ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Bio-Inspired Robotics Lab
Prof. Dr. Fumiya Iida

Semester Paper

Evolutionary Algorithm for Robotic Body Extension

Autumn Term 2014

Supervised by:
Luzius Brodbeck

Author:
Maria Veiga

Contents

Abstract **iii**

1 Introduction **1**

2 Encoding **3**
2.1 Stochastic based generation (SBG) 3
2.2 L-Systems 4
2.2.1 Motivation 4
2.2.2 Implementation 4

3 Simulation **7**
3.1 Genetic Algorithm 7
3.1.1 Enforcing diversity 7
3.1.2 Individuals 8
3.1.3 Genetic Operators 8
3.1.4 Fitness Function 9
3.2 Other changes 9

4 Results & Discussion **11**
4.1 Enforcing diversity 11
4.2 Numerical case studies 12
4.2.1 Using rotations 12
4.2.2 Reaching targets 14
4.3 Comparison between SBG and L-systems 16

5 Conclusion **17**
5.1 Further developments 17

A Algorithm manual **19**
A.1 Simulation workflow 19
A.2 Description of the functions 19

B Fitness Function **21**

Bibliography **22**

Abstract

In this project a genetic algorithm is used for evolutionary design of cube structures with the main objective to reach an arbitrary point in space. The main focus of this project is to bridge the gap between the simulation and the physical construction of these structures. For this purpose, different ways to represent a building plan for these are explored, namely, a stochastic based generation (SBG) approach and Lindenmayer systems (L-systems) are considered. Both methods are shown to converge for targets with a moderate Cartesian distance, however, it is shown that the L-systems outperform the SBG approach for more distant targets.

Chapter 1

Introduction

Biological systems with the ability to grow have a greater adaptive power, this is one of the reasons why they are able to adapt to different environments and perform more complex tasks. Thus, a robot that can grow different parts is very appealing due to its autonomy and adaptive power in uncertain environments or possibility to perform new tasks [1].
The platform available for this project is a 6 DOF robot arm with the ability to use hot melt adhesives (HMA) to glue simple elements together [2]. The objective is to form arbitrarily complex structures that, when connected to the main body, add or increase the functionality of the robot. The robotic arm building a simple 2-D structure can be seen in figure 1.1.

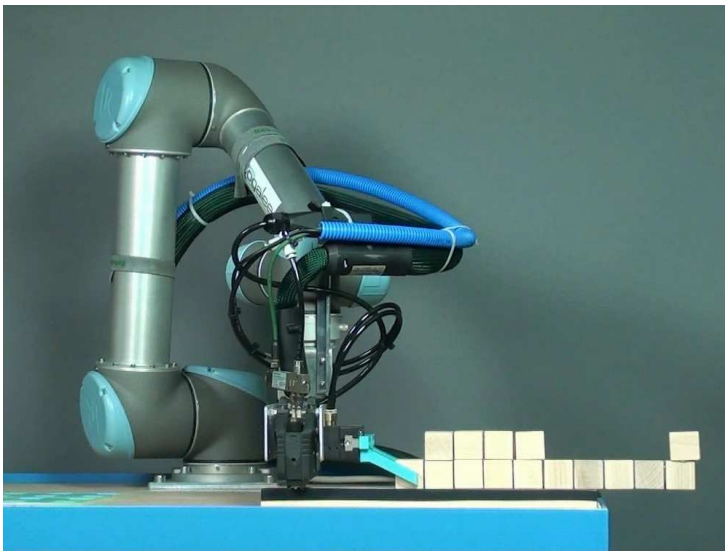
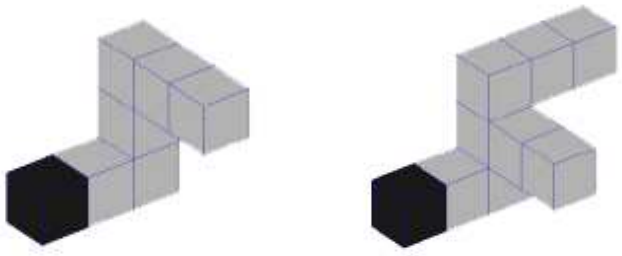


Figure 1.1: Robotic arm building a structure.

This project continues work that has been done at the Bio Inspired Robotics Lab (BIRL). Namely, in [3], a genetic algorithm is used to optimize a structure that reaches a point in space that the robot itself wouldn't be able to reach. This is done by placing an initial cube at the origin and adding cubes arbitrarily to one of the faces, repeating this recursively by randomly choosing an available cube. Then, through mutations and crossovers, new branches are added to the structure. An example of the structure that can be generated is shown in figure 1.2, detailing an initial structure and an added branch through mutation.



(a) Initial structure (b) With added branch

Figure 1.2: Structure generated in [3], featuring an initial structure and a added branch through mutation

In [4] major improvements to the algorithm are done such as reinforcement, stress and displacement analysis, and obstacle avoidance.
The main goal of this project is to bridge the gap between the simulation and the physical construction of a structure. In particular, the approach taken in this project is to find and optimize building plans (that encode the structures implicitly) such that the robot arm can interpret them and build the physical structures accordingly.
The main content of this report is spread over chapters 2 - 4. Chapter 2 describes the two methodologies used to generate the building plans of structures. In chapter 3, the optimization algorithm is introduced (Genetic Algorithm) and the changes to the evolution simulation are presented. In chapter 4, the results obtained are analyzed and the two methodologies used are compared. Finally, in chapter 5 closing remarks are given in addition to further developmental ideas.

Chapter 2

Encoding

One of the challenges of the transition between the simulation and the actual building of the structure is that there is not a unique way to construct an evolved structure. In previous works, the subject of the optimization was the structure itself. In this project, the optimization subject is changed to the building plan that encodes the structure implicitly, thus optimizing a set of instructions that can be handed to the robot. To achieve this, two distinct approaches to generate these building plans are used:

- Stochastic based generation (SBG)
- L-System

In this chapter, we will describe the details of both methods.

2.1 Stochastic based generation (SBG)

Using this method, a building plan (thereby denoted as genome) is generated randomly using the 5 different instructions detailed on table 2.1. The size of the building plan is pre-defined according to the task.

Table 2.1: Construction instructions.

Instruction	Meaning	Probability
1	Add a cube	0.2
2	Shift structure in positive y-direction	0.2
3	Shift structure in negative y-direction	0.2
4	Shift structure in positive x-direction	0.2
5	Shift structure in negative x-direction	0.2

In figure 2.1 it is shown the impact of each instruction applied to a structure generated by the SBG method.
The inclusion of an instruction that rotated the structure was also considered in order to produce bridge like structures, as shown in figure 2.2 where two structures are compared: one using a rotation and one without rotation. However, due to the added complexity to the structure in a 3D setting, this approach was not explored further.

2.2 L-Systems

2.2.1 Motivation

Lindermeyer systems (L-Systems) are a formal grammar system introduced by the botanist Lyndermeyer in the 18th century to define complex structures recursively [5]. The central idea to L-systems is the idea of rewriting, where one can define complex objects by successively replacing parts of a simple object using a set of rewriting rules. The rewriting is carried out recursively.
The simplest type of L-Systems are the D0L-systems (deterministic and context free). It is formally defined by the following triplet: $(\Sigma, \mathbf{P}, \alpha)$, where $\Sigma = \{s_1, s_2, s_3, ...\}$ is an alphabet, $\mathbf{P}$ is production map defined as $\mathbf{P} : \Sigma \to \Sigma'$, that maps each element of Σ to Σ' consisting of production rules, α is the starting point of the iteration, called the axiom. A simple example of a D0L-system can be seen in figure 2.3.
The use of this formality in robotics is not completely new, namely, this approach has been explored in [6]. For this project, this approach could lead to good results for two reasons:

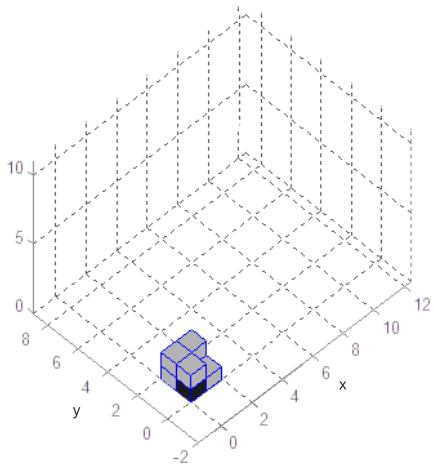
- The impact of crossover and mutation rules are greater
- The storage efficiency of these rules, technically, we only have to store the set of rules, the number of iterations and the starting seed

2.2.2 Implementation

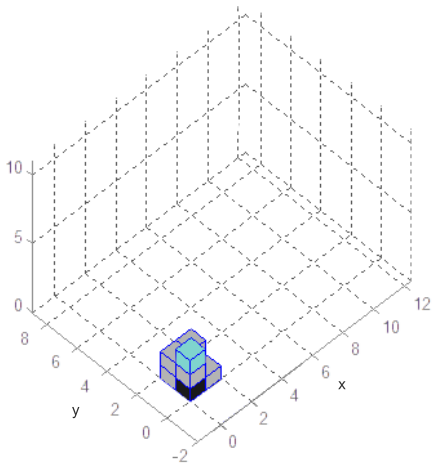
For our problem, our L-system triple is defined as:

- $\Sigma = \{$'add a cube', 'shift in +x', 'shift in -x', 'shift in +y', 'shift in -y'$\}$
- $P : \Sigma \to \Sigma'$, where Σ' is generated randomly
- α is defined as adding a cube

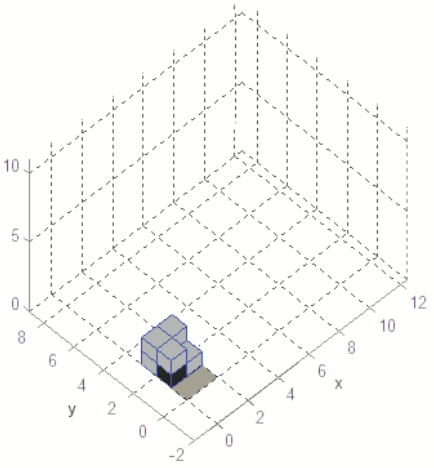
Further, the number of iterations that are performed are fixed.



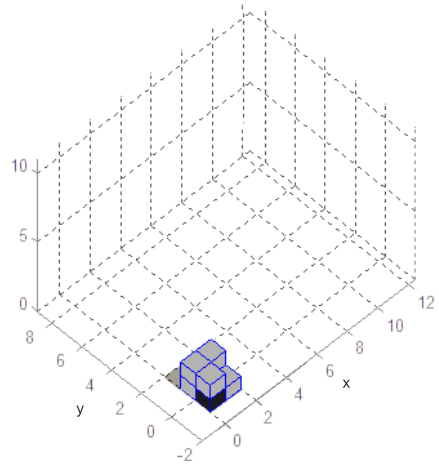
(a) Initial structure



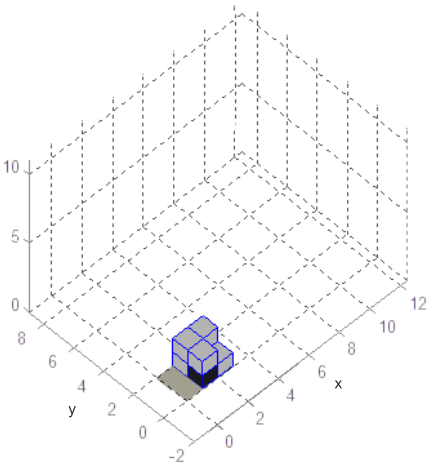
(b) Executing instruction 1 - adding a cube.



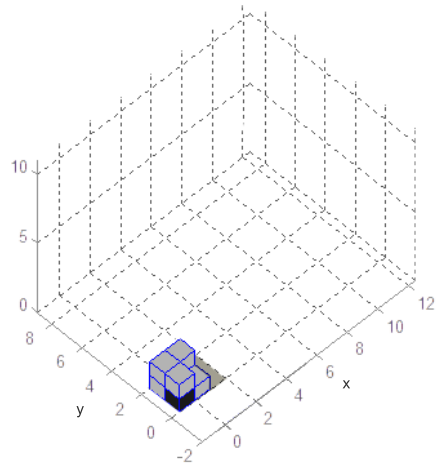
(c) Executing instruction 2 - shift in positive y-direction.



(d) Executing instruction 3 - shift in negative y-direction.

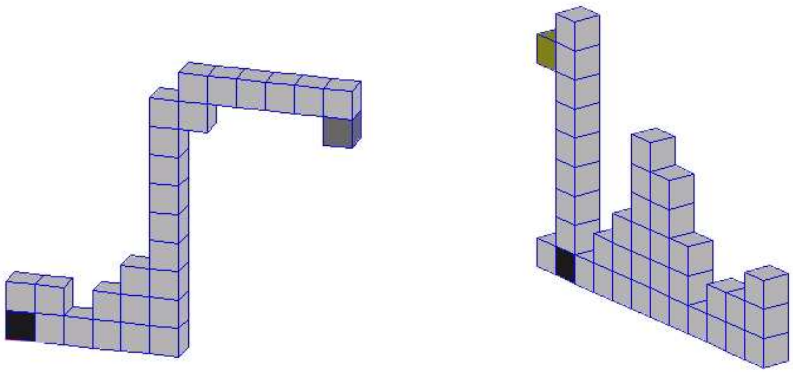


(e) Executing instruction 4 - shift in positive x-direction.



(f) Executing instruction 5 - shift in negative x-direction.

Figure 2.1: Impact of each instruction on the structure generated by the genome: (1 3 1 1 5 1 1 2 1 4 1).



(a) (b)

Figure 2.2: Structure generated with possibility of rotation and structure without possibility of rotation

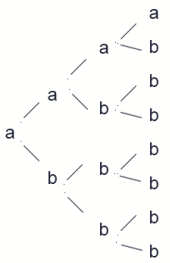


Figure 2.3: Example of the evolution of a D0L-system with $\Sigma : \{a, b\}$, $\mathbf{P}(a) \rightarrow ab$ and $\mathbf{P}(b) \rightarrow bb$ and starting axiom $\alpha = a$ after 3 iterations

Chapter 3

Simulation

In this chapter, the optimization algorithm is introduced and explained. Furthermore, the changes to the previous simulation, presented in [4], are denoted.

3.1 Genetic Algorithm

The Genetic Algorithm (GA) is based on the idea of evolution present in nature. It is a search heuristic used find and optimize solutions. A solution is called an individual and a set of solutions a population. Each individual is evaluated with respect to a fitness function, that aims to give an indication to how good the solution is. The individuals who are most fit are selected to produce offsprings for the next generation (iteration of the algorithm).
The genetic algorithm consists of the following main steps:

1. Initialize population randomly
2. Measure fitness of all individuals
3. Generate new population (applying elite selection, crossover and mutation)
4. Iterate until stopping criteria is reached

3.1.1 Enforcing diversity

The efficiency of the GA can be very dependent on its initial population [7], thus a parameter of similarity was introduced, that forced the initial population to be distinct.
Let **A**: {set of cube coordinates in structure A}, **B**: {set of cube coordinates in structure B} and $|\cdot|$ denote cardinality, then the similarity parameter is defined as:

$$\text{similarity } \% = \frac{|A \cap B|}{\min(|A|, |B|)} \times 100 \tag{3.1}$$

For example, in the limit case where the similarity parameter is set to 100%, only the organisms that are exactly the same are excluded. On the other hand, when the similarity parameter is set to 20%, all the organisms that score 20% (or more) of similarity are excluded. This impact of this parameter is that when it is set to be low, only organisms that are very different are allowed in the initial population.

3.1.2 Individuals

Stochastic based generation (SBG)

The individuals were the building plans - sequences of integers where each integer corresponds to an instruction. The number of cubes per organism was predetermined.

L-system based encoding

The individuals were the L-system rules, which determine how an instruction transforms after one iteration.

3.1.3 Genetic Operators

For both of the simulations, we induced mutation and crossover only.

Stochastic based encoding

The mutation was done in the following way:
Given a genome, each instruction has a predefined probability[1] to mutate. The genome is iterated through and the positions to mutate are determined. There are 3 mutations that occur with equal probability:

- Change the instruction - e.g: $1 \rightarrow 3$
- Duplicate the instruction - e.g: $1 \rightarrow 11$
- Delete the instruction - e.g: $1 \rightarrow []$

The crossover operation was done in the following way: Two individuals are picked randomly. Then, a random point is chosen in each genome to determine where the individual's genome gets cut. The remaining parts get merged together to form a new individual with a mixed genome, possibly with a different length. An example of this operation can be seen below:
Genome A: 111**2**1411
↑
Genome B: 1312**1**151
↑
This example yields a new organism with genome: 111**21**151

L-system based encoding

The mutation was done in the following way:
Given a set of rules, a subset of these rules are randomly chosen for mutation. The possible mutations, having chosen the rule, are:

- Change the instruction - e.g: $1 \rightarrow 3$
- Duplicate the instruction - e.g: $1 \rightarrow 11$
- Delete the instruction - e.g: $1 \rightarrow []$

Furthermore, there is the option to bias the mutation towards growth, by increasing the probability of the instruction that adds a cube.
The crossover operation was done in the following way: Two organisms (sets of rules) are picked randomly. Then, a random subset of rules of organism A is chosen. The rules chosen from organism A are substituted by the rules in organism B. The

[1] for our simulations we set $p = 0.2$

new set of rules, obtained by mixing the rules from organism A and B form a new organism. For example, consider the following genome with the rules to be swapped in bold face:
Genome A: rule 1.1, **rule 1.2**, **rule 1.3**
Genome B: rule 2.1, **rule 2.2**, **rule 2.3**
This example yields a new organism with genome: rule 1.1, **rule 2.2**, **rule 2.3**

3.1.4 Fitness Function

The fitness function from the previous work of [4], presented in equation 3.2, was slightly adjusted to allow for bigger structures, namely, the negative weight given to amount of cubes was decreased and incorporated stricter stress constraints. The inclusion of 'number of operations', which includes the information on how much work the robot should do, is also possible but our results were not measured with this parameter.

$$\begin{aligned} \text{fitness} &= W_{Distance} \times (1 - Distance) - W_{NumCubes} \times NumCubes - ... \\ &... \quad -W_{MaxStress} \times MaxStress - W_{MaxDisplacement} \times MaxDisplacement + ... \\ &... \quad +W_{NumConnections} * NumOfConnections \end{aligned} \tag{3.2}$$

The closest to 100, the fitter the structure is. The full details of the equation can be found in appendix B.

3.2 Other changes

- **Connected structures** Solutions that represent disconnected structures do not make sense in our problem, so the connectiveness of the structures was enforced in the following way: if a shift instruction is executed and the next instruction leads to a disconnected structure, a cube is placed to avoid that (and the building plan adjusted accordingly);
- **Tumble check** A stability check was added to determine whether the structure would tumble or not by calculating the center of mass of the structure;
- **Surface of support** A surface of support was added to approximate to what would happen in a real setting, where only a part of the structure has ground support;
- **Memory reduction** For larger structures, the memory allocation became a problem in particular due to the Truss matrix generation in the stability check part of the simulation. So the simulation now only stores the fittest individual from each generation and matrices that consist of integers are stored as singles;
- The code was re-factored to introduce a modular structure.

Chapter 4

Results & Discussion

In this section, results from various case studies will be reported and discussed. The task is to reach an arbitrary point in a Cartesian space, described by (x, y, z) coordinates. For example, as shown in figure 4.1, the target is set to be $(10, 10, 10)$.

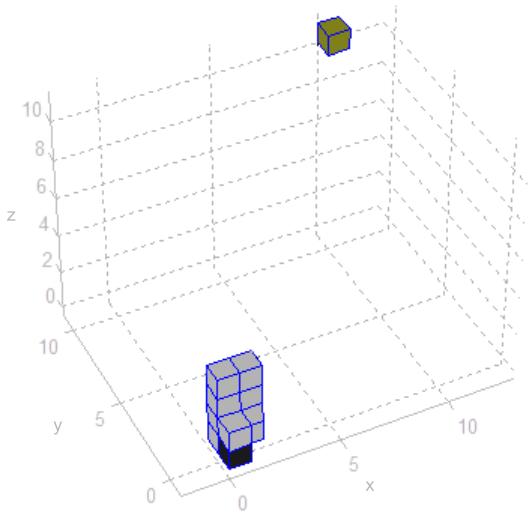


Figure 4.1: Structure starting from (0,0,0) with the target, represented by the golden block, set to be (10,10,10)

4.1 Enforcing diversity

The similarity parameter is calculated as described in Chapter 3, where the organism is rejected if it is $\geq$ x % similar to the previous organism. In the limit case of 100% similarity, we only discard organisms that are exactly the same. A simulation run is performed to gain insight on the impact of this similarity measure. The numerical results can be found on table 4.1.
The hypothesis is to have a better coverage of the solution space if differences are enforced, because this way a very similar initial population is avoided. A plot for each $\leq$ % similarity is shown in figure 4.2 and these show:

- The variation across the simulations seems to be smaller when differences are

enforced. This means that the simulation is less impacted by a poor choice of initial population.

- There is a quicker convergence, on average, using some difference enforcement. For example, if we observe the generation where more than 50% of the simulation runs are above fitness level 70, we obtain for similarity level of 40%, generation 25, whereas for no difference enforcement, generation 34.

Table 4.1: Varying the similarity parameter, with the following parameters for the simulation: Maximum generation: 60, target to reach: $(10, 10, 15)$, population size: 20, averaged over 10 runs.

Similarity	Fitness Average	Fitness Std deviation
20%	85.91	6.32
40%	84.19	5.57
60%	82.71	6.96
80%	84.25	8.45
100%	85.29	9.08

4.2 Numerical case studies

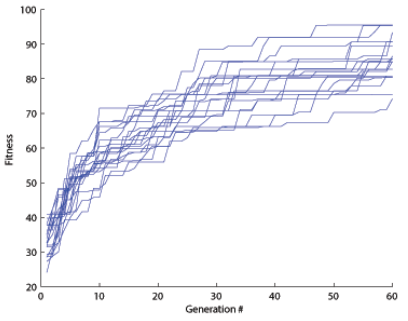
In this section, results from various case studies will be reported and discussed. The task is to reach an arbitrary point in space, using the encoding methods described previously.

4.2.1 Using rotations

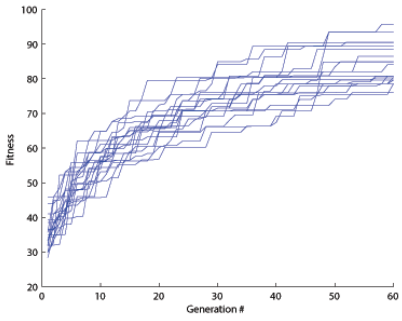
Although this encoding method was not carried through to completion during the project, the results and reasons for its exclusion will be reported and discussed for the sake of completeness.
Parameters:

- 2D setting
- 30 generations
- 10 organisms per generation
- 5x5 surface of support
- target: (0, 10, 10)

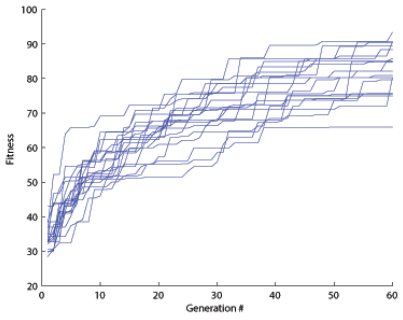
Although this method would provide a more complete coverage of the solution space (by allowing bridge like structures), the physical construction of these objects would be more complex. The center of rotation has to be well determined and the stability of the structure during the whole process has to be guaranteed. This also adds significant computation cost because although the final structure might be feasible, the construction process is not necessarily guaranteed to be feasible at each step, as shown in figure 4.3. An intermediate check was introduced for each operation executed and this made the simulation significantly slower, moreover, the generalization to 3D would not be straightforward thus investigation on this approach was halted.



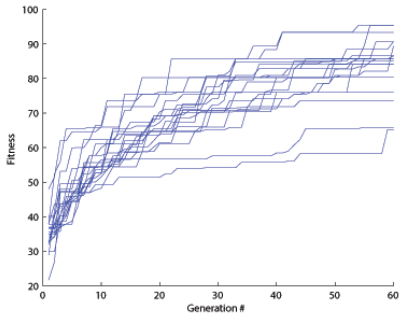
(a) Similarity 20%



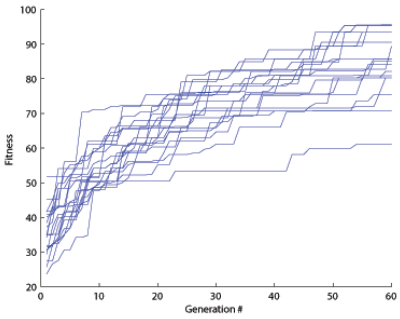
(b) Similarity 40%



(c) Similarity 60%

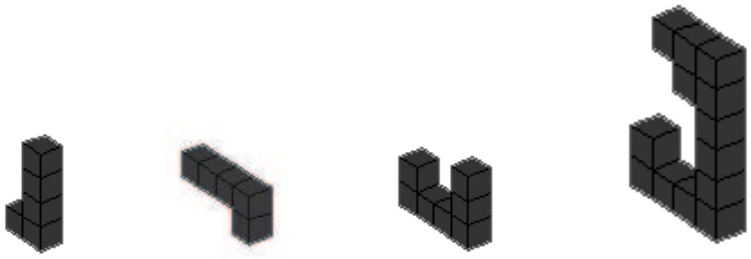


(d) Similarity 80%



(e) Similarity 100%

Figure 4.2: In each of the plots we fixed the similarity parameter, each plotted line represents a simulation run and the relation between the generation number (x-axis) and the fitness value of the fittest organism (y-axis)



(a) Step 6 (b) Step 7 (c) Step 12 (d) Final structure

Figure 4.3: Stepwise construction of the shape shown in d). Given that the lowest part of the figure is the only part touching the ground, it is seen that the structure generated in 7 is not stable, in particular, its center of mass falls outside of the base.

4.2.2 Reaching targets

In this subsection, the task to reach different targets is investigated. In particular, in the first case presented, a structure is constructed to reach a target that is located in the following coordinates (20, 20, 20), defined on a Cartesian plane. In the second case, the task is to reach a target located in the coordinates (25, 40, 40).

Reaching target (20, 20, 20)

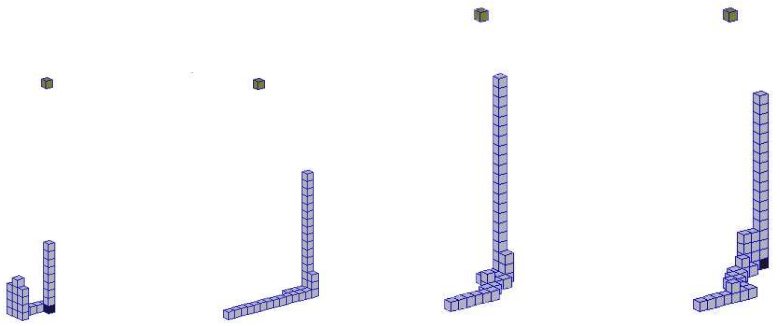
The parameters for this simulation run are defined below:

- 100 generations
- 20 organisms per generation
- 15x15 surface of support
- Averaged over 10 runs
- Similarity fixed at 40%

The results of this simulation can be seen in table 4.2 and the structures generated by the SBG method and L-systems method can be visually observed in figure 4.4 and 4.5 respectively.

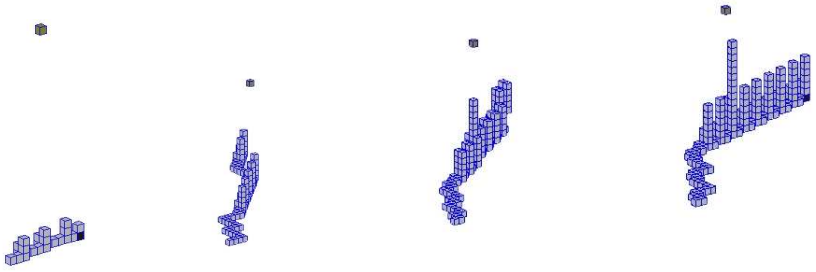
Table 4.2: Average fitness of the fittest organism in the terminal generation reaching the target (20, 20, 20)

Population number	SBG	L-system
10	54.24	80.62



(a) Generation 1 (b) Generation 20 (c) Generation 60 (d) Generation 100

Figure 4.4: Fittest organism per generation using SBG, reaching target $(20, 20, 20)$.



(a) Generation 1 (b) Generation 20 (c) Generation 60 (d) Generation 100

Figure 4.5: Fittest organism per generation using L-Systems, reaching target $(20, 20, 20)$.

Reaching target (25, 40, 40)

- 100 generations
- 20 organisms per generation
- 30x30 surface of support
- Averaged over 10 runs
- Similarity fixed at 40%

Reaching target (25, 40, 40) was significantly harder due to the distance from surface of support. Both methods showed poor convergence. Due to this poor convergence, the L-system method was slighly changed to introduce a growth bias in the mutation, this means that for each mutation there is a fixed probability that the instruction 'add a cube' is added to the rule [1]. This slight bias lead to better results, as shown in table 4.3.

[1] growth bias probability set to 0.4

Table 4.3: Comparison between different methods, including a growth bias in the L-systems encoding

Parameters	SBG	L-Systems	Biased L-Systems
Average fitness	42.45 ± 4.58	55.21 ± 14.02	81.85 ± 7.33
Time taken (s)	635.58	1409.87	9713.50
Memory allocation (GB)	0.6	0.6	0.8

4.3 Comparison between SBG and L-systems

Referring to the results displayed in the section above and in table 4.3, a few main observations can be drawn:

- For targets that are close, both methods seem to converge and perform reasonably well
- For targets that are further away, the L-systems approach outperformed the SBG
- The introduction of a growth bias influenced the convergence positively, which comes from the idea to start from small structures and allow them to grow.

Chapter 5

Conclusion

This project details the theory and process to achieve the generation of building plans that can be passed to the robot to build structures that add functionality and adaptability to it. In particular, in this project, instruction sets that encode the building method of structures were generated successfully using a genetic algorithm, using two encoding methods: SBG and L-systems).
The first approach used was the SBG method. The convergence of the algorithm using rotations was good in 2-D but the generalization to 3-D was not efficient nor feasible, so this approach was abandoned, reducing the number of instructions to five, as described on table 2.1.
The second approach used was the encoding using L-systems with the same set of instructions. Using this method, the building plan was not generated sequentially but based on a Production map and a fixed number of iterations. A growth bias option was also added.
One of the main challenges for the convergence of solutions, in particular when using the L-systems encoding was that the structures were often disconnected, a fix was introduced to enforce connectiveness of these.
Both methods were used in the context of a genetic algorithm to determine the optimal solution. A parameter of similarity was also introduced in order to prevent initial generations being too similar. The observed impact of this parameter was a smaller variance across multiple runs and often, less generations were needed to reach a particular fitness value.
From the numerical case studies, the L-systems encoding seems to be more robust and perform better in comparison to the initial SBG method. Although positive results were obtained, these were only done in simulation. The feasibility of these building plans should still be studied.

5.1 Further developments

One clear problem of both of these approaches is the fact that, most likely, the optimal solution can not be obtained using these encodings because of the simplicity of the instructions. One drawback, pointed out earlier in this report, was the inability to build bridge like structures. This can have a significant influence on where the center of mass of a structure is and, consequently, in its stability. One attempt to overcome this was the use of rotations but it was found to not be very efficient. Another way could be using auxiliary structures when building the structure as detailed on figure 5.1.
Another issue that was detected was that a genome often contains redundant operations, leading to a building plan that is not efficient, so these sequences of redundant

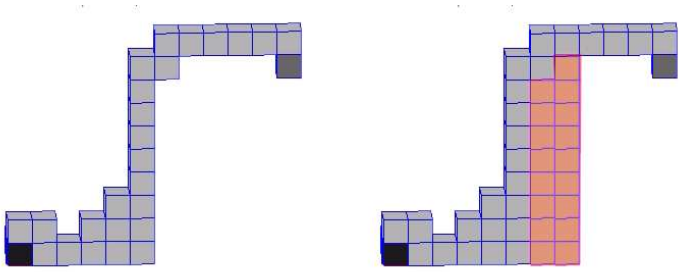


Figure 5.1: Building the structure on the left hand side with the use of auxilary structures (denoted in red) that are not connected to the main structure and that can be removed afterwards.

operations could be detected and substituted. For example, the following sequence: 'move left' 'move left' 'move right' could equivalently be 'move left'.
There are different fronts that could be explored, namely, physical constraints of the robotic arm, effective parameter search to determine optimal parameters such as mutation and crossover rate, growth bias rate, population size and generations.
From the L-systems approach, there are more interesting L-systems, namely context dependent ones where the production map **P** of one instruction depends on the instructions around it. For example, this could be useful for structures that are not solely composed by cubes but also actuators (or other elements)

Appendix A

Algorithm manual

This appendix aims to provide an overview of the algorithm and simulation code used in this project.

A.1 Simulation workflow

The simulation is started with the file **initialization.m**, where the default parameters are loaded from two files: **initPars.m** and **algoPars.m** - establishes the parameters for the algorithm, namely, production rule, activating obstacles, rendering.
The user has the choice to overwrite some of the initial default parameters. The algorithm is then run based on the established parameters. A structure containing a list of organisms (fittest organism per generation) and their corresponding fitness is obtained after the algorithm run and can be plotted and further explored.

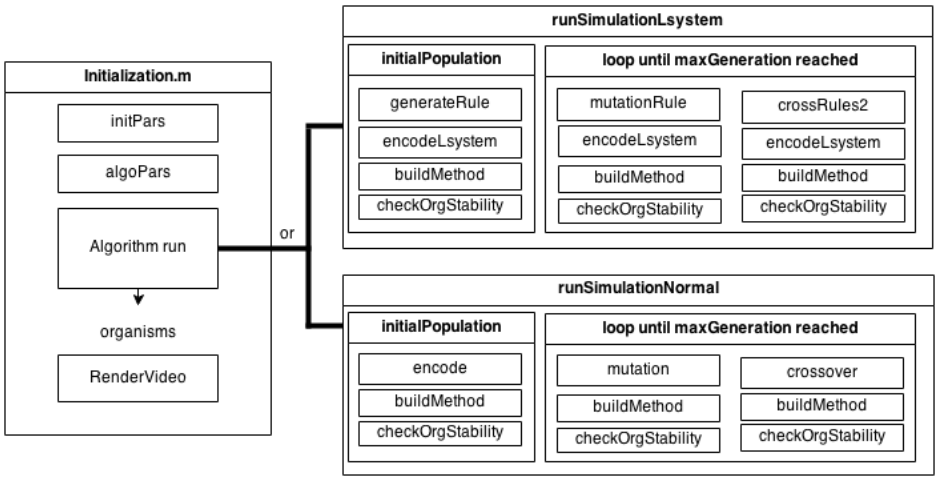


Figure A.1: Code workflow diagram.

A.2 Description of the functions

initParm.m - establishes the parameters for environment of the simulation, such as number of generations, population, target, obstacles

algoParm.m - establishes the parameters for the algorithm, namely, production rule, activating obstacles, rendering
initialization.m - file to be called to start the simulation

Production rule specific

Stochastic based generation

runSimulationNormal.m - contains the genetic algorithm evolving the building plans generated randomly
encode.m - generates the genome, returns a string
buildMethod.m - interprets a genome and produces an organism represented by the coordinates of each cube
mix.m - crossover operation for the SBG method
mutation.m - mutation operation for the SBG method

L-systems

runSimulationLsystem.m - contains the genetic algorithm evolving the building plans encoded by rules
generateRule.m - generates a set of rules to transform the alphabet
encodeLsystem.m - given a set of rules, generates the genome
crossRules2.m - crossover operation for the L-system method, by mixing complete rules
mutationRule.m - mutation operation for the L-system method
crossRules.m - crossover operation for the L-system method *(unused)*
crossOverRule.m - crossover operation for the L-system method *(unused)*

Common functions

checkFitness.m - ranks the organisms by descending fitness
compareStructures.m - compares the similarity between two structures
fitnessFunction.m - calculates the fitness of an organism
genecheck.m - checks if the genome is still buildable after crossover or mutation operations
checkOrgStability.m - encompasses the various stability checks
collisionCheck.m - checks for collisions of the structure against objects or target
stressCheck.m - checks for the stresses generated in the structure
tumbleCheck.m - checks if the structure will tumble
ST.m - Truss analysis

Rendering

makeEvolutionMovie.m - records a simulation run
createcube.m - creates a cube
renderStructure.m - creates a plot with the structure

Appendix B

Fitness Function

As quoted from the work done in [4]:

$$\begin{aligned} \text{fitness} &= W_{Distance} \times (1 - Distance) - W_{NumCubes} \times NumCubes - ... \\ &... \quad -W_{MaxStress} \times MaxStress - W_{MaxDisplacement} \times MaxDisplacement + ... \\ &... \quad +W_{NumConnections} * NumOfConnections \end{aligned} \tag{B.1}$$

- ***Distance*** *this value is normalized (current distance divided by distance from the origin to the target) so it is in range from 0 to 1. Since 1 corresponds to the highest distance from the target, 1 - Distance was used in fitness function.*
- ***NumberOfCubes*** *is not a normalized value so it can be any number bigger than 1. It is desirable that structure has less cubes and that is why there is a negative sign in front of NumberOfCubes parameter.*
- ***MaximumDisplacement*** *can be any number from 0 to an upper limit determined by the maximum stress in the structure.*
- ***MaximumStress*** *value is normalized (maximum stress is divided by critical stress) and it is in range from 0 to 1.*
- ***NumberOfConnections*** *the Value of this parameter is normalized with 6 × NumberOfCubes.*

Bibliography

[1] P. Funes, J. Pollack: *Evolutionary Body Building: Adaptive Physical Designs for Robots.* Artificial Life Volume 4, Number 4 (1998), p. 337-257.

[2] L. Brodbeck, L. Wang, F. Iida: *Robotic body extension based on Hot Melt Adhesives.* IEEE International Conference on Robotics and Automation (2012), p. 4322-4327.

[3] J. Seitz: *Autonomous design of modular robot extensions using an evolutionary algorithm.* BIR Lab, Swiss Federal Institute of Technology Zurich

[4] T. Novkovic: *Extension of an evolutionary design algorithm to complex tasks.* BIR Lab, Swiss Federal Institute of Technology Zurich

[5] G. Rozenberg,A. Salomaa: *The mathematical theory of L systems.* Academic Press, New York, (1980), chapter 4

[6] S. Hornby, J. Pollack: *Evolving L-systems to generate virtual creatures.* Computers and Graphics Volume 25, Number 6 (2005), p.1041-1048.

[7] F. Simeoni: *Evolutionary optimization of interplanetary trajectories: improvements from initial diversification.* Journal of Aerospace Engineering (2011), p. 1277-1288