributed control removes the issues of changing morphology, we must still then facilitate for global synchronization of the actuators. In this paper, we implement and discuss a few approaches to control when co-optimizing morphology and control, and investigate what effects they have on morphology.

Throughout Evolutionary Robotics' short history, there have been many approaches to co-optimizing morphology and control. Most notably, the work of Karl Sims showed virtual creatures evolved using a nested graph where body and control elements were connected (Sims, 1994). Controllers were duplicated as body parts were copied in a semi-distributed approach. Lipson and Pollack (2000) later showed a pipeline to transfer such virtual creatures to reality, where they used centralized control. Later, Cheney et al. (2013, 2014) displayed soft-robots that evolved control through an indirect encoding using a Compositional Pattern Producing Network (CPPN). Similarly, Auerbach and Bongard (2011) used a CPPN encoding to generate a morphology and weights for a centralized continuous time recurrent neural network (CTRNN) controller. Both these approaches have reuse of control elements due to the indirect encodings.

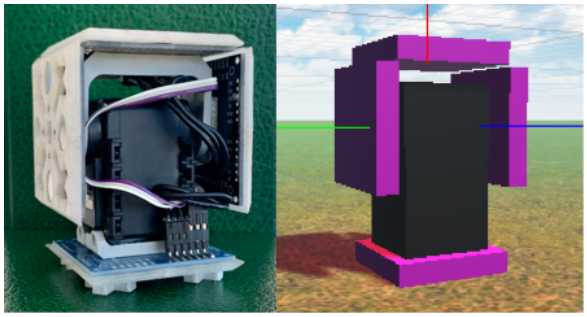


Figure 1: **The EMeRGE module**. The left image is an example of a real EMeRGE module, and the right is our Unity simplification.

A common problem when co-optimizing morphology and control is that of early convergence of morphology only. As described by Joachimczak et al. (2016), and later explored further by Cheney et al. (2016), the morphology will reach its almost final form relatively early. Cheney et al. theorize that because the controller interacts with the environment through the interface of the body, changes to the body will scramble the control. However, this effect has not been studied much in modular robots. Decentralized control in modular robots could potentially make this less of a problem because adding a new module with the same controller as its parent could still work without scrambling the overall

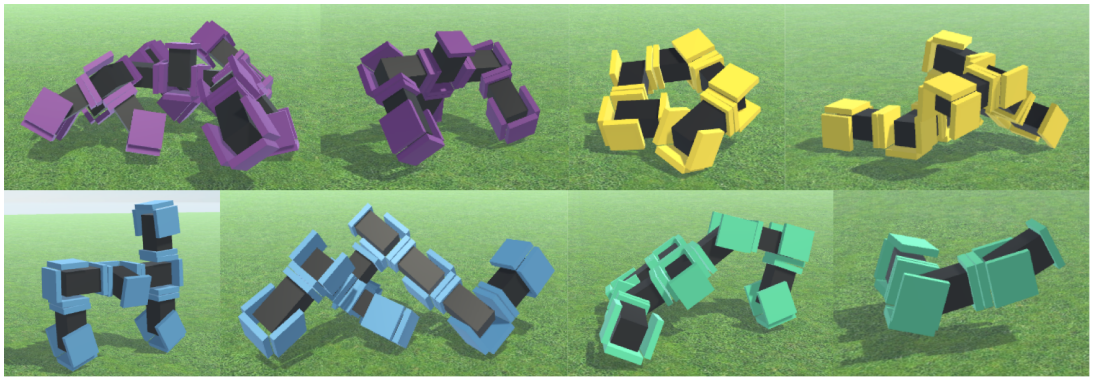


Figure 2: **Examples of well-performing morphologies.** Purple and top left: Copy controller. Yellow and top right: Sine controller. Blue and bottom left: Decentralized CTRNN controller. Teal and bottom right: Centralized CTRNN controller.

performance.

Modular robotics (MR) concerns robots built from separable modules or units that encapsulate some function of a robotic system (Stoy et al., 2010). This is as opposed to an integrated design with no clear sectional modularity. The modules contain actuation, computation, energy, and sensing as needed, as well as some mechanism to connect and transmit to other modules. They can easily be reconfigured by hand or by machine, making them highly suited for rapid prototyping of robot designs.

The field of modular robotics is quite young, with some of the first notable papers, like the CEBOT (Kawauchi et al., 1992) and Fracta (Murata et al., 1994) papers, being published in the 90s. Modular robots promised easily reconfigurable robots that could adapt their form to any use (Yim et al., 2000). Early systems such as the MTRAN showed self-reconfiguration into shapes for walking and climbing (Murata et al., 2002), and Zykov et al. (2005) showed the first minimal example of self-reproduction in modular robots. Throughout the 2000s and early 2010s, the focus was mostly on exploring the promise of reconfiguration and creating novel mechanical solutions for the modules.

Later Marbach and Ijspeert (2004), inspired by the works of Sims (1994) and Lipson and Pollack (2000), started to co-evolve configuration and control. With works like Marbach and Ijspeert's Adam and EDHMoR (Faíña et al., 2013), researchers started to experiment with the pipeline to create modular robots for any task. Evolutionary algorithms were uniquely suited for optimizing the robots, because the complexity of control scales exponentially with the number of modules (Marbach and Ijspeert, 2004). Additionally, by co-evolving we avoid the limitations and biases a human designer would bring, hopefully producing more novel and better adapted solutions (Faíña et al., 2013).

Many systems in MR use sine wave generator controllers when the focus of the research is elsewhere (Faíña et al., 2013; Liu et al., 2017; Veenstra et al., 2017), because they produce periodical movement with few parameters to optimize. This controller is a good baseline for the behavior of other controllers, as it is what we minimally expect from a decentralized controller.

Evolved centralized control is something that is not much used in MR, as the focus early on was on hand-crafted centralized reconfiguration systems (Murata et al., 2002). Later works have also shown centralized control that focus on task execution and locomotion (Brunete et al., 2012), however these are not evolved and do not necessarily scale well. It is still thought that some form of centralized control could be more suited to task execution (Seo et al., 2019), and so evolving centralized control should be explored.

Even so, decentralized control has also shown impressive results in task execution. Christensen (2006) showed an example of a decentralized neural network controller, which could self-reconfigure and self-repair. Their modules' swarm-like, imperfect behavior was able to control above 3000 modules in simulation, showing the scalability of good decentralized control. Another good example of neural network control is Jelisavcic et al.'s (2019) use of CPGs in the RoboGen modules. While this still results in distributed control, a CPPN encoding determines the CPG weights and can still enable module synchronization.

With the goal of investigating centralized and distributed controllers, we have implemented one centralized and two decentralized controllers, as well as a sine wave controller for a baseline. These controllers were implemented on a chain-type modular robot system, using the EMeRGE modules. Through measuring performance and morphological diversity, we evaluate which control approach is better suited for co-optimizing morphology and control in light of premature convergence of morphology. We believe that our approach to decentralized control and our findings of how centralized control affect morphology can help future researchers and developers better determine which control approach to implement in their own robotic systems.

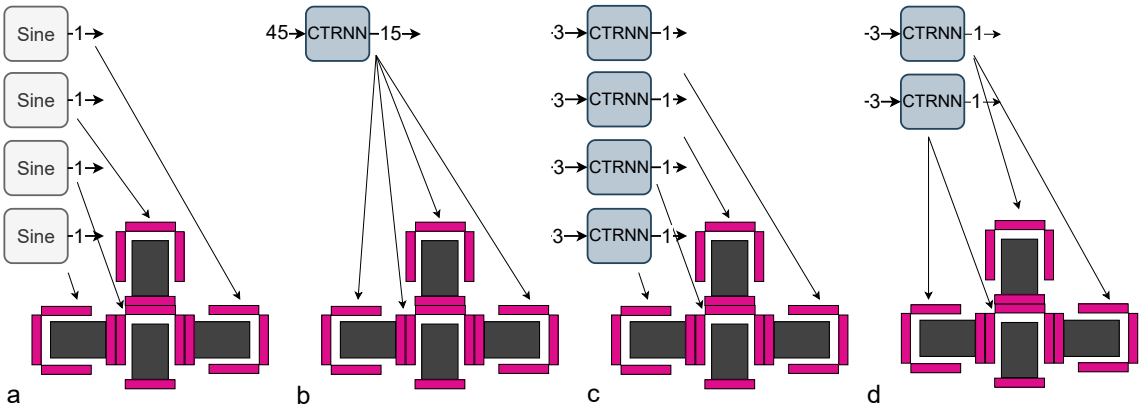


Figure 3: **The controllers and how they could map to the modules in a small modular robot.** The arrows with numbers represent inputs (left side of boxes) and outputs (right side of boxes). a) The sine controller, b) The centralized CTRNN controller, c) The decentralized CTRNN controller, d) The copy controller.

Methods

Our system consists of modular robots simulated in a flat ground environment and being measured on the task of locomotion. To co-optimize the morphology and controller, an evolutionary algorithm is used. This, as well as the four controllers we are investigating, will be presented below.

For this project, an environment and simulated modular robots were built in Unity with the framework of ML-Agents[1]. ML-Agents version 1.0.7 was used. Unity uses the Nvidia PhysX physics engine, which supports rigid body dynamics and updates physics steps every 0.02 seconds. The default physics engine settings were used.

The Modules

We are using the EMeRGE module (Moreno et al., 2017), see Figure 1. It is a simple module with one servo motor, so that each module works like a hinge. It has four connection faces, one male at the base, and three female on the top and sides. The male connection face can only connect to the female ones and vice versa, meaning the robot will have a root module with three possible child modules, growing outwards in a tree-like structure. Example morphologies can be seen in Figure 2.

The original design for the modules include infrared proximity sensors on each face. This was also implemented in our abstraction of the module, although the workings of the exact sensor model was not replicated. Instead, sensors here register all distances through a ray cast, meaning the distances it can register is not capped and could get very high. For not registering a distance, for example from being angled towards the sky, a sensor returns -1. In the event that a module occupies the connection site, the sensor will pick up the small distance towards the module. This was kept because the modules will move a little in relation to each other, which a controller might use to detect the presence and behavior of child modules.

The Controllers

There are four controllers implemented in this system (Figure 3), each described below. For all of them, the output produced is the desired angle of a module's servo.

Three of the controllers use a continuous time recurrent neural network (CTRNN), for which the neat-python library was used[2]. This network was chosen in order to have the possibility of dynamic temporal behavior (Beer, 1995). Here each node updates based on a differential equation, with neuron potentials as dependent variables.

neat-python's CTRNN mutation operators can adjust weights and biases, change activation and aggregation functions, and cut away/disable or add/enable connections and nodes. Here, close to default values were used for each gene mutation rate, and this was equal for all our CTRNN controllers. These rates were then scaled by the controller's control mutation rate.

Open-loop sine wave generator The open-loop sine wave generator is a decentralized controller that performs well because it produces periodic movement through sine waves, although it has no sensor input. In our case, the sine wave controller is used to provide a baseline to compare the other controllers to. The controller is given by the function

$$y(t) = A * \sin(w * t + p) + o \quad (1)$$

where A is the amplitude, w is the frequency, t is time, p is phase, and o is offset. y(t) is the controller output at time t that is directly fed to the servo's desired angle.

[1]The source code can be found at https://github.com/mia-katrin/Modbots

[2]https://neat-python.readthedocs.io/en/latest/

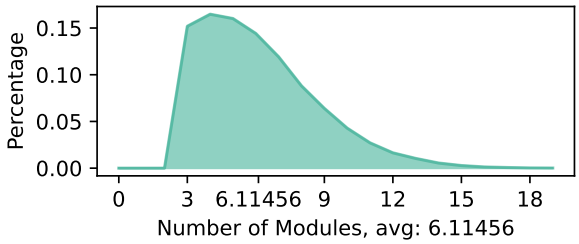


Figure 4: **The distribution of number of modules in individuals in a random population.**

In order to enable easier synchronization between the modules, the frequency was fixed in all sine wave controllers and was not allowed to mutate. We noticed that without this, the sine controller was susceptible to choose local optima solutions. The sine wave controller has 3 parameters for each joint, meaning an average robot of 6 modules will have 18 parameters to optimize for the controller. When a module is added to the morphology, it is instantiated with the control parameters of the parent module.

Centralized CTRNN A straightforward approach to using a CTRNN for a modular robot is to simply gather all sensor outputs and feed them into one big CTRNN, that then outputs all controller actions. This leads to a fixed size CTRNN controller.

In initial experiments with the centralized CTRNN controller, different numbers of inputs and outputs were tested. Because there was a clear tendency to have a small amount of modules, we chose 15 to be the number of modules that would be controlled. This allowed the network to be as small as possible while still accommodating larger creatures at initialisation, however it was very rare to see larger creatures with this controller. When a modular robot is smaller than 15 modules, the rest of the inputs to the network is set to 0.

The centralized CTRNN controller will have 45 inputs and 15 outputs. It is initialized with 45 hidden nodes, but is only 20% connected. The order of mapping modules to input and output follows a depth first ordering of the modules, so that the first three inputs and first output goes to the root, the second three inputs and second output goes to its child, and so on.

The centralized CTRNN controller ends up having circa 600 connections and 60 nodes, each with respectively 3 and 5 parameters to tune, for a total of 2100 parameters.

Decentralized CTRNN Foregoing the benefits of a centralized brain, a decentralized approach leads to less parameters and a less complex optimization. The decentralized approach consists of each module getting its own CTRNN controller. When a module is added, it is instantiated with the control parameters of the parent module.

Here, we were able to use a small compact network of 3 inputs and 1 output, and with 3 hidden nodes and enabling all connections from the start, we get 4 nodes and 16 connections, totalling 68 parameters. For an average sized robot with one of these controllers in every module, this leads to about 400 parameters.

Copy Decentralized CTRNN The copy decentralized CTRNN controller, or the copy controller for short, is our alternative to the decentralized approach. It functions by having each robot keep a list of two CTRNN networks, which maps to different modules. The networks are the same as in the decentralized CTRNN controller. At initialization, these networks are clones, but as the optimization progresses they will mutate separately. The modules will then mutate which network they use for control, theoretically allowing specialization. When a module is added to the morphology, it will use the same network as its parent module.

At the start of an evaluation, the networks are copied into their corresponding modules, hence the name. They then function independently of each other, and because of different sensor input such as detecting ground or the presence of child modules, they will likely behave differently. Nevertheless, it is reasonable to assume it will not achieve the level of specialization that the decentralized CTRNN controller can. While this might be a trade-off, we assume the copy controller will be quicker to achieve a good fitness, and possibly not be as dependent on number of modules.

This controller, like the centralized CTRNN controller, is the same no matter the size of the robot. This gives us two controllers with 68 parameters, for a total of 136 parameters. Additionally, each module can change which controller they use, giving us a further average of 6 parameters.

Evolutionary Algorithm

The evolutionary algorithm used had tournament selection, with a tournament size of 4, and generational replacement. It was implemented using the DEAP framework (Fortin et al., 2012). Generational replacement was chosen because it can sometimes dislodge a population from early convergence. This happens because the best genotype will rarely be kept when the population is mutated and no elites are kept. Other parameters of the algorithm were also chosen to keep diversity. Most notably, the tournament selection size was kept small to increase selection pressure on the elites, while still being large enough to avoid a noisy evolutionary progression.

Morphology and controller had separate mutation rates and mutation powers, where all gene values mutated with a Gaussian distribution based on the mutation power. For the rates, a parameter sweep was done for each controller on a grid of 8 values for both controller and morphology (Figure 5), a total of 64 pairs. Each pair was run for 50 generations with a population size of 50, which totalled 2500

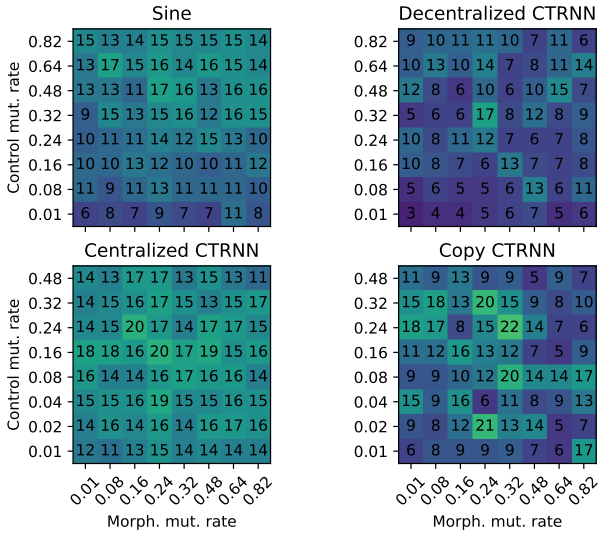


Figure 5: **The grids filled for the parameter tuning**, values are rounded average fitness for each mutation rate pair.

evaluations. This was done 4 times for each pair. Finally, the data for each morphology mutation rate and each controller mutation rate were collapsed and plotted against each other. The best performing option was then chosen.

The 8 sweep values were chosen for each controller and the body based on whether or not they divided the mutation rate internally. In the body and the sine and decentralized CTRNN controller, the mutation rate is divided by the number of modules in order to encourage the use of more modules. If it were not divided, a larger creature would mutate more than a smaller one. This would make it an unstable solution that quickly changed and disappeared from the search space. Since the average number of modules is 6 (Figure 4), the per-module mutation rate is 0.14 for the largest sweep value of 0.82. In the morphology mutation, this is further divided by circa 1/4th for each gene in the module.

Encoding of robots

For the encoding of the robots, a direct encoding is used. A directed tree is generated from a root node, after which it is sent to the simulator. The simulator builds the robot and prunes any branches that collide, prioritising keeping modules closer to the root. Modules that are not expressed are still kept in the genome.

The morphology can mutate by adding and removing modules, as well as changing the angle of modules. There is a slightly higher chance to get an add module mutation in order to bias creatures to grow. Additionally, one mutation will make a module duplicate one child branch to another connection site. This mutation was chosen to facilitate symmetry and larger jumps in the morphology landscape.

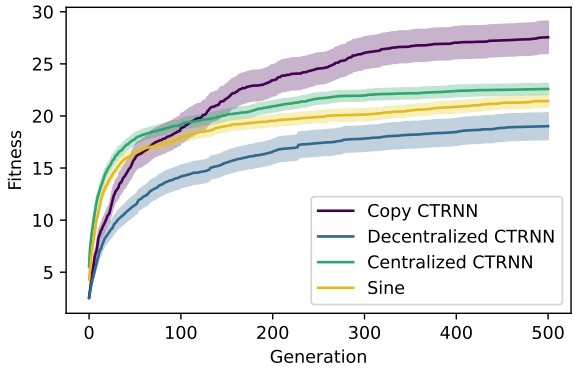


Figure 6: **The fitness progressions for all four controllers.** The solid lines are averages, and the shaded areas represent the standard error. Distributions can be seen in Figure 7.

Fitness function

The task that we measured the modular robots on was locomotion away from the origin during a set amount of time (100 steps of 0.2 seconds, roughly equalling 20 seconds in real time). Because this often leads to robots discovering the immediate optima of the somersault, or simply falling over, the robots were given 2 seconds to fall before the evaluation started. At the same time, the controller is not given input or give output. The fitness function is then

$$Fitness = \sqrt{(x_{end} - x_{start})^2 + (y_{end} - y_{start})^2} \qquad (2)$$

where x_{start} is the x-position after 2 seconds of simulation, and x_{end} is the x-position when the simulation ends. Likewise for y. The evaluation will stop early after 4 seconds from start if the robot has not moved in the last 2 seconds.

Results

Mutation rate sweep

As can be seen in Figure 5, the results from the sweep were often quite even. Only the copy and decentralized CTRNN controllers saw huge differences between different pairs, but all controllers had only minor differences after collapsing the data. Therefore, we settled on choosing a few of the ones that were contenders after the initial sweep, and run a few more evolutionary runs on those. A winner would then often be more clear. The final parameters for all controllers can be seen in Table 1.

Controller performance

The final runs were done on populations of 50 individuals for 500 generations, for a total of 25 000 evaluations. This was done for all controllers 64 times, and the resulting performances can be seen in Figure 6. Additionally, Figure 7 shows the distribution of performances for the different controllers.

Table 1: **The mutation rate parameters chosen after the sweep.** Note that for the sine wave and decentralized CTRNN controller, as well as the morphology, the working mutation rate on one module is divided by the number of modules in the robot.

Controller	Morph. rate	Controller rate
Sine wave	0.32	0.64
Centralized CTRNN	0.24	0.16
Decentralized CTRNN	0.24	0.48
Copy CTRNN	0.32	0.08

In order to get an overview of all significant differences, 6 two-sided Mann-Whitney U test were performed between all controllers at generations 50 and 500, a total of 12 tests. An alpha level of 0.05 was chosen. Because we were conducting multiple comparisons, Bonferroni correction was used. This gives us an adjusted alpha level of 0.05 / 12 = 0.00416.

At generation 50, the sine and centralized CTRNN controllers were significantly different from the decentralized CTRNN controller (both $p < 0.0001$). There was no significant difference between the sine and copy ($p > 0.2$), sine and centralized CTRNN ($p > 0.07$), copy and centralized CTRNN ($p > 0.06$), and the copy and decentralized CTRNN controllers ($p > 0.01$).

At generation 500, the copy controller was significantly different from the centralized CTRNN, the sine, and the decentralized CTRNN controllers (respective p-values $p < 0.003$, $p < 0.0004$, $p < 0.0002$). There was no significant difference between the sine controller and the centralized and decentralized CTRNN controllers (both $p > 0.1$), or between the centralized and the decentralized CTRNN controllers ($p > 0.04$).

Effect on morphology

In Figure 8, the progressions for number of modules in morphologies are plotted for all the controllers. Here, we can see that the sine and centralized CTRNN controllers both end up at a lower average number of modules than the copy and decentralized CTRNN controllers. To confirm if this was significant, as previously 6 two-sided Mann-Whitney U tests were performed with Bonferroni correction between the different count distributions. An alpha level of 0.05 was used, which means that with correction we consider p-values below 0.05 / 6 = 0.0083 as significant.

Here we found that the copy and decentralized CTRNN controllers had no significant difference between them ($p > 0.4$) and the sine and centralized CTRNN controllers likewise had no difference ($p > 0.2$). However, the copy and decentralized CTRNN controllers were both different from the sine and centralized CTRNN controllers (all $p < 0.0001$).

Qualitatively, we recognise this from looking at the robots. The sine and centralized controllers had small, effective strategies while the copy and decentralized CTRNN controller both tended towards larger morphologies. The sine and centralized CTRNN controllers would do large, powerful movements, with some modules in the morphology not moving at all. As opposed to this, the copy and decentralized CTRNN controllers favored small, rapid movements. Here, the copy moved most its modules, while the decentralized CTRNN controller sometimes had unmoving modules. Example behaviors can be seen in the accompanying video[3].

In Figure 9, we see that there seems to be a divide between larger creatures that get high fitness, and those that get low fitness. Presumably, this is because falling strategies tend to be larger in order to get further, but it is also interesting to see

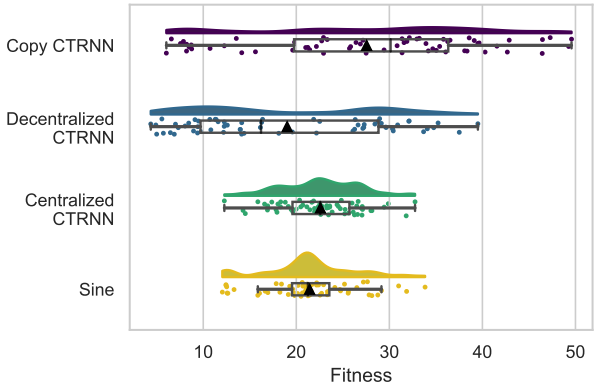


Figure 7: **The final fitnesses for all runs for each controller.** The distributions are showed with the underlying points scattered below. A boxplot is placed over the scattered values, with the mean marked with a triangle.

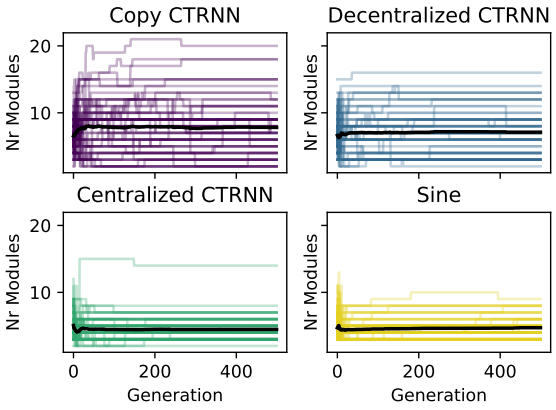


Figure 8: **The progression of number of modules for all runs.** Only changes in number of modules that led to a more fit individual is shown.

[3] https://www.mn.uio.no/ifi/english/research/groups/robin/research-projects/cocomo/media/kvalsund_master_thesis.mp4

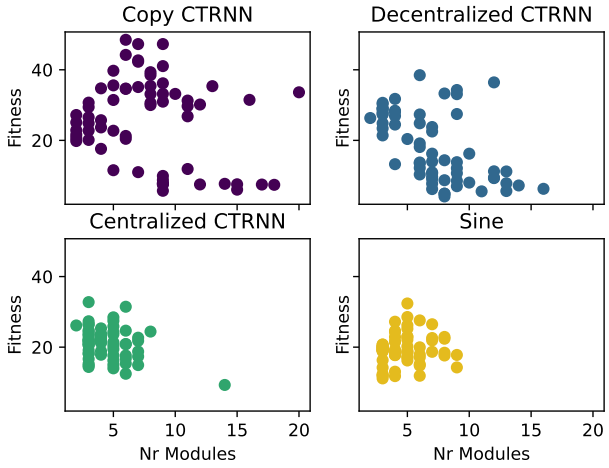


Figure 9: **The expressed modules of all robots plotted against their fitness.**

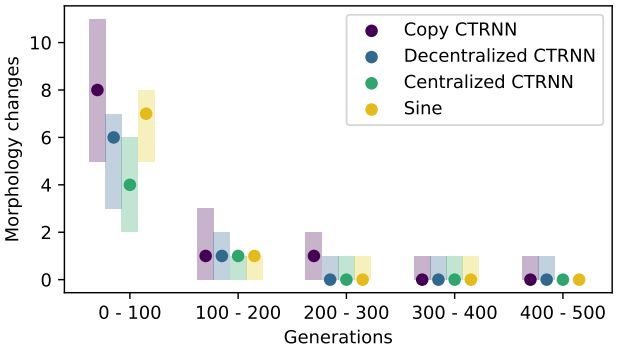


Figure 10: **Number of morphology changes that led to better fit individuals in each generation interval**. The shaded areas are between the 25 and 75 percentiles, and the dots are the medians.

that some larger solutions did quite well. Especially the copy controller did well with larger morphologies, managing to produce a fitness on par with the centralized CTRNN and sine controllers with upwards of 10 expressed modules.

When looking into the issue of early convergence of morphology only, Figure 10 was made. It shows shaded areas between the 25 and 75 percentiles of number of beneficial morphology changes in a run in each interval of 100 generations. It shows that the centralized CTRNN and sine controllers stop mutating morphology after 400 generations. The last two controllers kept mutating all the way up until the end. Interestingly, 72% of the copy controller runs experienced morphology changes in the 100-200 generation interval, and more than 50% had morphology changes in the 200-300 interval.

Discussion

Our results have shown that the copy controller performs significantly better than other controllers when co-optimizing the morphology and control of modular robots. Since it duplicates behaviors, modules are more likely to be synchronized. Moreover, when a new module is added, a working control unit can be inherited that is already potentially useful. Even though the sine and decentralized CTRNN controllers had a similar feature of inheriting the parent module's control, the copy controller is more likely to be useful since it is already evolved to work in many different parts of the robot. In addition, because a control mutation affects multiple modules, it has an overall larger effect on the behavior of the robot compared to a mutation in the other controllers. Because of this, the copy CTRNN approach is less able to fine-tune a single controller compared to the other approaches and therefore may rely on morphological change to see a performance increase. This feature would thereby promote continued morphological diversification compared to the other controllers, as seen in Figure 10.

From the fitness progressions, we can see that the sine and centralized CTRNN controllers converged rather fast compared to the other two. They also showed a pattern of quickly finding a final morphology of relatively small size, and then optimizing the controller. Meanwhile, the other two controllers spent time developing both and thus converged slower. Because having more modules means the robot has more potential force, allowing for more movement and higher fitness overall, the sine and centralized CTRNN controllers were then at a disadvantage. These results confirm that there is a trade-off between fine-tuning controllers and getting a good fitness with a small morphology while losing the potential of getting a higher fitness and a large morphology.

The distributions of solutions for the controllers vary wildly, as seen in Figure 7. While the sine and centralized CTRNN controllers had a more solidly high performance, the decentralized CTRNN controllers both had very flat distributions, stretching from the worst to the best performances recorded. The lower fitnesses can be accounted for as robots that grow tall and fall in one direction, as some of these have been visually confirmed to be. The higher values of the copy, decentralized and centralized CTRNN controllers often had rapid module movements that either led to small jumps or shuffling behaviors. Because of fixing the sine controller's frequency, this strategy was not available to the sine controller, and so its worse performance must at least be partially attributed to that. Even though it could have rivalled the others by growing larger, the CTRNN controllers' behavior was likely less complicated to evolve. Still, the sine and centralized CTRNN much more often arrived at very similar local optima, which tended to have the same fitness. Here, the centralized CTRNN controller had

an advantage over the sine controller, because it could optimize further by adding non-periodic movements.

Because we had the same morphology mutation rate for the sine and copy controllers, we could expect similar morphological diversity from these. However, it is clear from our results that this is not the case. When keeping in mind that some of the more scalable strategies available to the CTRNN controllers were not available to the sine controller, it could simply be that there were less available good morphologies for the sine controller. Even so, the lack of exploration of morphologies suggest that the sine controller could be improved. Perhaps using a decentralized copy approach like the copy controller would have enabled more morphological diversification during evolution.

The centralized CTRNN approach that was implemented could evolve a network topology that connected to up to 15 modules. This means that evolving larger morphologies would involve the CTRNN accommodating for more outputs. Since the CTRNN in this case would then have even more parameters to optimize, we would expect the centralized CTRNN to converge even quicker. This could potentially be overcome by connecting parts of the neural network of the centralized CTRNN to the morphology and copying these parts of the CTRNN when a new module is introduced. Another possible cause for the rapid convergence seen in the centralized CTRNN is that the initial experiments to determine the supported number of modules only ran for 100 generations. Although we have no indication that it would, the centralized CTRNN approach could generate larger morphologies when given different mutation rate values.

The copy and decentralized controllers both had issues of some number of unexpressed modules being added to the genome. These were unexpressed because they collided with other modules or the floor. 2 and 5 out of 64 samples from respectively the copy and decentralized CTRNN controllers had 10 to 20 unexpressed modules. Since they were unexpressed, the only effect they had on the individual was lowering the per-module mutation rate. This meant that the morphology in both, and the controllers in the decentralized CTRNN, would mutate less as the number of unexpressed modules grew. This bloating of the genome would in theory stabilize them from mutating. The two other controllers did not have issues with this.

Another issue is that of some pervasive local optima. Likely due to the angular shape of the EMeRGE modules, initial populations of robots found success with a single module dragging itself forward. In an attempt to avoid this, we constrained the robots to having a limp root module. This mitigated the problem somewhat, but similar strategies of using the corners of the female connection plates persisted all throughout the project. For example, the aforementioned jumping and shuffling behaviors are likely only possible because of the module shape.

We wanted our results to be as applicable as possible to other evolutionary algorithms, and so we kept ours simple. However, with the persistence of local optima solutions, we ran the risk of not being able to study the effects of the controllers as they all chose similar optima. Therefore, we tailored our algorithm to keep diversity, for example by having generational replacement. While using a diversity maintenance method could have minimized the occurrence of local optima solutions, it would have been difficult to parse which results where caused by the controllers and which were caused by the algorithm. Even so, in future work the same controller approaches could be tested with diversity enhancing methods to investigate what different control behaviors arise.

It would be useful to measure the different controllers on tasks or environments that require more sensing. Here, we would be better able to study different strategies that emerged, and whether the controllers were as equally equipped for task execution as they were for locomotion. It seems probable that the centralized CTRNN would perform better than the others here. The decentralized approaches would benefit from communication between modules, and coupled CPGs such as the ones used by Ijspeert et al. (2007) could likely work well here.

Lastly, in order to test whether these findings translate to other robotic systems, the same controller types should be implemented on a system with different modules and/or controllers. Since the EMeRGE module's female connection plates can be used for effective dragging and jumping, a less angular option like the RoboGrammar modules (Zhao et al., 2020) could force the controllers to choose more complicated movements. In addition, the previously discussed CPGs can be used to achieve periodic motion and would be a good option to further incorporate the sensors in locomotion.

Conclusion

In this article we implemented and tested four controllers that were co-optimized along with a modular robot morphology. With testing three decentralized and one centralized controller, we got insight into how these can be done well and different challenges that arises for each approach. Markedly, we learned that there is significant advantage to simplify your controller to facilitate for global synchronization, as was found when the copy controller out-competed the decentralized CTRNN controller. The copy approach allows for better new control of added modules, thus more morphological development, and larger jumps in the search space. Regarding centralized control, we highlighted the early convergence of morphology and performance when it comes to having a complex controller to optimize. Given that these findings translate to other controller networks and morphologies, they can aid future choices of control when co-optimizing morphology and control.

References

Auerbach, J. E. and Bongard, J. C. (2011). Evolving complete robots with cppn-neat: the utility of recurrent connections. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1475–1482.

Beer, R. D. (1995). On the dynamics of small continuous-time recurrent neural networks. *Adaptive Behavior*, 3(4):469–509.

Brunete, A., Hernando, M., Gambao, E., and Torres, J. E. (2012). A behaviour-based control architecture for heterogeneous modular, multi-configurable, chained micro-robots. *Robotics and Autonomous Systems*, 60(12):1607–1624.

Cheney, N., Bongard, J., SunSpiral, V., and Lipson, H. (2016). On the difficulty of co-optimizing morphology and control in evolved virtual creatures. In *Proceedings of the Artificial Life Conference 2016, ALIFE 2016*.

Cheney, N., Clune, J., and Lipson, H. (2014). Evolved electrophysiological soft robots. In *Artificial Life 14 - Proceedings of the 14th International Conference on the Synthesis and Simulation of Living Systems, ALIFE 2014*.

Cheney, N., MacCurdy, R., Clune, J., and Lipson, H. (2013). Unshackling evolution: Evolving soft robots with multiple materials and a powerful generative encoding. In *GECCO 2013 - Proceedings of the 2013 Genetic and Evolutionary Computation Conference*.

Christensen, D. J. (2006). Evolution of shape-changing and self-repairing control for the atron self-reconfigurable robot. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 2539–2545. IEEE.

Faíña, A., Bellas, F., López-Peña, F., and Duro, R. J. (2013). EDHMoR: Evolutionary designer of heterogeneous modular robots. *Engineering Applications of Artificial Intelligence*.

Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A., Parizeau, M., and Gagné, C. (2012). DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175.

Ijspeert, A. J., Crespi, A., Ryczko, D., and Cabelguen, J.-M. (2007). From swimming to walking with a salamander robot driven by a spinal cord model. *science*, 315(5817):1416–1420.

Jelisavcic, M., Glette, K., Haasdijk, E., and Eiben, A. E. (2019). Lamarckian Evolution of Simulated Modular Robots. *Frontiers in Robotics and AI*.

Joachimczak, M., Suzuki, R., and Arita, T. (2016). Artificial Metamorphosis: Evolutionary Design of Transforming, Soft-Bodied Robots. *Artificial Life*, 22(3).

Kawauchi, Y., Fukuda, T., and Inaba, M. (1992). "a strategy of self-organization for cellular robotic system (cebot)". In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 1558–1565. IEEE.

Lipson, H. and Pollack, J. B. (2000). Automatic design and manufacture of robotic lifeforms. *Nature*, 406(6799).

Liu, C., Liu, J., Moreno, R., Veenstra, F., and Faina, A. (2017). The impact of module morphologies on modular robots. In *2017 18th International Conference on Advanced Robotics, ICAR 2017*.

Marbach, D. and Ijspeert, A. J. (2004). Co-evolution of configuration and control for homogenous modular robots. In *Proceedings of the eighth conference on intelligent autonomous systems (IAS8)*, pages 712–719. IOS Press.

Moreno, R., Liu, C., Faina, A., Hernandez, H., and Gomez, J. (2017). The emerge modular robot, an open platform for quick testing of evolved robot morphologies. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 71–72.

Murata, S., Kurokawa, H., and Kokaji, S. (1994). Self-assembling machine. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 441–448. IEEE.

Murata, S., Yoshida, E., Kamimura, A., Kurokawa, H., Tomita, K., and Kokaji, S. (2002). M-TRAN: Self-reconfigurable modular robotic system. *IEEE/ASME Transactions on Mechatronics*, 7(4).

Seo, J., Paik, J., and Yim, M. (2019). Modular Reconfigurable Robotics. *Annual Review of Control, Robotics, and Autonomous Systems*, 2(1).

Sims, K. (1994). Evolving virtual creatures. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1994*, pages 15–22.

Stoy, K., Brandt, D., Christensen, D. J., and Brandt, D. (2010). *Self-reconfigurable robots: an introduction*. Mit Press Cambridge.

Veenstra, F., Faina, A., Risi, S., and Stoy, K. (2017). Evolution and morphogenesis of simulated modular robots: A comparison between a direct and generative encoding. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 10199 LNCS.

Yim, M., Duff, D. G., and Roufas, K. D. (2000). PolyBot: a modular reconfigurable robot. In *Proceedings - IEEE International Conference on Robotics and Automation*, volume 1.

Zhao, A., Xu, J., Konaković Luković, M., Hughes, J., Speilberg, A., Rus, D., and Matusik, W. (2020). Robogrammar: Graph grammar for terrain-optimized robot design. *ACM Transactions on Graphics (TOG)*, 39(6):1–16.

Zykov, V., Mytilinaios, E., Adams, B., and Lipson, H. (2005). Self-reproducing machines. *Nature*, 435(7039):163–164.

Exploring the Effects of Centralized and Decentralized Control on Morphology and Performance

Mia-Katrin Kvalsund[1], Kyrre Glette[1,2] and Frank Veenstra[1]

[1]Department of Informatics, University of Oslo, Norway
[2]RITMO, University of Oslo, Norway
mia.kvalsund@gmail.com

Abstract

In Evolutionary Robotics, co-optimizing of morphology and control can be used to develop robots automatically for a task. However, this complicates the question of whether to choose a centralized or decentralized controller. Here, we study the effects of one centralized and three decentralized neural network controllers on modular robot performance and morphologies. We found that a decentralized approach that duplicates networks across the robot body performs significantly better than the other approaches. It also had more morphological diversity. If these findings translate to other robot systems, they can help future researchers design controllers for their co-optimized robots.

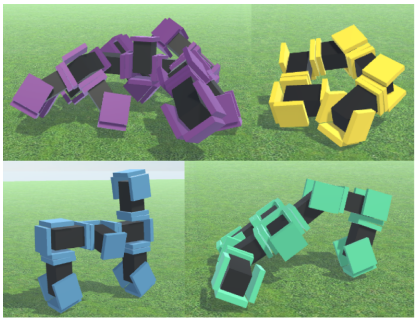


Figure 1: **Examples of well-performing morphologies.**

Introduction

When co-optimizing morphology and control in robots, researchers are often faced with the question of choosing centralized or decentralized control. In both approaches, there are challenges: On the one hand, centralized control may be better for task execution [1] and sensor inputs can affect any output. On the other hand, it is difficult to design a controller for a changing number of inputs and outputs, as the control might not be scalable [1]. A solution can be found in decentralized control since it grows with the morphology, however we might lose the benefit of synchronization.

Modular robotics (MR) concern robots built from a few separable modules that encapsulate some function of a robot, for example actuation or sensing [2]. Because the modules are loosely attached through magnets or latches, the robots are easily reconfigured by hand or machine, which opens avenues for quick prototyping and autonomous self-reconfiguration.

In theory, co-optimizing both morphology and control simultaneously can allow for solutions that utilizes the body as a cognitive resource through morphological computation [3]. In addition, co-optimizing morphology and control in modular robots can find the optimal configuration of modules for a given task. This limits the human bias, and more innovative and natural robots can be found [4].

However, co-optimizing morphology and control tends to lead to early convergence of morphology, as only the control continues optimizing [5; 6]. A possible explanation for this is that when the morphology mutates, it scrambles the function of the controller [5]. This effect has not been studied much in modular robotics, and it might be that the effect is less present in a decentralized modular robot: If a controller is optimized to work well in many modules, the addition of a module with such a control might be less likely to scramble the overall behavior. With the successful control systems of Sims [7] and Auerbach et al. [8], who both reuse parts of their control systems across the robot through indirect encodings, a control system with this behavior might be found through duplication.

Here, we present our work on co-optimizing centralized and decentralized neural network controllers along with modular robot morphologies. In particular, four controllers are implemented and tested on the task of locomotion. We find that a decentralized controller that duplicates control units into several parts of the robot shows better performance and more morphological diversification compared to the other controllers we investigated.

The Controllers

The following is an explanation of our controllers, their performance, and their effect on robot morphology. The modules used are the EMeRGE modules [9], with one depth sensor on each of the three connection faces.

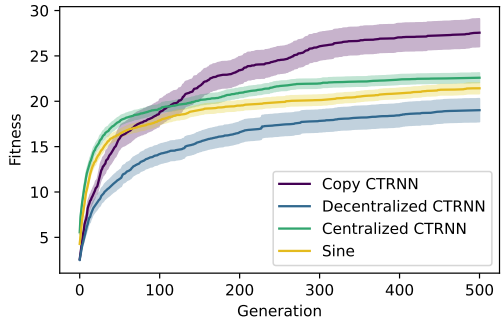


Figure 2: **The fitness progressions for all four controllers.** The solid lines are averages of the best individuals in each generation, and the shaded areas represent the standard error.

There are four controllers. Three of them utilize continuous time recurrent neural networks (CTRNNs), for which the neat-python implementation is used[1]. This network was chosen because it can be used to model biological neural networks [10] and because it allows for temporal dynamic behavior through recurrent connections [11]. Each node updates based on a differential equation, with neuron potentials as dependent variables.

The first controller is a decentralized sine controller, that is implemented by allowing one sine wave with evolvable parameters to directly control a module's actuator. This functions as a baseline of what the other controllers should be able to achieve. The second controller is the centralized CTRNN controller, that consists of one CTRNN that takes in all sensor inputs and outputs all actions. Its inputs are padded with zeros after the sensor inputs have been added. The third controller is the decentralized CTRNN controller where one small CTRNN is put in every module. The fourth controller is the copy CTRNN controller, which also has one small CTRNN in every module. However, this controller only keeps two small CTRNNs and distributes these across the body, copying them into different modules. Further details can be found in our paper on this experiment[2].

The robots are co-optimized with an evolutionary algorithm in a flat ground environment built in Unity. Performance is measured as traversed distance away from the start position. Each controller's control and morphology mutation rates were tuned through grid search. The final runs, of 50 individuals run for 500 generations, were done 64 times for each controller. The significant differences between the controllers were found with Mann-Whitney U tests using an alpha level of 0.05 that was Bonferroni corrected.

At generation 50, the only significant difference was between the sine and centralized CTRNN controllers and the decentralized CTRNN controller. The former two controllers had quickly gained fitnesses that were higher than the latter. At generation 500, the copy controller had a significantly higher average fitness than the other three, who now had no significant differences between them.

In addition, the resulting robot morphologies were investigated. When comparing robot sizes, measured as each robot's number of modules, it was found that the copy and decentralized CTRNN controllers had significantly larger sizes compared to the sine and centralized CTRNN controllers. It was also found that the copy controller appeared to experience more beneficial morphology mutations at later generations compared to the other controllers. The centralized CTRNN controller experienced the least beneficial changes, and settled quickly on small morphologies.

Our findings show that the copy controller performs significantly better than the other controllers when co-optimizing the morphology and control of modular robots. Notably, it has a much better performance than the decentralized CTRNN controller, which it is most similar to, confirming that adding simple duplication of control is more effective than a naive decentralized approach. At the same time, the copy controller also outperforms the centralized control, which suggests that decentralized control can achieve benefits that outweigh the benefits of simple centralized control if done well. Because each control unit in the copy controller is evolved to work well in many differently positioned modules, adding modules through mutation is often successful and results in larger creatures. This is beneficial because larger creatures have more actuation and force, and thus the potential to move further.

The copy controller is less able to fine-tune control compared to the other controllers because each change to a control unit affects multiple parts of the robot. This leads to the copy controller taking large jumps across the search space, and relying on morphological mutation to continue optimizing. Compared to this, the centralized CTRNN controller is able to fine tune, at the expense of early convergence of morphology. These findings confirm that there seems to be a trade-off between being able to fine tune with a small morphology and losing the potential to get a larger fitness with a larger morphology.

Conclusion

When co-optimizing morphology and control, the choice of centralized or decentralized controller becomes an issue because of changing inputs and outputs. Through co-optimizing four controllers, we have found that the one that duplicates behavior across the robot morphology performs the best and displays the most morphological diversity. If this translates to other robotic systems, it suggests that controllers that duplicate behavior can lessen early convergence of morphology and increase fitness when co-optimizing. It is our hope that these findings can help researchers choose control systems when co-optimizing robotic systems.

[1] https://neat-python.readthedocs.io/en/latest/

[2] Omitted until publication

References

[1] J. Seo, J. Paik, and M. Yim, "Modular Reconfigurable Robotics," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 2, no. 1, 2019.

[2] K. Stoy, D. Brandt, D. J. Christensen, and D. Brandt, *Self-reconfigurable robots: an introduction*. Mit Press Cambridge, 2010.

[3] C. Paul, "Morphological computation. A basis for the analysis of morphology and control requirements," *Robotics and Autonomous Systems*, vol. 54, no. 8, 2006.

[4] A. Faíña, F. Bellas, F. López-Peña, and R. J. Duro, "EDHMoR: Evolutionary designer of heterogeneous modular robots," *Engineering Applications of Artificial Intelligence*, 2013.

[5] N. Cheney, J. Bongard, V. SunSpiral, and H. Lipson, "On the difficulty of co-optimizing morphology and control in evolved virtual creatures," in *Proceedings of the Artificial Life Conference 2016, ALIFE 2016*, 2016.

[6] M. Joachimczak, R. Suzuki, and T. Arita, "Artificial Metamorphosis: Evolutionary Design of Transforming, Soft-Bodied Robots," *Artificial Life*, vol. 22, no. 3, 2016.

[7] K. Sims, "Evolving virtual creatures," in *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1994*, pp. 15–22, 1994.

[8] J. E. Auerbach and J. C. Bongard, "Evolving complete robots with cppn-neat: the utility of recurrent connections," in *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pp. 1475–1482, 2011.

[9] R. Moreno, C. Liu, A. Faina, H. Hernandez, and J. Gomez, "The emerge modular robot, an open platform for quick testing of evolved robot morphologies," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 71–72, 2017.

[10] R. D. Beer, "The dynamics of adaptive behavior: A research program," *Robotics and Autonomous Systems*, vol. 20, no. 2-4, pp. 257–289, 1997.

[11] R. D. Beer, "On the dynamics of small continuous-time recurrent neural networks," *Adaptive Behavior*, vol. 3, no. 4, pp. 469–509, 1995.