UiO : Department of Informatics
University of Oslo

Adjustable legged robot

Morphological changes in the chase of an optimal parameter for gait optimization on walking robots

Sigbjørn Eide

Master's Thesis Spring 2017

Adjustable legged robot

Sigbjørn Eide

May 2, 2017

Abstract

——Write abstract here——

Acknowledgements

I would like to express my deepest gratitude to my supervisor, associate professor Mats Høvin, for continues guidance trough the work on this thesis. I would also like to thank PhD candidate Tønnes Nygaard for keeping your door open for interesting discussions, help with equipment and general guidance.

My fellow students also deserve a big thank you for creating a great environment for both studying and socialising.

Last but not least, I would like to thank my family, friends and co-workers for the endless support through this period.

Contents

1 Introduction **1**
- 1.1 Research goals . . . 1
- 1.2 Outline of the thesis . . . 1

2 Background **3**
- 2.1 Evolutionary Computing . . . 3
  - 2.1.1 Inspiration from Biology . . . 4
  - 2.1.2 Why use Evolution in Computer Science? . . . 6
- 2.2 Evolutionary Algorithms . . . 6
  - 2.2.1 Genetic Algorithms . . . 7
  - 2.2.2 Evolution Strategies . . . 9
  - 2.2.3 Evolutionary Programming . . . 10
  - 2.2.4 Genetic Programming . . . 10
  - 2.2.5 Evaluating Performance of Evolutionary Algorithms 11
- 2.3 Optimization and Search . . . 13
  - 2.3.1 Exhaustive Search . . . 13
  - 2.3.2 Greedy Search . . . 14
  - 2.3.3 Hill Climbing . . . 14
- 2.4 Memetic Algorithms . . . 14
  - 2.4.1 Local Search . . . 14
  - 2.4.2 Structure of a Memetic Algorithm . . . 16
  - 2.4.3 Adaptive Memetic Algorithms . . . 16
- 2.5 Evolutionary Robotics . . . 18
  - 2.5.1 Evolving robots . . . 18
  - 2.5.2 Reality gap . . . 19
  - 2.5.3 Recent work . . . 20
- 2.6 Robot gait . . . 21
  - 2.6.1 Static gait . . . 22
  - 2.6.2 Dynamic gait . . . 22

3 Software and tools **25**
- 3.1 Robot Design . . . 25
  - 3.1.1 SolidWorks . . . 25
- 3.2 3D Printing . . . 27
  - 3.2.1 Insight . . . 27
  - 3.2.2 Control Center . . . 27
  - 3.2.3 Fortus 250mc . . . 28

3.3 Electronics and Mechanics . . . 28
3.3.1 Dynamixel Servo AX-18A . . . 28
3.3.2 Stepper Motor . . . 29
3.3.3 Arduino Board . . . 32
3.4 Programming . . . 33
3.4.1 Processing . . . 33
3.4.2 Arduino IDE . . . 34

4 Implementation and experimental setup 35
4.1 Experimental setup . . . 35
4.2 Mechanics . . . 37
4.3 Version 0 . . . 38
4.4 Version 1 . . . 39
4.4.1 Modeling . . . 39
4.4.2 Simulation . . . 39
4.4.3 Creating movement . . . 40
4.5 Version 2 . . . 41
4.5.1 Modeling . . . 41
4.5.2 Simulation . . . 42
4.5.3 Creating movement . . . 43
4.6 Version 3 . . . 45
4.6.1 Modeling . . . 45
4.6.2 Simulation . . . 46
4.6.3 Creating movement . . . 47

5 Experiments and results 49
5.1 Experiments . . . 49
5.1.1 Fitness . . . 49
5.1.2 Version 1 . . . 50
5.1.3 Version 2 . . . 51
5.1.4 Version 3 . . . 52
5.2 Optimization with local search . . . 53
5.2.1 Hill climbing experiments . . . 53
5.2.2 Version 2 results . . . 54
5.2.3 Version 3 results . . . 55
5.3 Optimization with GA . . . 56
5.3.1 Genetic algorithm experiments . . . 56
5.3.2 Optimization with angle . . . 57
5.3.3 Optimization with angle and length . . . 63

6 Discussion 65
6.1 General discussion . . . 65
6.1.1 Challenges . . . 65
6.2 Conclusion . . . 65
6.3 Future work . . . 65

List of Figures

2.1 Terminology in natural evolution and problem solving . . . 4
2.2 Overview of a chromosome . . . 5
2.3 Overview of the general EA . . . 7
2.4 Example of 1-point crossover . . . 8
2.5 Example of bit-flip mutation . . . 9
2.6 Overview of the general MA . . . 17
2.7 Example of evolved robot . . . 19

3.1 The Fortus 250mc . . . 29
3.2 The AX-18A servo from Robotis . . . 30
3.3 The stepper motor from Mercury Motors . . . 30
3.4 The states of a H-bridge . . . 31
3.5 The EasyDriver from Allegro MicroSystems . . . 32
3.6 The Arduino Duemilanove board . . . 33

4.1 Experimental setup . . . 36
4.2 Example of performance measurement . . . 37
4.3 The 3D model of Version 0 . . . 38
4.4 Version 1 during simulation in Solidworks . . . 40
4.5 Timing diagram of the creep gait used on Version 1 . . . 41
4.6 3D model of Version 2 . . . 43
4.7 Version 2 during simulation . . . 44
4.8 Timing diagram of the human-like gait used on Version 2 . . . 44
4.9 3D model of Version 3 . . . 46
4.10 Version 3 during simulation . . . 47

5.1 Printed and assembled Version 1 . . . 50
5.2 Version 2 during an experiment . . . 52
5.3 Version 3 during an experiment . . . 53
5.4 Pseudocode of the hill climbing algorithm . . . 54
5.5 Pseudocode of the GA . . . 57
5.6 Comparison of the GAs run to optimize the gait with angle as parameter . . . 62
5.7 Max fitness for each generation for optimization with angle and length . . . 63

List of Tables

2.1 Sketch of simple GA 8
2.2 Sketch of Evolution Strategies 10
2.3 Sketch of Evolutionary Programming 10
2.4 Sketch of Genetic Programming 11

3.1 Overview of the software used in the thesis 25

5.1 Fitness of evenly distributed leg lengths on Version 2 55
5.2 Results of hill climber around position 24 on Version 2 55
5.3 Results of hill climber around position 32 on Version 2 55
5.4 Fitness of evenly distributed leg lengths on Version 3 56
5.5 Results of hill climber around position 24 on Version 3 56
5.6 GA1 - mutation rate 17% and crossover rate 70% 58
5.7 GA2- 1 - mutation rate 20 % and crossover rate 60% 59
5.8 GA2- 2 - mutation rate 20% and crossover rate 60% 60
5.9 GA3- 1- mutation rate 20% and crossover rate 0% 61
5.10 GA3 - 2- mutation rate 20% and crossover rate 0% 62
5.11 GA - length and angle 64

Chapter 1

Introduction

——Write introduction here——

1.1 Research goals

The main goal of this thesis is to investigate if adjustment of morphological parameters during the optimization process of a gait could increase the robot's performance. To test the performance, this thesis will use forward walking speed as a performance measure. The investigation involves designing and building multiple versions of a morphological adjustable robot, together with running optimization on the gait of the robot using evolutionary algorithms and local search. The hypothesis is that adjustment of morphological parameters, e.g. leg length, is a useful parameter for optimization of a robot gait. Thus, the idea is that using adjustment of the morphology as a parameter in the optimization, will result in increased performance compared to optimization of a gait on a robot with static morphology designed without a priori knowledge about morphology design.

1.2 Outline of the thesis

The thesis consists of 6 chapters: introduction, background, software and tools, implementation and experimental setup, experiments and results, and discussion.

Chapter 2 contains the background and theory behind the experiments of the thesis,together with some of the recent work in the topics of the thesis. Chapter 3 gives an introduction to the software and tools used to prepare and perform the experiments to test the hypothesis.

Chapter 4 describes the implementation done, and the experimental setup used, to perform the experiments. Chapter 5 explains the experiments, show the results and compares the results against the hypothesis. The last chapter, chapter 6, contains a general discussion about the results and the topics of the thesis, together with a conclusion and suggestions for future work.

Chapter 2

Background

This chapter gives an overview of the background and theory used in the thesis. This includes an introduction to evolutionary computing, evolutionary algorithms, memtic algorithms and evolutionary computing, together with methods to compare algorithms and a short introduction to gait classification. It also describes some of the recent work that is done in the fields of this thesis.

2.1 Evolutionary Computing

Evolutionary computing (EC) is a research field within computer science [1]. The name comes from the inspiration taken from the natural evolution. Natural evolution is a good source for inspiration because of how all the different animals are able to survive in their own environment. The most important part that is inherited from natural evolution, is the strategy to solve problems by trying and failing.

To understand how natural evolution could be useful for computer science, a brief description of a simple way to look at natural evolution follows. An environment in nature is filled with a population of individuals that try to survive and reproduce. In other words, there is a area in nature where a group of animals live, and their daily tasks are to fight for survival and extend their family. Each individual have a fitness, which is an expression of how well they perform their daily tasks, e.g. if an animal is slow and weak, it will have less chance to survive compared to fast and strong animals. When solving problems by the stochastic generate and test principal, a group of possible solutions is tested, where each solution is called a candidate solution. The quality of each solution relies on how well the solution solves the problem. The quality is used to determine the chance that the given solution will be used to try and create even better solutions. A comparison of the terms in natural evolution and problem solving in computer science is shown figure 2.1.

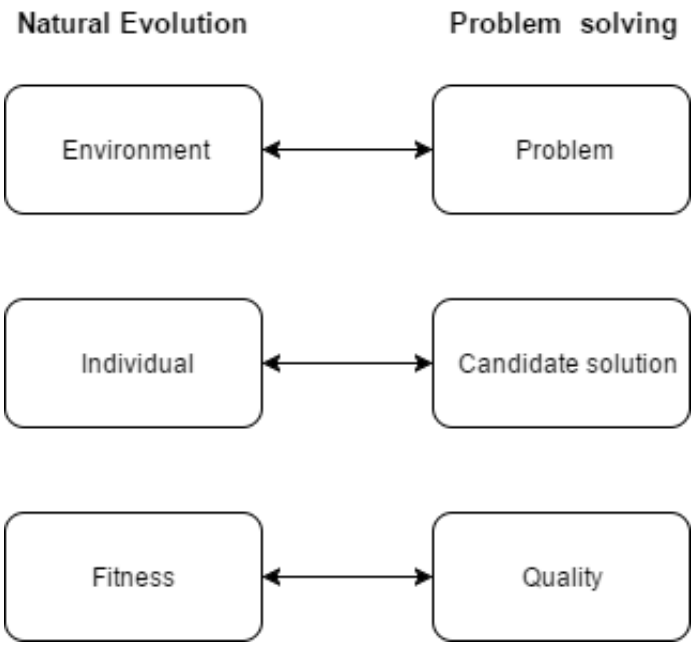

Figure 2.1: Terminology in natural evolution and problem solving

2.1.1 Inspiration from Biology

Darwin's theory of evolution [2] offers an explanation of how evolution in biology works. It also explains that natural selection is an important part of evolution. If the previously mentioned environment only have enough space for a given number of individuals, and they all want to reproduce, selection of who gets to reproduce is important to avoid an exponential growing population. The concept of natural selection, which says that the individuals that fit the environment best are favoured, is known as the survival of the fittest [1]. This type of competition based selection is one of the two most important principals of the evolutionary process. The other one comes from Darwin's results from phenotypic variations on members in a population. Phenotypic traits are the features of an individual that affect the relation to the environment and other individuals. This could be physical features like running fast or being strong as previously mentioned, or behavioural features like staying out of trouble to survive. Each individual have its own unique combination of phenotypic traits, which the chance of surviving and reproducing is a result of. The idea behind reproducing the individuals with the best phenotypic traits is that some traits are heritable. Darwin's idea was that with the help of small, random variations called mutation, the good and heritable traits could follow from generation to generation and create even better individuals.

Genetics

A better understanding of natural evolution could be achieved by taking a look at molecular genetics. The genetics will give an understanding of what is going on behind the scenes of the phenotypic features, especially when it comes to inheritance. The most important observation that could be made by looking at the genetics, is that each phenotypic property is represented at a genotypic level [1]. An individual's genotype is an encoding on the inside of what the phenotype is on the outside. This encoding could be inherited with the help of genes. Eiben and Smith [1] define genes as the functional unit that encode phenotypes. Eye color an body size are examples on physical attributes that could be determined by genes. When talking about inheritance, it is important to know the relation between genes and alleles. An allele is one of the values a gene could have, e.g. in a simplified gene for hair color, the allele could be blonde, brown or red. Genes and alleles are terms used to describe how a chromosome is built up. A chromosome is an array of genes, and all genes of an organism is stored in chromosomes. The relation between genes, alleles and a chromosome can be seen in figure 2.2.

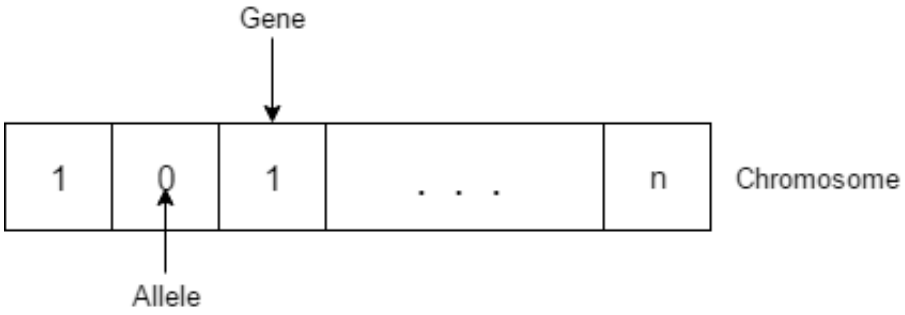

Figure 2.2: Overview of a chromosome (array of genetic data), where an allele is the values a gene could have (in this case 0 or 1), and a gene is an element of the array

In biology, the encoding from phenotype to genotype is not one-to-one. It is possible that one gene affects more than one phenotypic trait (pleitropy), and it is also possible that one phenotypic trait could be affected by more than one gene (polygeny). Difference in phenotypic traits is always caused by difference in the genotype, which is caused by mutation of genes, or recombination of genes by sexual reproduction. A version of the recombination in biology is used in EC under the name crossover. This means to combine the features from two individuals in an offspring. The crossover in EC does not work exactly as the crossover in biology, since the crossover in biology happens during formation of cells, and not during mating and reproduction. It is important to understand that all the changes are made in the genotype, and these changes are not visible for the naked eye, while the selection is based on observable traits; the phenotypic traits.

2.1.2 Why use Evolution in Computer Science?

To be able to solve problems automatically is one of the subjects in computer science that is given the most attention. In other words, being able to develop algorithms in computer science is important. It is common to look at nature to find possible ways to solve science problems. It is in particular two powerful problem solvers in nature that is focused on: the human brain, which came up with all the inventions in the world we know today, and the evolutionary process, which created the human brain. The former leads to the field of neurocomputing, which is defined as a reasearch field within computer science concerned with neural networks that develop associations between objects in response to their invironment [3], while the latter is the base for evolutionary computing [1].

Another type of motivation can be found by looking at computer science from a technical perspective. The increasing use of computers to solve different problems in the end of the 20th century, created a growing need for automatic problem solving. The research, as well as the development in this field, has not kept up with the increasing need of automation. Hence, there is less time to do thorough problem analysis, which is needed to to design problem specific algorithms. At the same time, the complexity of the problems to be solved keeps increasing. Together, these concerns implies a need for algorithms that can be used on many different problems, without to much adjustments for problem specifics, and delivering good results in reasonable time. This is where EC comes in. Evolutionary algorithms (explained in section 2.2) provide an answer to these concerns and include algorithms for an increasing base of problems with more and more complexity, solving them faster and faster.

2.2 Evolutionary Algorithms

There are many different types of evolutionary algorithms (EA). They have different operators, but the general idea behind these algorithms is the same [1]. They start out with initializing a population of individuals. The population then go through a parent selection, to select parents for recombination based on the current fitness. Recombination and/or mutation is used in the search of even better results. Recombination is applied on two or more different candidates. These candidates are the selected parents, while the results of the recombination is called children/offspring. Mutation is applied to one candidate and the result is one new candidate. The results from recombination and mutation go through survivor selection. Survivor selection is done by using a fitness function that make the fittest survive. The fitness function is testing the quality of the individuals and send some of the best on to the next generation. The same fitness function is testing the quality on the new generation of candidates, and the new candidates are compared against the old generation. In some algorithms the age is used as an extra parameter in finding the best candidates. The process from parent selection to survivor

selection is iterated until a solution is found or a decided limit of iterations is reached. The process of the general EA is shown in figure 2.3.

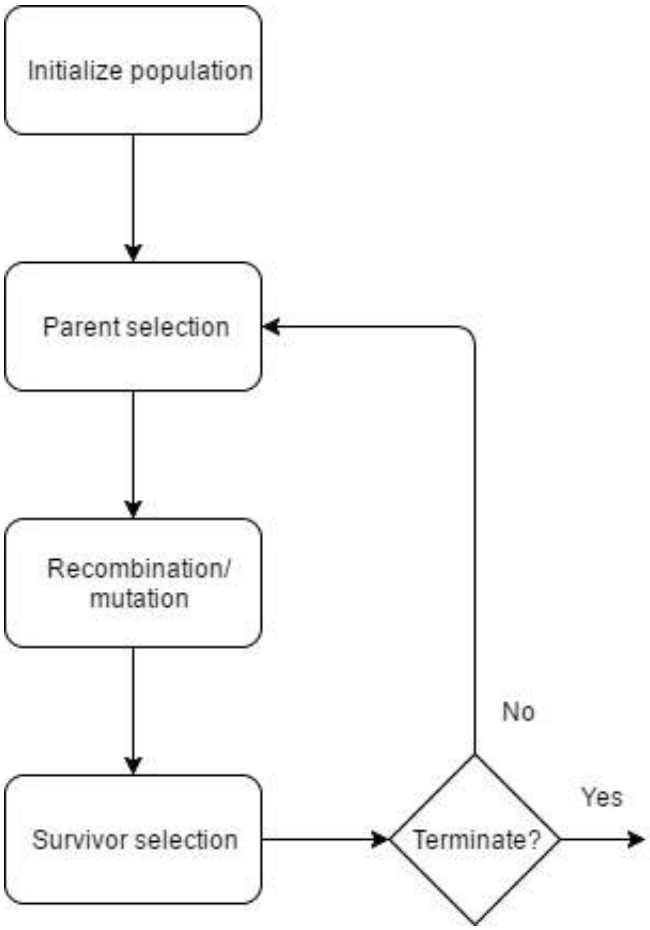

Figure 2.3: Overview of the general EA

2.2.1 Genetic Algorithms

Genetic algorithms (GA) are one of the most known subsets of evolutionary algorithms. It was initially discoverd by Holland while he was studying adaptive behaviour [1]. Genetic algorithms are considered as a method for function optimisation, although some strongly disagree [4]. The classic genetic algorithms we know today are more or less defined by the research of Holland and Goldberg [5]. These algorithms have a binary representation, roulette wheel selection, a low probability of mutation and are often referd to as simple genetic algorithms [1].

Representation	Bit-strings
Recombination	1-point crossover
Mutation	Bit flip
Parent selection	Fitness proportional - implemented by roulette wheel
Survivor selection	Generational

Table 2.1: Sketch of simple GA

Roulette wheel parent selection

The simple GA use roulette wheel parent selection. This is a selection method that selects the parents by a random selection where each individual have a probability to get selected based on their fitness. It works like a spinning a roulette wheel, where each individual have a fitness proportional part of the wheel. This could be implemented with the use of cumulative probability. If you have individual A with fitness 2, individual B with fitness 3 and individual C with fitness 5, A would have a probability of 2/10 to get chosen. B would have 3/10 while C would have 5/10. The cumulative probabilities would be A : 2/10, B: 5/10 and C: 10/10. By drawing a random number between 0 and 10 and checking if it is lower than first A, then B and then C, the first probability the random number is lower than, will be the parent.

1-point crossover

The simple GA use a recombination called 1-point crossover. This works by selecting a random number and taking the bits ahead of this point from the first parent, and the bits behind this point from the second parent to form the first child. The second child consist of the bits ahead of the point from the second parent, and the bits behind the point from the first parent. This recombination makes sure of sufficient exploration in the algorithm. To not get too much randomness, a crossover probability decides if a generation should go through recombination. It is normal to use a crossover rate at $60 - 80\%$ [1]. An example of the 1-point crossover is shown in figure 2.4.

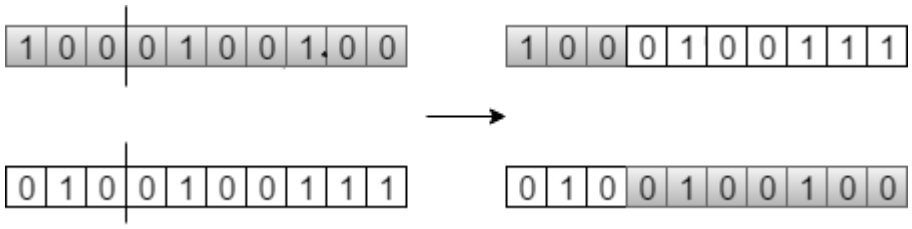

Figure 2.4: Example of 1-point crossover where the crossover point is ahead of the fourth bit

Bit-flip mutation

The most common mutation operator for binary representations, is the bit-flip mutation [1]. Thus, this is the mutation used by the simple GA. The bit-flip mutation considers each bit individually, and gives each bit a probability of flipping between 1 and 0. In other words, if a bit is selected to mutate, it changes the value from 0 to 1 or 0 to 1. This mutation, as mutation in general, makes sure that the algorithm have sufficient exploitation. It is common to use a low probability of mutation, and according to Eiben and Smith [1], it is normal to use a mutation rate between one gene per offspring and one gene per generation. An example of how the bit-flip mutation works is shown in figure 2.5.

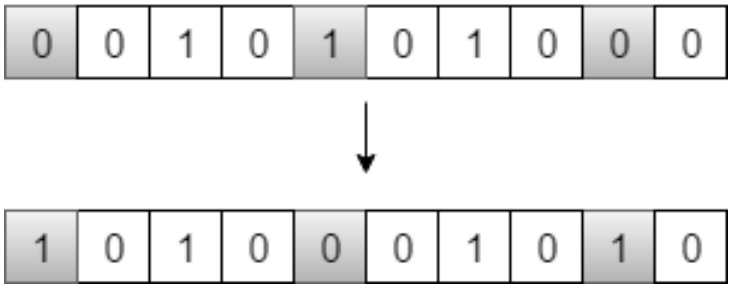

Figure 2.5: Example of bit-flip mutation where the bits marked grey were selected for flipping

2.2.2 Evolution Strategies

Evolution strategies (ES) were invented by the two German scientists Rechenberg and Schwefel in the early 1960s. They were studing at the Technical University of Berlin, and worked on an aplication concerning shape optimisation [6]. The first evolution strategies were simple two-membered (1+1) ES's (one plus one ES), working in a vector space. The one plus one ES generates an offspring by adding a random number to each of the elements of the parent vector. Followed by a fitness test which decides if the offspring is to be accepted. An alternative strategy in the early stages was the (1,1) ES (one comma one ES). This strategy always replaces the parent with the offspring. The random numbers are drawn for a Gaussian distribution with a mean zero and a standard deviation σ, called the mutation step size [1]. One of the key breakouts in the early stages of the ES was the $1/5$ success rule by Rechenberg. This was used as a simple method for on-line adjustment of step sizes.

The evolution strategies have evolved over the years, and in the 1970s the concept of multi-membered ES with $(\mu + \lambda)$ and (μ, λ) notations was introduced. This notation is based on μ individuals in the population and λ offspring generated in one cycle. These notations led to the development of a useful feature in evolutionary computing: self-adaptation of strategy parameters.

Self-adaptation of strategy parameters means that some parameters of the algorithm are changed in a specific way during a run. The parameters are included in the chromosomes and interact with the solutions. The notation of an ES chromosome can be expressed as <x,p>, where x is the object variables and p is the strategy parameters. Since self-adaptation was discovered, most ESs have been self adaptive. The feature is also more and more commonly adopted by other EAs.

Representation	Real-valued vectors
Recombination	Discrete or intermediary
Mutation	Gaussian perturbation
Parent selection	Uniform random
Survivor selection	Deterministic elitist replacement by (μ, λ) or $(\mu + \lambda)$

Table 2.2: Sketch of Evolution Strategies

2.2.3 Evolutionary Programming

Evolutionary programming (EP) came to life in the 1960s. It was developed by Fogel et al., and used to simulate evolution as a learning process while trying to generate artificial intelligence [7]. The classic EP algorithm used finite-state machines as individuals, compared to the real-valued representation that is normal today. Evolutionary programming has nearly merged with evolutionary strategies, but there is still a difference in the biological inspiration.

One of the differences in biological inspiration is that one point in the EP search space stands for a species and not an individual. Thus, EP has no recombination. Each parent generates one offspring and the whole population compete in stochastic round-robin tournaments for survival. Because of this open and realistic approach, the representation and mutation should be driven by the problem.

Representation	Real-valued vectors
Recombination	None
Mutation	Gaussian perturbation
Parent selection	Deterministic(each parent create one offspring)
Survivor selection	Probabilistic $(\mu + \mu)$

Table 2.3: Sketch of Evolutionary Programming

2.2.4 Genetic Programming

Genetic programming (GP) is one of the last found EAs. It uses tree structures as representation, and is often used for machine learning. While other EAs are looking for input that gives maximum payoff, GP algorithms are looking for models with maximum fit.

GP often uses the so-called ramped half-and-half method for initialization. In this method a maximum depth of the trees is chosen, and each member of the initial population is created using the full method (each branch has maximum depth) or the grow method (branches can have different depth, up to the maximum depth).

In GP, offsprings are typically created by either recombination or mutation, compared to recombination and mutation in other EAs. It is recommended to use a small rate of mutation. Koza's research [8] suggests that genetic programming works without mutation, while the research of Banzaf et al. [9] suggests a mutation rate of 5%.

Representation	Tree structures
Recombination	Exchange of subtrees
Mutation	Random change in trees
Parent selection	Fitness proportional)
Survivor selection	Generational replacement

Table 2.4: Sketch of Genetic Programming

2.2.5 Evaluating Performance of Evolutionary Algorithms

To evaluate the quality of an EA it is common to compare the given algorithm to other traditional EAs. The quality of an EA is mostly compared by the performance measure of the algorithm. Because EAs are stochastic, a number of experiments needs to be evaluated to get enough data to measure the performance of a given algorithm.

Performance could be measured in many ways. Eiben and Smith[1] mentioned three different performance measures. Success rate (SR), mean best fitness (MBF) and average number of evaluations to a solution (AES) are all methods used to measure the performance of EAs. When an experiment concerns a problem where the optimal solution can be recognized, or where sufficient solution quality is given, finding a solution of the desired quality could be measured by SR. SR is measured by the percentage of runs with sufficient solution quality. If a problem does not have an optimal solution, SR could in theory not be used to measure the performance. This is the case for problems where the optimal solution is unknown, and the EA is used to find the best possible solution. In practical problems a success criteria could by given in a way that makes SR a valid performance measure, even if the optimal solution is unknown. This is often done by comparing the solution to a previous solution and using improvement as a success criteria.

Unlike SR, the MBF could be used for any kind of problem. The fitness of the best individual is captured after each run, and the MBF is the average of these over all runs. There is no general rule on when to use SR or the MBF to measure the performance of an EA. The biggest difference between them are that SR is valid for a limited number of problems, while the MBF is always a valid measure. There is no relation between the result of SR

and the MBF. If the SR is really good or really bad, it does not give any indication on what the MBF is. Even if the MBF is the average of the best fitness of all runs, the best-ever or worst-ever fitness would be more interesting for some problems. This depends of the goal of the algorithm. For design problems the best-ever fitness could be very interesting since one good solution is all that is needed, while for repetitive problems the worst-ever could be used study worst-case scenarios. It is important to take into account that both SR an MBF measure performance within a specified maximum of computing. If the specified maximum changes, the ranking of the algorithms could change as well. SR and MBF measures the performance of an EA regard effectiveness, where effectiveness is how far the algorithm can get within the given limit.

Another way to measure performance is in terms of efficiency or speed. Speed is often measured in computer time, CPU time or user time. These types of measures depends on the equipment being used, and are therefore not suited for reproducible research. A common workaround for this problem is to count the number of points visited in the search space. Because of the immediate evaluation in EAs, this measure is normally expressed as the number of fitness evaluations. Because of the stochastic nature of EAs, the measure is always based on a number of independent runs. This is where the average number of evaluations to a solution (AES) are used. In AES, it is only the successful runs that are used to calculate the average. Because of the dependency on the definition of a solution, AES suffers from the same limitations as SR. It can also be misleading in some cases. Extra computational effort that increases performance, big difference in evaluation time and very quick evaluations are examples of cases which gives AES some difficulties.

Statistical Analysis

To compare the measured performance of two EAs, or two different configurations of the same algorithm, a statistical test that shows a significant difference between the two should be used. The number of different methods used in the field over the recent years have proved how important it is to use statistical analysis when comparing two algorithms. Statistical analysis usually uses parametric tests based on average and variance, but recent research have also tested the use of non-parametric tests to analyse results [10]. To be able to use parametric tests, there should be enough knowledge about the problem to make accurate assumptions. If this knowledge is missing, non-parametric tests would be a better fit, since non-parametric test does not require specific conditions to compare two algorithms.

One of the main test methods used in statistical analysis are hypothesis testing. The two complimentary hypotheses in statistical hypothesis testing are the null hypothesis H_0 and the alternative hypothesis H_1. H_0 is the hypothesis that the data is taken from the same population, which means that there is no significant difference between the data sets. As the complimentary, H_1 is the hypothesis that there is a difference, that the data

is taken from different populations. To show statistical significance, it is normal to look for a way to conclude that H_0 is very unlikely. This is often done by calculating a p-value. The p-value could be calculated as a function of the result of a statistical test. The p-value is used to show if a test is statistical significant or not. This is shown by comparing the p-value to a chosen threshold, where lower than 5% is a commonly used threshold. The p-value could also be used as an indication on how significant the test is. To increase the probability of rejecting H_0, a large sample size is preferred.

Data visualisation

There are many ways to look at the results from an EA. The results could e.g. be shown in tables, but it is often preferred to visualize it. Visual evaluation could e.g. be done by plots that show how the population have developed through the evolution. This could be shown with curves of the mean fitness in every generation through the evolution. To be able to use the visualization to compare different runs, the runs should have an equal number of generations. This will show the average performance of the algorithm, and how efficient it is. To show the robustness of the algorithm, the fitness variance could be plotted instead. High variance means low robustness and visa versa.

To see if the diversity of the population in the algorithm was large enough, it could be useful to look at a plot of all the fitness values through one run. EAs often depend on a large diversity to give a good result, since it could get stuck at a local maximum if the individuals are to similar. The plot of all the fitness values through one run could also be used to see if the best-ever or worst-ever fitness value was as desired. The best-ever could e.g. be useful in design problems.

2.3 Optimization and Search

In an optimization problem the model is known and the desired output is specified. The task of the problem is to find the input(s) that leads to the desired output [1]. There are several ways to use a search to try to solve an optimization problem. In this context a search is a way to attempt optimization without using gradients. There are three basic approaches to optimization using search. These three methods are described next.

2.3.1 Exhaustive Search

An exhaustive search is a search that tries every possible solution and picks the best one. This is guaranteed to find the global optimum, but it is not a good option for most problems with a bit of complexity. Since it tries out all possible solutions it could be very time consuming. The computational complexity is often worse than exponential, and for most problems it's not a reasonable method to use.

2.3.2 Greedy Search

The greedy search is computational much cheaper than the exhaustive search. It passes through the system one time, and makes the best local choice available at each stage. This method depends a lot more on the starting point and is not guaranteed to find the best solution in any way. This is a simple algorithm, but could give a fairly good result if you run it several times with different starting points. This will hopefully get you close to the global optima.

2.3.3 Hill Climbing

The hill climbing algorithm performs a local search close to the current solution. It moves on by selecting any option that improves the result. The initial solution could be picked randomly, and could have a big impact on how good the end solution is. The algorithm stops after a pre-defined number of iterations, or if it have not improved in a pre-defined length of time. There is no way to predict how good the solution will be. This is because the hill climbing algorithm, as most local searches, could get stuck at a local optima.

2.4 Memetic Algorithms

The No Free Lunch theorem[11] says that there is no such thing as a general efficient problem solver. In evolutionary computing this means that the performance of each EA depends on the problem, and there is not one superior algorithm for all problems as suggested in the 1980s [1]. To be able to get even better solutions using EAs, there are several concepts of combining different algorithms. Combining problem-specific heuristics and EAs, a hybrid EA with two EAs and combining local search with an EA are examples on how this could be done.

The idea of memes was introduced by Dawkins [12]. Memes are a way to describe units of transmission, as genes are used to describe units of biological transmission. A meme could be described as a little piece of a puzzle that is needed to create the whole picture, where the whole picture is e.g. a knowledge or a skill that is developed by learning. Moscato [13] came up with the name memetic algorithm (MA) to cover several techniques where EAs are combined with one or more phases of local search, or with problem-specific information.

2.4.1 Local Search

Local search algorithms work by picking a starting point, and searching the neighbourhood for a better solution. If a better solution exists, then this is accepted as the current solution. the search continues in the neighbourhood of the new, currently best, solution. If no better solution is found in the neighbourhood of the current best solution, the search terminates. This search will eventually find a local optimum: the best solution in the

neighbourhood. These algorithms are often quick to find a local optima, but there is no guarantee that this solution is any where near the global optima[1].

Local search used in MAs are a bit different. This local search algorithm have three important variables that affect the result. These three variables are described next.

The first variable is the pivot rule, which is the termination condition for the inner loop. The pivot rule could be chosen between steepest ascent and greedy ascent, where in the former, the termination condition is that the whole neighbourhood has to be searched, while in the latter, the termination condition that an improvement is found. If the neighbourhood is too large to search though, a small, random part could be used instead.

The second variable is the depth of the search, which is the termination condition for the outer loop. The depth could by chosen from one iteration to search until a local optima is found. There is done a fair amount of research on changing the depth on the local search in MAs [14]. The research implies that changing the depth could effect the performance in terms of time and quality of the solution.

The third variable is the function for generating the neighbourhood. This is considered as the most important variable, and is often defined as a set of points that are reachable from the current point with some move operator. Merz and Freisleben [15] show that the move operator is important in terms of efficiency and effectiveness of the local search, and thus also the result of the MA.

Lamarckianism and Baldwin Effect

In the local search algorithm described above, the current solution is always replaced by a fitter neighbour if found. In MAs, the local search is used as an improvement or a learning phase within the evolutionary cycle. Learning from biology, it should be considered if the individual changes should be kept in the genotype, or if the improved fitness should be awarded the original member of the population [1].

Whether individual's offspring were able to inherit learned traits, or not, was a major issue in the nineteenth century [1]. Lamarck was arguing in favour, while Baldwin had his own idea with the Baldwin effect [16]. The Baldwin effect suggests a mechanism where evolutionary progress can adapt without changing the fitness of individuals by learning, thus without changes in the genetic characteristic. The latter idea is strongly favoured in modern theories in genetics. Eiben and smith [1] show an example of why the Baldwin effect is favoured, where the mapping from DNA to protein is explained. They also explain why this makes the Baldwin effect more credible than Lamarckianism when it comes to genetics.

Since a MA is a computer algorithm, it is not restricted by these previously described biological constraints. Thus, both Larmarckianism and the Baldwin effect is usually possible to implement in a MA. MAs are often named Lamarckian or Baldwinian, where the former is used if the result of the local search replace the individual in the population, while

the latter is used when the original individual is kept with the fitness of the result from the local search. It has been done a lot of research on Lamarckianism and the Baldwin effect, e.g. by Hinton and Nowlan[17], which showed that the Baldwin effect have a nice effect on the evolution of artificial neural networks. Some of the research also include comparison of the benefits of the two [18, 19]. In most recent work with MAs, the trend is to use only Lamarckian, or a combination of the two, where the improved fitness is used, and the original individual is replaced by a better one with a given probability.

2.4.2 Structure of a Memetic Algorithm

There are many different ways an EA could be used together with local search or problem specific knowledge. In this section the structure of a MA that use local search on the individuals in the population of the EA will be discussed. Eiben and Smith [1] show examples of other MAs using e.g. heuristic or intelligent initialization and intelligent crossover and mutation.

Using local search on the individuals in the population of the EA is the most common way to hybridize an EA. This is also what fits best with Dawkins' concept of memes, which is described in the introduction to section 2.4. The local search could be used both after the initialization of the population, and after recombination and/or mutation. These MAs start out by initializing the population, which often is done randomly or by using problem specific knowledge. Local search is then performed on the individuals in the population to increase the quality of the population before entering the main loop. The population then go through a parent selection, to select parents for recombination based on the current fitness. Recombination and/or mutation is applied next, to try to increase the fitness. It is then performed a local search on the offspring to improve the fitness even more. Each individual will now be close to a local optima. The best individuals will be sent to the next generation, and this process is iterated until a solution is found or a decided limit of iterations is reached. The process of the general MA is shown in figure 2.6, and could be compared to figure 2.3 to see the difference between the MA and the clean EA.

2.4.3 Adaptive Memetic Algorithms

When designing a MA using heuristic improvements or local search, it is important to the correct way to improve heuristics, or chose the correct move operator for the local search. In other words, it is important to chose the correct way to generate the set of neighbouring points. In section 2.4.1, it is briefly mentioned that the move operator is important in terms of efficiency and effectiveness of the local search, and thus the MA.

To avoid the difficulty with choosing the correct move operator for every problem, multiple local search operators could be used together. Krasnogar and Smith [20] introduced what they called "multimeme" algorithms, where multiple local search methods were used together

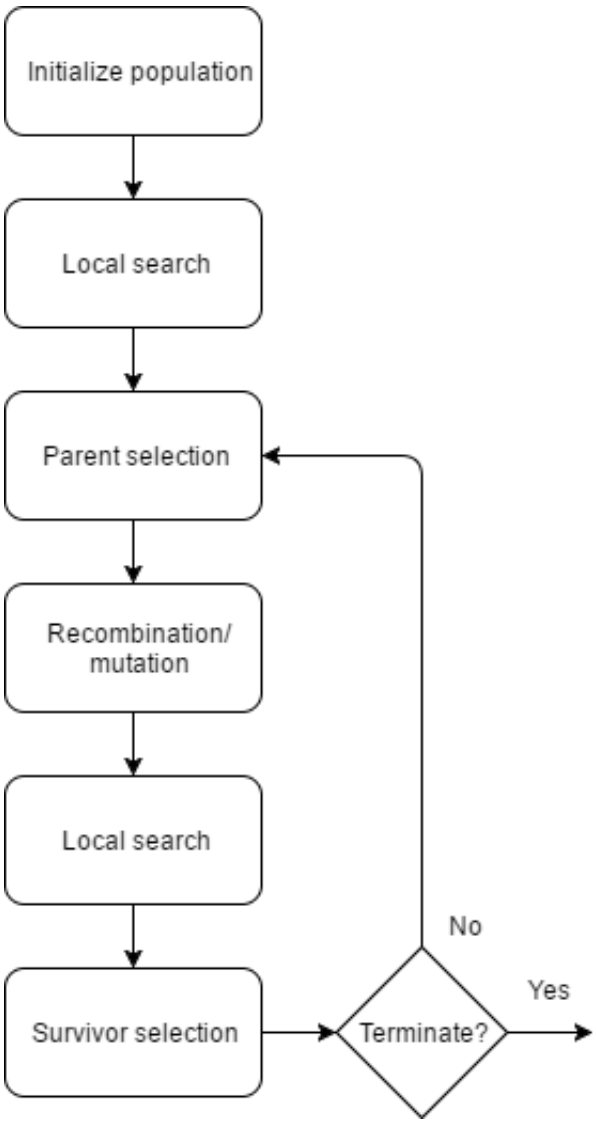

Figure 2.6: Overview of the general MA

with the EA. This method also included a way to chose between the different local search methods, depending on their usefulness. In this case they used self-adaptation, where each individual carried a gene with information on what local search algorithm to use. Ong et al. [21] presented a classification of the different methods in what they named "Adaptive Memetic Algorithms". This classification was later reviewed by Neri and Cotta [22], where they presented an updated version of the classification. The updated classification suggests the following four categories; Adaptive Hyper-heuristic, where coordination of memes is done according to heuristic rules; Meta-Lamarckian learning, where the results of each local search effect how often this specific local search is used; Self-adaptive and Co-Evolutionary, where the memes are a part of the evolution, which includes to go through recombination and selection;

Fitness Diversity-Adaptive, where a measure of diversity is used to chose between memes.

The essentials in these methods are that they use a pool of local search operators that compete to be used by the algorithm. Based on the described ideas, Meuth et al. [23] suggested the following classification; First-Generation MAs, defined as "Global search paired with local search"; Second-Generation MAs, defined as "Global search with multiple local optimizers. Memetic information passed to offspring"; Third-Generation MAs, defined as "Global search with multiple local optimizers. Memetic information passed to offspring. A mapping between evolutionary trajectory and choice of local optimizer is learned". It is worth noticing that the focus of many of these algorithms show some of the most common design issues for implementing MAs.

2.5 Evolutionary Robotics

Evolutionary algorithms (EA) are used for different topics in robotics. In Evolutionary robotics (ER), EAs are used to optimize some or all aspects of an autonomous robot [24]. The use of EAs is what differs this part of robotics from the mainstream robotic field, which primarily use machine learning to optimize some aspects of the robot. The goal of mainstream robotics is to keep increasing the performance of a given robot, while the long-term goal of ER is to create general algorithms for creating robots. As most things in computer science, using EAs have a cost and a benefit. The cost is that optimization with an EA can never guarantee that an optimal solution is found. It is often stuck in a local optima, and it is impossible to tell how long it will take to find an optimal solution for a given robot. The benefit is that few assumption must be made about the given problem. EAs can improve both the parameters as the robot gait(walking pattern), and the robot morphology(the shape of the robot).

An EA often runs through many generations to find a good solution. Thus, optimization in ER is often first carried out in simulation. Typically an EA generates a population of virtual robots. Each robot is evaluated and get a fitness based on the evaluation. These robots compete against each other, and the fittest may be physically assembled to test the performance in the real world. When using evolution, the researcher may not have to make so many detailed decisions about the design. This could lead to designs that are not obvious for the researcher, especially for designs that are non-intuitive for humans. Bongard [24] offers a wider description of the field of ER.

2.5.1 Evolving robots

The use of evolution in robotics is previously associated with evolving control policies. As previously mentioned, the long-term goal of ER is to be able to use evolution to design robots. There is done some research on the topic of evolving morphology of the robots, or even better, a combination

of both morphology and control policies. Sims [25, 26] used genetic algorithms to evolve both morphology and control policies of robots. These robots were eventually able to e.g. walk, jump and swim, which is a proof that it is possible to use evolution to create actual working robots. This was backed up by the similar type of robots in the GOLEM project [27]. To show an example of what an evolved robot could look like, one of the robots used by Samuelsen [28] is shown in figure2.7.

Figure 2.7: Example of evolved robot

2.5.2 Reality gap

Although it is normal to use simulation in robotics to test the models, the validity of simulations is highly debated when it comes to building robots. Simulations may be very useful to test different models, but Brooks [29] pointed out that it is hard to simulate the actual dynamics of the real world, and it could end up as a waste of time to solve problems that don't exist in the real world. He also mentioned that programs that works well i simulation, might completely fail in the real world. There are several problems that could arise when using computer models to develop control systems for real robots [30]:

- Simulation usually don't manage to take account for all the physical laws that affect the robots in the real world, e.g. mass, weight, friction.
- Physical sensors deliver values based on a certain preciseness, and commands to actuators have an unknown precision, while simulation often use sensors and actuators that give perfect feedback.

- Different physical sensors and actuators might deliver different performance, even if they are supposed to be identical. This may be caused by minor differences in the electronics or mechanics or the placement on the robot. This difference is often ignored in simulation.

This difference between the performance in simulation and the real world is called "the reality gap". Jakobi et al. [31] did some experiments on the subject and came up with the idea to add noise to the simulation. This could make the simulation come closer to the real world results, if the level of noise is reasonable. Zagal and Ruiz-Del-Solar [32] came up with a different approach. They came up with what they called a "back to reality" algorithm. The idea is to co-evolve the simulator and the robot at the same time. This includes performing regular validations and adaptations of the model in simulation while the robot is adapting. There is still some research being done on the reality gap, but the perfect solution to decrease the gap is yet to be found.

2.5.3 Recent work

Morphological changes to accelerate the evolution of robust behaviour

In 2011, Bongard [33] released a paper about how to use morphological changes to accelerate the evolution of robust behaviour. Here, the effect of different morphological changes like parametric change and phylogenetic change is described. The former indicates that the angle between the legs and the trunk of the body could be changed, while the number of body parts and degrees of freedom remain the same. The latter indicates that the body plan does not change during the evaluation, but may change between evolutionary generations.

By running a total of 5000 independent evolutionary trials, the effect of morphological change was measured on two different robots with five different types of morphological change, including the two previously mentioned, and five different environments. The results of these trials proves that morphological change could make the achievement of behaviour happen faster, and that the achieved behaviour is more robust. The morphological change have effect both if it is done over evolutionary time, and if it is done over the lifetime of a single organism.

Real-World Reproduction of Evolved Robot Morphologies

In 2015, Samuelsen and Glette [28] released a paper about real-world reproduction of robots evolved through simulation. The goal of the paper was to demonstrate that morphology-based ER is a good approach to generate physical robots for different purposes. This was done by presenting a complete construction process for physical robots based on simulation results. The robots were generated by EAs in simulation, and assembled physically by the use of off-the-shelf motor components together with 3D printed parts based on the models from simulation.

In a previous experiment [34], 42000 solutions were produced in simulation. Because of time and resources, five of these were selected to be printed out and tested in the real world. The selection was done based on different criteria, e.g. representative for a larger group of robots, forward walking performance etc.

To be able to compare the performance of the physical robots to the ones from simulation, the same evolutionary procedure was used. The robots were originally evolved to walk forward as fast as possible, thus, forward walking speed was a natural choice of performance measure. By running 50 evaluations of 4 seconds on each robot both in simulation and hardware, the performance of the physical robots were compared to the ones from simulation. The results of these evaluations show that the difference in performance from simulation to hardware varies from morphology to morphology. This confirms that there is still some work to be done to shrink the reality gap between simulation and physical robots in the real world. Adding noise in simulation and underestimating physical forces, e.g. friction, was suggested as possible improvements.

Memetic Robot Control Evolution and Adaption to Reality

In 2016, Ruud, Samuelsen and Glette [35] released a paper about memetic robot control evolution and adaption to reality. The hypothesis of this paper was that learning can improve evolution by moving individuals closer to local optima during evolution, and this would lead to increased quality on the solutions. To test this hypothesis, evolution was run with and without learning under equal conditions. Both Baldwinian and Lamarckian learning were tested in the evolution.

The experiments were done with a four-legged robot evolved in a previous experiment [28]. To optimize the robot control system parameters, a memetic algorithm based on a genetic algorithm together with a simple local search were used. The parameters were optimized for maximum forward walking speed, and this speed was also used as performance measure.

The performance was measured with a motion capture system for position and orientation measurements. This was done by running 1600 evaluations on each individual, which lead to a comparison of the different configurations. This comparison show that when a sufficient number of iterations were used, Lamarckian learning had a positive effect on the evolution. The results also show that Baldwainian learning, in these experiments, had a negative effect on the evolution, performing worse than both evolution with Lamarckian learning and evolution without learning.

2.6 Robot gait

There are many ways a robot could move. It could move on wheels, by crawling, by jumping, and, as in this thesis, it could move by walking. Walking is a general term of moving the legs to create movement. This

could be done in many different ways. By taking a closer look at a creature that walks, it is often possible to see a pattern. This pattern is known as a gait. By looking at different animals, inspiration for many different gaits can be found. A horse is a good example because of its ability to move with the use of many different gaits, e.g. trot, gallop and canter.

To classify robot gaits, this thesis use two general classifications; dynamic gait and static gait. The main criteria used for classification in this thesis is the stability of the robot while moving. The stability of the robot is an important factor in the design of robot gaits. To be able to get a fair comparison of the performance of the robots in the validation of the hypothesis, the gaits in this thesis lean as much as possible to one of the classifications. The two classifications are explained in the following sections, together with the difference between them.

2.6.1 Static gait

A static gait could be defined as a gait where the body is balanced at all times. For the body to be stable at all times, is known as static stability. The idea of static stability in gaits was inspired by insects. Insects use their massless legs to support their body while they create movement. To be able to keep the body in balance, the legs move in a way that ensure static stability [36]. Some of the earliest walking robots that were made, e.g. by Kumar and Waldron[37], used this type of a static gait to create motion. These early robots, as the robot made by Song and Waldron [38], consisted of huge and heavy mechanics that made them hard to control. By using the static stable gait principle from the insects, they were able to simplify the control of the robot. This resulted in restrictions in the robot's ability to move, in terms of inertial effects, friction, elasticity and so on.

The fact that the use of static stability can lead to improved motion control at the price of speed, is one of the main reasons why mainly static gaits will be used to test the hypothesis of this thesis. To be able to compare performance and get fair results, it is more important that the robots will perform with a high precision compared to achieving very high speed. This is because the robots will be compared to them selves with changing morphology, and the speed of the servos will be constant. Some humanly inspired dynamic gaits will also be tested, but these gaits will lean as much as possible against static stability.

2.6.2 Dynamic gait

If the intention of a robot is to get it to walk as fast as possible, the dynamics of the robot regarding stability have to be considered. Since static stability limits the walking speed, higher speed could be achieved with allowing the robot to get unstable. The principle of dynamic gaits is to allow the body to be unbalanced to use gravity to create faster movement. If a robot walks with a dynamic gait and the power is turned of, the robot will most likely fall over because of the lack of balance. This type of gaits are often made

with inspiration from the previously mentioned horse gaits like trot and gallop, human inspired gaits or other gaits from various animals.

Dynamic gaits may not fit all purposes, even if speed is often a desirable criteria. If the purpose of the robot is to use it in different types of industry, it might not be a good thing with more speed in the cost of control. If a robot in the car industry is set to build cars, it is more important that it does the same good job on each car, compared to making 2 cars faster, but destroying the third because of lack of control. Another example is if a robot has to be able to move on different surfaces, e.g. in space. Then it is more important that the robot is able to move, compared to the robot's ability to move very fast. This show that both dynamic and static gaits are functional in their own areas.

Chapter 3

Software and tools

This chapter presents the tools and software used to implement and perform the experiments of the thesis. This includes designing and simulating the robots, printing the robot parts, programming the different servos and motors and measuring the performance of the assembled hardware. Table 3.1 show an overview of the software used in the thesis, together with area of use and the version of each software.

Area of use	Software	Version
Robot design and simulation	Solidworks	2015 Education Edition
3D Model slicer	Insight	10.2
Send jobs to the 3D printer	Control Center	10.2
Programming servos	Processing	3.3
Programming step motors	Arduino	1.8.1

Table 3.1: Overview of the software used in the thesis

3.1 Robot Design

The different robots used to test the hypothesis is designed as 3D models of the different parts needed to assemble the physical robots. The robots are designed manually, with a presented model from the supervisor as a starting point.

3.1.1 SolidWorks

Solidworks[1], made by Dassault Systèmes, is a well known computer aided design (CAD) tool used to draw 2D drawings and design 3D models. It includes a lot of advanced features, but it is also fairly intuitive to use on a certain level. Solidworks works with parts and assemblies, where an assembly consists of several parts assembled together in relation to each other. In this thesis, Solidworks is used to design the 3D parts for the physical robots, and to simulate the complete robots as assemblies. The

[1] http://www.Solidworks.com/

basic technique used here, is to begin with a 2D drawing and extrude it into a third dimension. The possibility of inserting an existing part when designing a new one, is a useful feature when creating parts that should work together. This makes it easier to get all the measurements correct in relation to making everything fit together.

Simulation

Solidworks also offers a wide range of simulation possibilities, e.g. simulation of fluid flow, heat transfer, thermal analysis of moving air and liquid and so on. When it comes to motion studies, Soldiworks offers three different tools; animation, basic motion and motion analysis. These three tools includes the following features:

- The animation tool is available in the core of the normal version of Solidworks. Animation is used to animate the motion of assemblies. It is possible to add motors to drive the motion of several parts of the assembly. It is also possible to set positions on a given time by using key points on the timeline.
- The basic motion tool is also available in the core of the normal version of Solidworks. Basic motion includes the features from animation, but also offers a wider aspect of features that could be useful in simulation. This are features like the effects of springs, contact and gravity, which implies that basic motion is able to use mass in it's calculations.
- The motion analysis tool is an add-in to the premium version of Solidworks. Motion analysis is an even more advanced tool. It works much in the same way as the two others, but it is used for even better, and more accurate, simulation and analysis of the effect different motions have on the assembly. Motion analysis is able to simulate the effect of e.g. forces, dampers and friction. It uses advanced kinematic solvers and make use of material properties, mass and inertia in the calculations.

In this thesis, the motion analysis tool is used. This is because the motion analysis is able to simulate friction in addition to all the other features that basic motion also have. The simulation is mostly used to see if a given gait will be able to make the robot walk forward if there is enough friction. It is also used to see if the robots will be able to achieve sufficient balance, and to experiment with different gaits. Since friction and mass is hard to simulate, there will be a reality-gap, but all the advanced features of the motion studies in Solidworks helps to make the gap as small as possible.

3.2 3D Printing

The designed 3D models are printed to be able to test their physical performance. 3D printing is a great tool for applying the principle of rapid prototyping. Rapid prototyping means to spend little time on finding the perfect design, but rather printing the model to test the physical abilities in the real world.

3.2.1 Insight

After a 3D design is made in a CAD tool, it needs to be converted into toolpaths for the 3D printer. The process of converting 3D models into toolpaths for the 3D printer is called slicing. This is because it begins by slicing the model into horizontal layers. In this thesis, Insight[2], made by Stratasys, is used for this purpose. There are many different free slicers available on the market, e.g. Cura[3] and Slic3r[4], but Insight is developed by the same company as the 3D printer used in this thesis, and it also includes a software called Control Center, which manage and monitor the print jobs on the 3D printer.

In Insight, it is also possible to add support structures and select the density of the model to be printed. Support structures could be added automatically and manually edited. This makes it easy to control where the support structures will be added, and avoid it where it is not desirable. Support is important for the printer to be able to print structures where the model does not support all areas by it self. The printer is able to print small holes and overhang up to a reasonable limit without support structures. If this limit is exceeded, it will affect the quality and the precision of the model. The density of the model is a measure of the infill. A solid structure will take longer time to print than a model with less density. If less density is chosen to make the print go faster, it will allow a more hollow infill. This will affect the strength of the model, so the need for a strong model should be considered.

3.2.2 Control Center

Control Center is the software used to control the jobs of the 3D printer in this thesis. After the model is prepared in Insight by adding support, choosing the density and generating the toolpath, the job is built and opened in Control Center. Here, the model is placed on a chosen location on the tray. To be able to reuse the tray, it is important to avoid the measurement points of the printer. It is also important to avoid spots where other models have been printed, since the remains from a previous print could effect the quality of the new print. Control Center also monitor the build queue, and show how much time is remaining on the current print.

[2] http://www.smg3d.co.uk/3d_design_software/insight_software
[3] http://ultimaker.com/en/products/cura-software
[4] http://http://slic3r.org/

This is useful to be able to share the printer with others, and to increase the efficiency.

3.2.3 Fortus 250mc

The 3D printer used to print the physical robots in this thesis, is the Fortus 250mc. The Fortus 250mc is produced by Stratasys, and use the material ABSplus-P430[5]. ABSplus is a production-grade thermoplastic material, which is mechanically strong and stable over time. It is available in a wide spectre of colors, and it works with support material that is possible to remove with liquid. More specifications about the material could be found in the datasheet[6].

The Fortus 250mc features a 254 x 254 x 305 mm build area, and offers three different layer thicknesses: 0.178, 0.254 and 0.330 mm. The layer thickness could be chosen in Insight, and a layer thickness of 0.254 mm is used in this thesis. The Fortus 250mc builds the 3D models with the help of a well known technology called Fused Deposition Modeling (FDM). FDM technology means to build layer by layer by heating and extruding the thermoplastic. The thermoplastic is heated to a semi-liquid state before it is used to build the 3D model. One of the benefits with FDM technology is that it manages to build more complex models than some of the other technologies available on the market. The Fortus 250mc used in this thesis is shown in figure 3.1

3.3 Electronics and Mechanics

To be able to measure the performance of the robots, they have to move. To create motion, off-the-shelf motors and servos are used. These are used as moveable links between the parts. Some to create basic movement, and some in the search for better performance.

3.3.1 Dynamixel Servo AX-18A

To make robots move, inspiration is often taken from animals and the human body. The human body have many different joints that does some kind of rotation, e.g. the elbow and the shoulder. For basic forward movement of a robot, inspiration from animal and human legs can be of great use. To imitate joints like hips and knees to create forward movement, this thesis use the AX-18A servo from Robotis. The servo is shown in figure 3.2.

To program and control the servos, libraries for different pieces of software are available e.g. for Labview[7] and for Matlab[8]. In this thesis,

[5] http://www.stratasys.com/materials/fdm/absplus

[6] http://usglobalimages.stratasys.com/Main/Files/Material_Spec_Sheets/MSS_FDM_ABSplusP430.pdf

[7] http://www.ni.com/labview/

[8] http://www.mathworks.com/products/matlab.html

Figure 3.1: The Fortus 250mc

Processing[9], which will be explained in detail later, is used for this matter. It is possible to program the servos in two different modes; actuator mode and wheel mode. The latter means that it will rotate 360 degrees and keep rotating as long as the program runs, while the former means that it rotates to a given position within 0-300 degrees. There is 1024 positions available with values from 0 to 1023, which gives a resolution of 0.29 degrees between each position. It is also possible to program the speed of the rotation. The speed has the same number of steps as the position, and the speed can reach up to 97 rounds per minute (rpm) with no load. More detailed specifications could be found in the datasheet[10].

3.3.2 Stepper Motor

Since this thesis tries to find out if morphology changes, especially the length of the legs, could effect the performance of a robot, a way to implement a prismatic joint is needed. For this matter, a stepper motor from Mercury Motor is used together with a ball screw and a slider. The complete prismatic joints are shown in later chapters, together with the

[9] http://processing.org/

[10] http://support.robotis.com/en/product/actuator/dynamixel/ax_series/ax-18f.htm

Figure 3.2: The AX-18A servo from Robotis

complete robots. This stepper motor is a bipolar motor , which means that it has two poles in compliment to its stationary field. Ballico et al. [39] and Lee's patent [40] provides more detailed information about bipolar motors. It has a step size of 1.8 degrees, which gives a total number of 200 steps for a complete circle of 360 degrees. The stepper motor is shown in figure 3.3.

Figure 3.3: The stepper motor from Mercury Motors

The stepper motor could be controlled by an Arduino board with a connected motor driver. The the Arduino board is programmed in the Arduino software and all of these components will be explained in detail in the following sections. One might think that an Arduino would be able to control the stepper motor on its own, but there are several reasons why it can't:

- The Arduino provides to little current.
- The Arduino does not provide enough voltage.

- The electronics in the motor needs to be protected against reverse voltage from the inductor.
- If the stepper is charged with a heavy load, it could create big peaks in the voltage when it is turned on and of, which could end up with turning of the micro controller.

To cope with these problems, a motor driver is used between the Arduino and the stepper motor.

EasyDriver

The motor driver selected to use between the Arduino and the stepper motor in this thesis, is a stepper motor driver from Allegro MicroSystems called EasyDriver. As the name suggests, it is easy to use, and it is compatible with anything that could send a digital 0 to 5V pulse. It is possible to change this requirement to a digital 0 to 3.3V pulse with the help of a regulator on the board. When the driver is connected to a microcontroller that can provide the required pulse, it is used by sending a pulse to the STEP input on the driver. By the use of the built-in translator, this will make the motor take one step, where the step size is defined by two logical inputs. The possible step sizes are full, half, a quarter or an eight of a full step.

The EasyDriver is based on the A3967 chip from Allegro MicroSystems. This chip is a bi-polar driver, which means that it sends current both ways with the use of a H-bridge design. A H-bridge consists of 4 switches, and is the component that is used to control the high current and high voltage between the motor and the microcontroller. The states of the H-bridge are shown in figure 3.4.

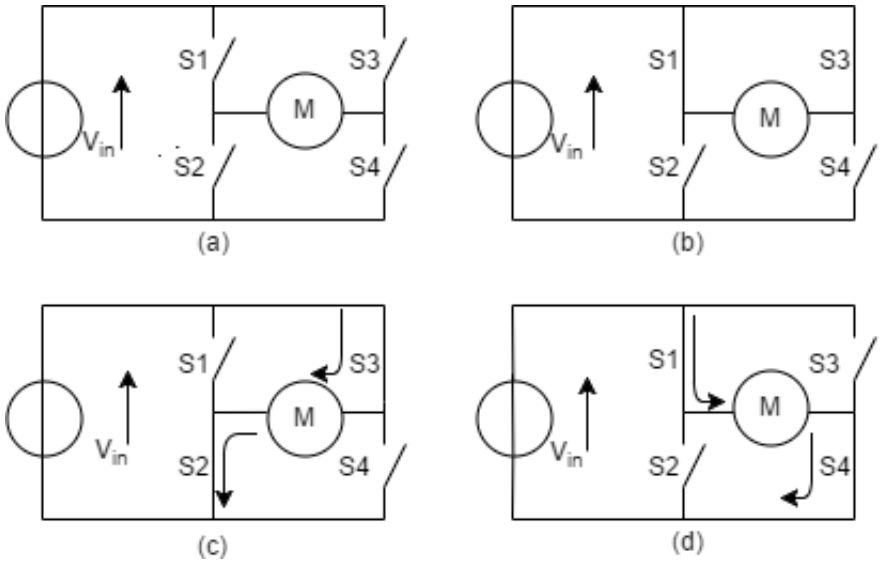

Figure 3.4: The states of a H-bridge

The EasyDriver is shown in figure 3.5. More information about the A3967 chip, which the EasyDriver is based on, can be found in the datasheet[11].

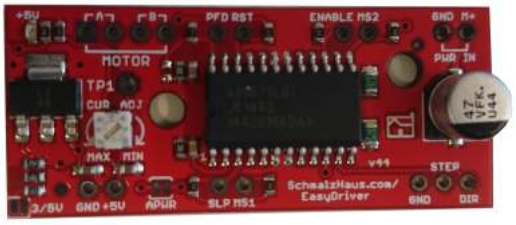

Figure 3.5: The EasyDriver from Allegro MicroSystems

3.3.3 Arduino Board

An Arduino board is a microcontroller with the possibility to attach different shields to provide additional functionality. A microcontroller is in short a small computer that consists of a processor together with memory and programmable input/output pins. A microcontroller could also include many other things, and e.g. Gridling and Weiss [41] and Bannatyne and Viot [42] offers some more detailed information. There are many Arduino boards available on the market, e.g. Arduino USB, Arduino Serial and Arduino Bluetooth. As the names suggest, the USB version communicates through USB, the serial version through serial ports and the bluetooth version supports wireless serial communication over bluetooth. The name of the board does not always point to what type of communication the board use, but could point to e.g. size of the board as with Arduino Mini and Nano, and to when the board is released as with the Arduino Duemilanove, where duemilanove means 2009 in Italian. What differs the different boards is not only the type of communication, but also the microncontroller the board is based on. This includes the processor and the number of analog and digital in and output pins.

With the help of a printed circuit board (PCB), it is possible to add a shield that provide functionality like Ethernet, Wifi, displays and motor drivers like the one that is used in this thesis. The board used in this thesis, is an Arduino Duemilanove. It is based on the ATmega328, and it has 14 digital input/output pins and 6 analog input pins. It can be connected to a computer with USB, and this will also give enough power for the board to run. Alternatively, a power adaptor could be used. The Arduino Duemlianove is shown in figure 3.6 .More detailed information about the board could be found in the datasheet[12].

[11] http://cdn.sparkfun.com/datasheets/Robotics/A3967-Datasheet.pdf

[12] http://www.atmel.com/images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168P
datasheet_Complete.pdf

Figure 3.6: The Arduino Duemilanove board

3.4 Programming

As previously stated, the servos and the stepper motors are the mechanics that are used to create movement in the experiments of this thesis. To make the servos and stepper motors behave as wanted, they need to be programmed. This section will introduce the pieces of software used to program the servos and stepper motors to make the robots move in the experiments of this thesis.

3.4.1 Processing

Processing, made by Casey Reas and Ben Fry, is an open-source integrated development environment (IDE) that is able to create images, animations and interactions. The idea was that one line of code should make something graphical happend. The creators call this kind of coding for sketching with code [43]. This is the reason that each computer program in processing is called a sketch and Processing is called a sketchbook. This implies that Processing is made for the principle of testing and failing, because it should be fairly fast to write one line of code and see what happens. The software is well documented on the website[13], and there are several instruction books [43, 44, 45] available.

Processing is not only a software, but it is also its own programming language. Processing is a dialect of the well known programming language Java. The syntax is close to identical, but Processing also includes features from other languages related to graphics and interaction [43].. The graphic elements are related to OpenGL and PostScript, where the latter is a language used to describe how a page should look, while the former is an application programming interface (API) for 2D and 3D vector rendering.

Processing consists of many different tools that work together. It is extended by several libraries that gives access to these tools and makes it possible to work within many different areas e.g. sound, data and network. This thesis will take advantage of a Processing library for serial communication. The serial library reads and writes data to and from

[13] http://processing.org/

external devices. It is done one byte at the time, and allows communication between e.g. computers and microcontrollers. The serial library is used to program the servos used to make the robot walk. How the servos could be programmed is explained in section 3.3.1.

3.4.2 Arduino IDE

The Arduino[14] IDE is an open-source IDE used to program the arduino boards introduced in section 3.3.3. Arduino is based on the Processing IDE, and use the processing programming language explained in section 3.4.1. Thus, a program made in Arduino is also called a sketch. The IDE is compatible with multiple platforms; it runs on Windows, Linux and Mac, which makes it widely used.

In this thesis, the Arduino IDE is used to program the Arduino board with an EasyDriver connected, which controls the stepper motors used for prismatic joints. This programming consists of sending the correct output to the correct pins of the EasyDriver, to make the stepper motor move as wanted. Stepping state, size of the steps and direction are examples of features that could be controlled from the Arduino IDE. The programming possibilities of the stepper motor and the EasyDriver is explained in more detail in section 3.3.2.

The Arduino philosophy is based on the same trying and failing principal as Processing. The goal is to find faster and more powerful ways to create better prototypes [46]. The possibility to create prototypes, and to try and fail without destroying anything, are some of reasons why Arduino is so popular both for hobby users and for education. This thesis also use the trying and failing principle for the robot design, and also when it comes to programming the Arduino to make the stepper motors move in a desirable way.

[14] http://www.arduino.cc/

Chapter 4

Implementation and experimental setup

This chapter will describe the implementation that was done to perform the experiments in this thesis. It will also introduce the experimental setup that was used to test the hypothesis. Since the hypothesis was that adjustable morphology as a parameter in optimization will lead to increased performance compared to optimization on a static morphology designed without a priori knowledge about morphology design, the experimental setup had to be able to test the performance. The performance in this thesis was measured in forward walking speed. Thus, the experimental setup had to be able to measure the speed in a reasonable way.

The description of the implementation will consist of the process of designing the different robots used to test the hypothesis. This includes choices made in the 3D modeling, use of simulation, developing gaits and selection of electronic equipment. The simulation was used in the process of developing gaits and testing if the robots were able to move. Instead of trying to create an advanced simulation setup to test the hypothesis in simulation, the focus was to test the hypothesis in the real world.

4.1 Experimental setup

To be able to test the hypothesis, the experimental setup needed to be able to test the performance of the robots. The performance measure is chosen to be forward walking speed. Thus, an environment where the setup could measure the forward walking speed was needed.

In previous experiments at the ROBIN research group, e.g. Samuelsen and Glette [28], a way to measure forward walking speed has been a subject. In the mentioned paper a motion tracking system is used. One of the challenges with this setup, is that the robot often walks out of bounds. Samuelsen and Glette solved this by making the robot turn when it reached the end of the tracking area. To avoid this challenge, this thesis used a different setup to measure the performance.

This thesis had the purpose of testing forward walking speed. Thus, if the experimental setup allowed it, forward movement was the only thing that the robots needed to be able to do. This lead to the experimental setup that was used in this thesis. The setup consisted of a rotating base with a connected rod. The rod was also able to move up and down, which allowed the robots to have different heights. The setup is shown in figure 4.1.

Figure 4.1: Experimental setup

The experimental setup also included a breadboard with an Arduino to control the stepper motors, an easydriver for each stepper motor, an USB to Dynamixel adapter to connect to the servos, power supplies and wires to connect everything together. The standard wires for the stepper motors and the servos were not long enough for the robot to move as desirable. Thus, the wires were extended to a desirable length with the help of longer wires and soldering. A laptop was used to run the Processing and Arduino sketches that controlled the movement of the robot.

The relation between speed, distance and time is as known $v = s/t$, where v is the speed, s is the distance and t is the time. Thus, the speed

could be measured by measuring the distance a robot walks in a given time. This was used to measure the performance in this thesis. The angle between the starting point and the stop point, measured by looking at the position of the rod, was used as distance measure. Thus, when a robot walked a given number of walking cycles, the angle showed if the robot was faster or slower based on the distance walked over the time it took to walk 3 cycles. The angle was measured manually with a protractor, and an example of a performance measurement is shown in figure 4.2

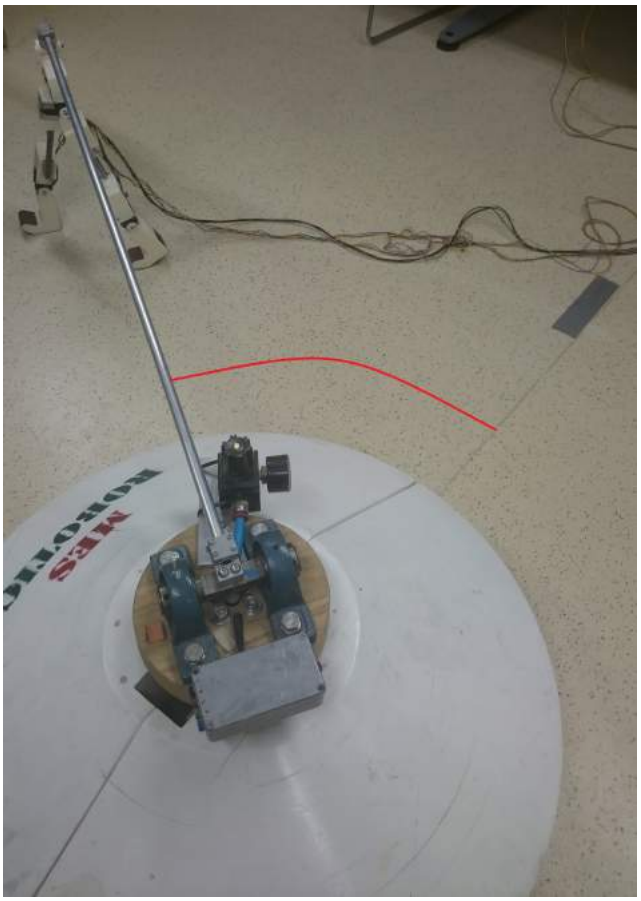

Figure 4.2: Example of performance measurement, where the red arc represents the angle between the starting line and the end position

4.2 Mechanics

Making the robots move was an important part of the implementation ahead of the experiments. To select the correct mechanics to create movement, some basic criteria was set. The most important criteria are mentioned bellow:

- Weight - It was important that the robot was able to move with the different motors attached, and that it did not break because of overweight.

- Power - The motors and servos had to be able to move the parts it were suppose to move. For the prismatic joints, it was important that the mechanics were strong enough to create movement, dispite forces from all directions.
- Controllability - To be able to test the hypothesis as much as wanted, it was important that it was fairly easy to program and control the mechanics.
- Accuracy - To get a fair comparison of the results, it was important that the mechanics were able to repeat the movement with a reasonable accuracy.

4.3 Version 0

When this thesis was started, a designed 3D model of an expected not-working robot was handed out. This robot was named Version 0, because, as later explained, it was not printed out and tested. This version consists of four legs and an adjustable body. The body had a prismatic joint, which makes it possible to adjust the distance between the legs on each side. This version was also designed with the possibility to turn. The 3D model of Version 0 is shown in figure 4.3.

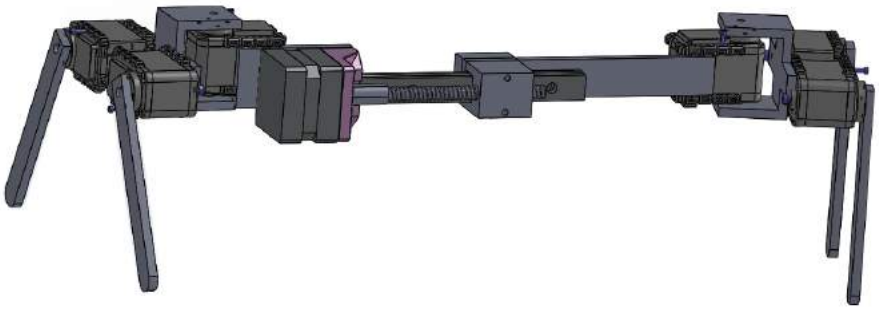

Figure 4.3: The 3D model of Version 0

To be able to test the hypothesis on the robot, a way to measure the performance was needed. The performance measure in this thesis was the forward walking speed, thus, the robot needed to be able to walk forward. To get as precise measurements as possible, it was desirable that the robot walked as straight as possible. This was the base for the decision of removing the robot's ability to turn. This, together with some other design improvements, lead to Version 1. The remaining details about the parts and functionality of Version 0, is explained in the next section, which introduces Version 1.

4.4 Version 1

Version 1 was the first robot that was printed and tested in the real world. It was based on the handed out Version 0, but some modifications were made because of undesirable features, together with some practical design adjustments.

4.4.1 Modeling

The modeling of Version 1 consisted of doing the desirable modifications on Version 0, and making all the parts ready for printing. The first modification that was made, was to remove the robot's ability to turn. The ability to turn was modeled with the help of a servo, and the servo had the function as the connection between the legs on the side and the body. The turning servos was replaced by a static connection on each side. These connections had to fit the servos that were used to move the legs.

This kind of modeling was done in Solidworks. It was done by inserting a 3D model of the servo into a new part and creating the model for the connections by drawing on top of the servo. These two connections, together with a stepper motor, a ball screw, a rail, a slider and some additional parts to make everything fit together, amounted the prismatic joint that could adjust the length of the body. The two connections had to be modeled differently on the side that was a part of the prismatic joint. One of the connections had to have a mount for the stepper motor, while the length of them needed to be calculated to make room for the sliding parts.

The prismatic joint was made with the stepper motor as the part that creates movement. A mount for the rail with room for the ball screw was modeled to fit on top of the stepper motor. The rail, which was made of steel, was glued to this mount, while the ball screw was glued to a modeled connector on the tip of the motor. The other side of the joint consisted of a slider with a screw hole for the ball screw, which slid on the rail and moved when the stepper motor rotated the ball screw. This slider had a connector for the rotating rod and was connected to the connection for the legs on that side. The 3D model of Version 1 is shown in figure 4.4, trying to walk towards the camera during simulation.

4.4.2 Simulation

Version 1 was simulated with the help of the motion analysis tool in Solidworks. The simulation was used to test static gaits, and see if the robot was able to move forward. The simulation was done by placing a rotary motor on all four servos to make the legs move. The prismatic joint was held static in the beginning of the simulation, to just test if the robot was able to move forward.

Gravity was added to create an environment that was as real as possible. It was also added contact between the legs and the floor, where the friction was set to dry rubber. This type of friction was used to allow

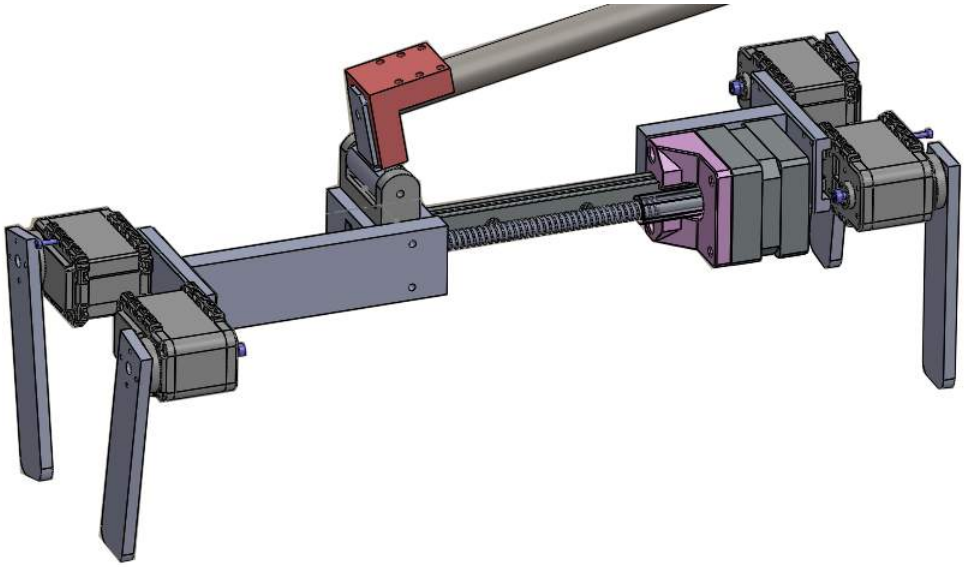

Figure 4.4: Version 1 during simulation in Solidworks

as little sliding on the legs as possible. The rod that is used to support the robot in the physical tests, is also assembled in Solidworks and used in the simulation. A small part of the rod and the connection between the robot and the rod during simulation can be seen in Figure 4.4.

The simulation showed that this robot would not be able to create markable forward movement with a static gait. This is because it only has the possibility to move the legs either forward or backwards, and not up and down. This creates a problem with friction. Somewhere in the walking cycle the robot will try to move a leg that is placed on the ground and has a lot of weight on it. This will make the robot move in the opposite direction of what is intended. The simulation showed that this creates a pattern of three small steps forward and one huge step backwards. Even though the simulation showed that the robot will have a hard time creating forward movement without the help of a dynamic gait, the robot was printed out and tested in the real world to test that the simulation is correct. This was also done to do a discussion about static versus dynamic gaits, and to see if real world forces could lead to something the simulation can't reproduce.

4.4.3 Creating movement

To make the robot move, the servos needed to be programmed. Since the simulation showed that a static gait most likely will not be able to make the robot move forward, the choice of gait fell on a common static gait, the creep gait. The gait was implemented in Processing, with the timing diagram shown in figure 4.5 used as inspiration.

This gait was, as previously mentioned, used to show that this robot is not able to walk forward with a static gait. It may be possible to create

forward movement by using dynamic elements like jumping, but it was not desirable to use that kind of dynamic elements in this thesis.

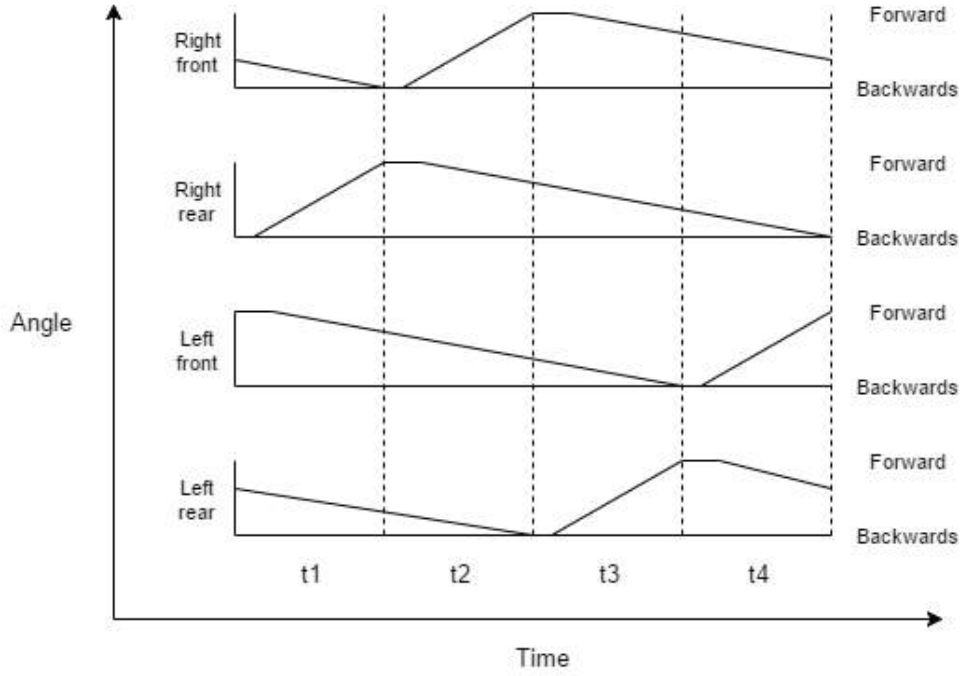

Figure 4.5: Timing diagram of the creep gait used on Version 1

4.5 Version 2

Version 2 was the second version that was printed and tested in the real world. It had a completely different morphology from Version 1, but some of the design choices were based on the problems of Version 1. These choices lead to a wanted progression from Version 1 to Version 2. Version 2 was the first, and in the end, the only robot that tried to test the hypothesis, and was therefore an important part of the thesis.

4.5.1 Modeling

The modeling of Version 2 consisted of creating a reasonable model with the intention to test the hypothesis. Thus, the model needed to be able to move forward in greater manner than Version 1. The model was created from scratch in Solidworks. A human walking mechanism was used as inspiration for the model, and later on the gait. Since the rod in the experimental setup gave some support to the robot, it was possible to create and test a human-like model only consisting of the mechanism of the legs. Thus, the model had the contours of a human body from the hips and down.

The first feature that Version 2 needed to have, was the ability to lift the legs and not only being able to move them forward and backwards. This

lead to an extra rotational joint on each leg. These rotational joints were modeled with a servo as in Version 1. The extra joints were placed above the already exiting joints to work as the hips of the robot. These two joints were then connected to a top to keep everything together. This top also had a connector to connect the robot to the rod.

When the robot was able to lift the legs and move them forward and backwards, the most essential feature needed to be implemented. This was the ability to adjust the length of the legs to test the hypothesis. This ability was implemented with a prismatic joint between the two rotational joints on each leg. The prismatic joints were modeled in the same way as the prismatic joint in Version 1, but with a thinner rail to reduce the weight. The weight of the prismatic joint was a possible showstopper, with especially the stepper motor and the rail adding weight. Too much weight could have made the whole robot break, or, as in this case, the weight reduce the ability to move. To make the prismatic joint fit in Version 2, a connection between the prismatic joint and the lower rotational joint was made. This connection had a mount for the servo and screw holes to connect the slider and for the ball screw that was used to adjust the length of the prismatic joint.

The legs were modeled to fit the rotating part of the servo in lower rotational joint. The shape of the foot was modeled to look like a boot. This features a reasonable big flat area underneath to create enough friction to be able to walk. It was important that the tip of the foot was not too long, since that could have made it impossible to move the leg forward and backwards, even if the leg was lifted from the ground. The 3D model of Version 2 is shown in figure 4.6.

4.5.2 Simulation

Version 2 was simulated with the same tools in Solidworks as Version 1. This included the use of the motion analysis tool, adding rotary motors on all the servos, adding gravity and contact with friction and the prismatic joints were also held static in the simulation of Version 2. The simulation was used to test if the robot was able to walk forward, and to test and find a reasonable gait. The assembly that was simulated also included the rotating rod that was connected to the robot. Figure 4.7 show Version 2 in action during simulation.

The simulation addressed important things to make this robot walk. The most important one was that the feet needed to be lifted high enough to be able to swing the foot forward and backwards without touching the ground. If the foot touched the ground during the swing, the robot would move in the opposite direction of the what was intended.

In contrast of the simulation of Version 1, the simulation of Version 2 showed that this robot was able to create markable forward movement. Thus, the simulation of Version 2 showed that this robot would bring the thesis one step closer to test the hypothesis. As previously mentioned, the friction in the simulation and in the real world may differ significantly, so this was one of the questions that only the printed robot could answer. Another question that the simulation left unanswered, was if the robot was

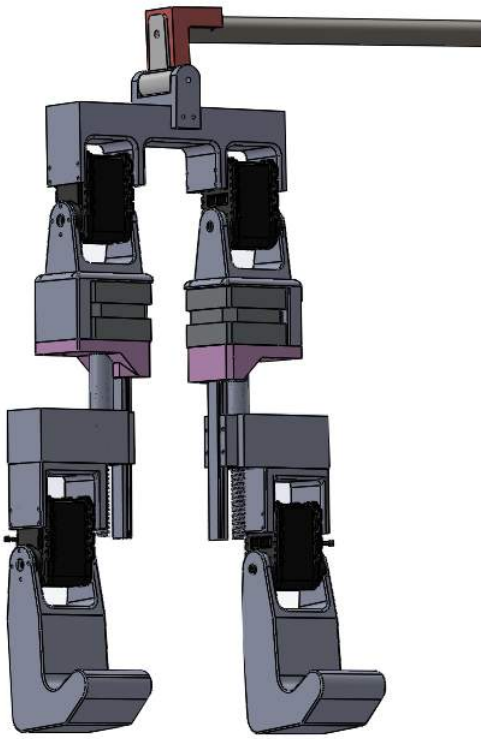

Figure 4.6: 3D model of Version 2

able to carry the weight of the prismatic joint with the stepper motor and the rail.

4.5.3 Creating movement

To make the robot walk, the servos needed to be programmed. The upper servos needed to be programmed to swing the legs forward and backwards, while the lower servos needed to be programmed to lift the feet to a position where they don't crash during the swing, together with placing the feet flat on the ground again to achieve enough friction to move the other leg. A human-like gait was used as inspiration for the gait of the robot, and the timing from the simulation that the gait was based is shown in figure 4.8.

The gait was developed by trying and failing different timings during the simulation in Solidworks. To transfer the timings from the simulation to the real robot proved to be difficult. The effect of weight, gravity, friction and other physical factors lead to changes in the timing of the gait. Another bump in the road, was the difference between moving the rotary motors in the simulation, and sending commands to the servos. In the simulation it is possible to tell a motor to go from angle a to angle b in a given time, while the servos gets a command of moving to a given position in a given speed. This makes it hard for the servos to perform the same

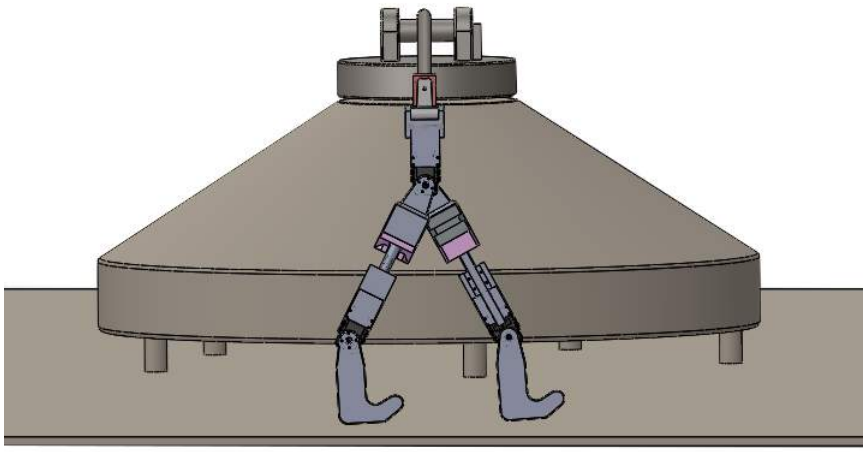

Figure 4.7: Version 2 during simulation

timing as the motors in the simulation. The most important features were markable forward movement and being able to repeat the gait as good as possible. As mentioned for Version 1, dynamic elements like jumping were not desirable, and the developed gait was based on a goal of as much static stability as possible.

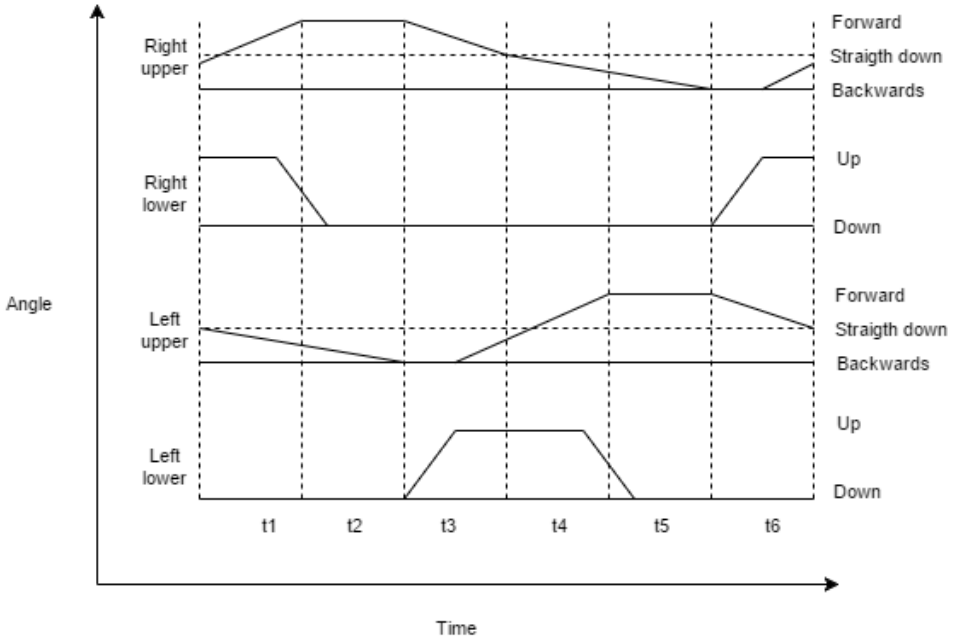

Figure 4.8: Timing diagram of the human-like gait used on Version 2

To test the hypothesis by doing experiments on adjusting the length of the legs, the stepper motors, which controls the prismatic joints, needed to be programmed. They needed to be programmed to run a reasonable number of steps to adjust the length with small steps. After some tuning,

the stepper motors were programmed to run 1000 steps as one step for the prismatic joint. This step size for the prismatic joint was measured to about 1.5 mm, which gave a total of 33 steps. To make the stepper motors move, the Arduino needed to send digital one and digital zero to do one step. One step is followed by a delay that is needed for the Easydriver to make the motors work. This delay is called the stepping period, and this thesis used a stepping period of 150 ms.

4.6 Version 3

Version 3 was the third and last version that was printed and tested. It was based on the walking mechanism from Version 2, but with some design adjustments. The progress from Version 2 to Version 3 included reaching for static stability. This, and the rest of the design, will be explained in detail in the following sections, together with the use of simulation and making the robot walk to perform the experiments.

4.6.1 Modeling

The modeling of Version 3 consisted of improving the design from Version 2 to get closer to the goal; concluding the hypothesis. Version 2 was based on a human-like designed, which made it impossible to achieve total static stability. As previously stated, static stability was desirable to get as exact results as possible. Since static stability means that the robot should be balances while moving, it was natural to think in the direction of additional legs to achieve better balance.

The modeling was done in Solidworks, and since the adjustable legs on Version 2 worked in a desirable way when tested during the gait design process, these legs were used as a starting point. Thus, the same prismatic joint was used to adjust the length of the legs in Version 3, as in Version 2. To achieve the mentioned static stability by adding more legs, how many legs to add needed to be figured out. With inspiration from the gaits mentioned in section 2.6, the choice fell on 4 legs. It was also important to keep the robot as lightweight as possible for the servos to be able to move the robot as desirable.

Based on the stated weight challenge, the design of the part that should hold the robot together needed to take the weight into account. Since this part's only job was to hold the robot together, it was made as simple as possible. It was modeled as a plate to be placed on top of the legs, with one leg in each corner. This plate needed to be long enough for the legs to be able to move without crashing. This lead to the plate being so long that it needed to be divided into two parts to be able to print it in the 3D printer. One side was designed with a crack, while the other was designed with a flap to fit in the crack so they could be glued together. To keep the weight down, the part was printed with low density, which means that the part was nearly hollow inside. It was also designed as thin as possible, but it needed a reasonable thickness to be able to hold the weight and the forces

during movement. The chosen thickness was 15mm.

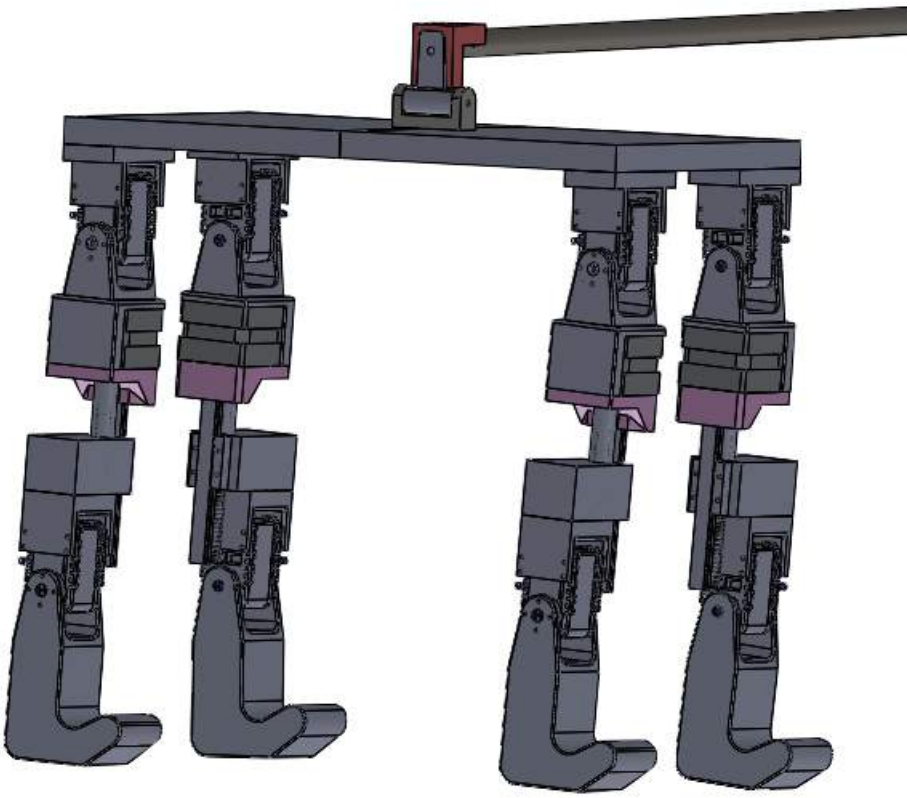

Figure 4.9: 3D model of Version 3

To be able to measure the performance with the help of the experimental setup, the robot needed to have a mount for the connection to the rod. This mount was placed in the center of the plate holding the robot together. It was not possible to print the mount together with the plate, since it was placed on top of the connection between the two parts forming the plate. Thus, the mount was printed as an own part and glued to the plate. To fasten the upper servos on the legs to the plate, a new mount was designed. This mount was originally designed with screw holes to fasten them to the plate, but the mounts were glued to the top to avoid making small screw holes in the plate. This was based on the fact that the plate was printed with low density, and small holes could have a critical effect on the strength of the part. The designed model of Version 3 is shown in figure 4.9.

4.6.2 Simulation

Version 3 was simulated with the same motion analysis tool in Solidworks as previously mentioned for the earlier versions. Rotary motors were added to simulate the rotation on each of the servos, gravity was added, contact with dry rubber friction was used and the length of the legs

were held static during the whole simulation. The robot was connected to a model of the experimental setup, including the rotating hub with a connected rod.

The simulation was used to test if the robot was able to walk, and to find a gait that could help test the hypothesis. The same gait as the gait tested on Version 1, a creep gait, was tested. The biggest difference from Version 1, was Version 3's ability to lift the legs. With this ability, the challenges that Version 1 experienced were close to non existing.

The simulation showed that this gait would be able to make the robot walk. The simulation also showed that the robot should be very close to achieving total static stability with the use of this gait. However, the previous versions showed that the physical aspects, like balance, could have a markable reality gap. It is hard to simulate the correct weight of the models, and thus, gravity's effect on the robot, which is one of the main reasons for this reality gap. The simulation showed that if the reality gap was not to significant, it would lead the thesis closer to a conclusion of the hypothesis. Figure 4.10 show Version 3 in motion during simulation.

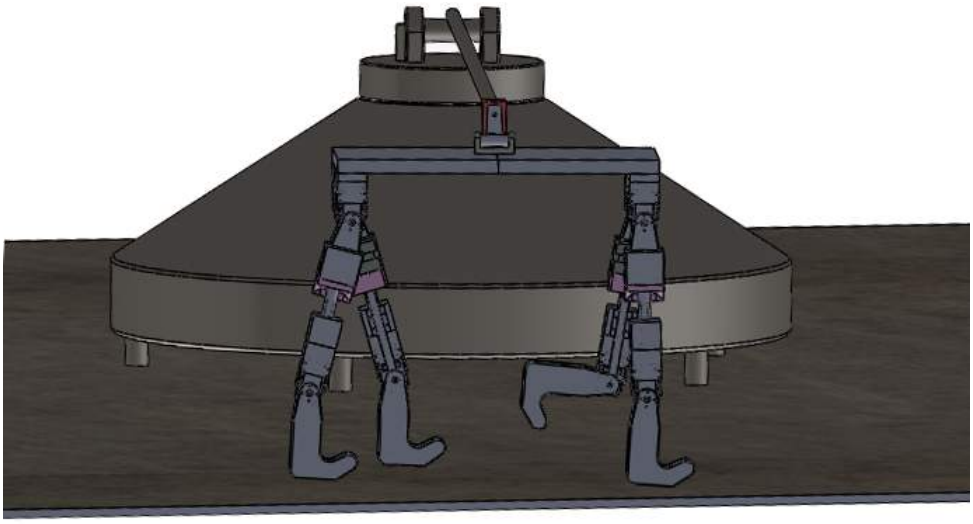

Figure 4.10: Version 3 during simulation

4.6.3 Creating movement

As for the other robots, the servos needed to be programmed to make the robot walk. It was the same situation for the different servos as in Version 2, the upper servos were programmed to swing the legs forward and backwards, while the lower servos were programmed to lift feet to a position where it did not crash during the swing. The lower servos were also programmed to make sure that the flat part of the foot was placed on the ground to achieve as much friction as possible. As stated in the section about the simulation, the same creep gait as used on Version 1 was used as inspiration. The timing is the same as in figure 4.5, but in addition, the legs

are lifted in the following order; right rear in t1, right front in t2, left rear in t3 and left front in t4.

As previously stated for Version 2, the reality gap from simulation to the real world regarding timing in gait design was a big challenge. With 4 legs that should be timed and interact together, compared to 2 legs on Version 2, the reality gap was an even bigger challenge. To make the feet interact as in the simulation proved to time consuming, which lead to the design of a new gait. This gait took inspiration from the gait used on Version 2, but with the diagonal legs working together in the same timing. With some modifications, this is also known as a trot gait. In a normal trot gait, there is a time in each cycle where all 4 legs are lifted from the ground. This was not implemented in the gait for Version 3, since this thesis tried to avoid dynamic movement.

Chapter 5

Experiments and results

This chapter describes the experiments done to test the hypothesis, show the results conducted from the experiments and compare them to the hypothesis. The experiments consists of testing different 3D printed robots with the ability to adjust the morphology, with the main focus on adjustable legs. The ability to adjust the morphology was used as an parameter in optimization of robot gaits. To test the hypothesis, optimization of a gait with adjustable morphology is tested and compared to optimization with a fixed morphology.

The results from the different robots with different algorithms is explained and shown in tables, while the comparison is done by explanation and curves that visualize the difference between the results.

5.1 Experiments

5.1.1 Fitness

Both EAs and local search have the goal of finding as good fitness as possible. Thus, fitness is a measure of how good a solution is. The hypothesis state that the use of adjustable morphology will lead to increased performance. Since these experiments are done to test the hypothesis, they need a common fitness that is a measure of how good a solution is.

As previously stated, the performance measure of this thesis is forward walking speed. Since fitness and performance are two sides of the same case, forward walking speed was a natural selection when it comes to fitness. Since the experiments is based on optimization, the goal of each algorithm is to find the best possible fitness, and thus, the best possible forward walking speed. The methods for measuring the performance, and thus the fitness, is explained in section 4.1, but the highlight is that the fitness is measured by the distance the robot is able to walk during 3 walking cycles, where the distance is measure by the angle between the starting line and the end position.

5.1.2 Version 1

The experiment on Version 1 consisted of checking if the robot was able to create forward movement with the use of a static gait. Since the simulation showed that this most likely would not be possible, it was natural to not make any algorithms before this was tested. Version 1 was printed and assembled ahead of the experiment, and the assembled model is shown in figure 5.1.

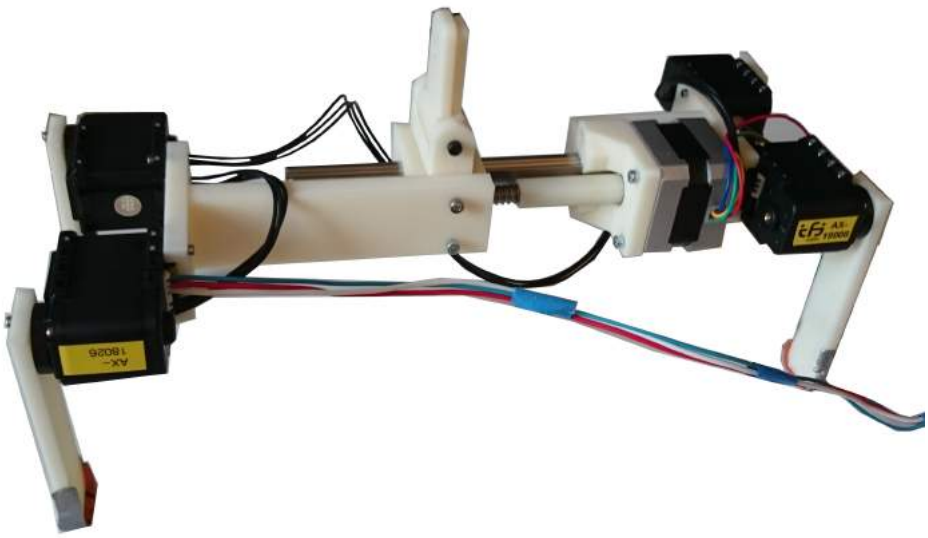

Figure 5.1: Printed and assembled Version 1

Version 1 was tested with the previously introduced creep gait. The timing diagram for this gait was shown in figure 4.5. To be able to perform this gait, and any static gait for that matter, the robot needs to be able to lift the legs, as well as move them forward and backwards. This robot did not have that ability, but the gait was tested to prove that it was not able to create markable forward movement.

To properly test if the robot was able to move forward, sufficient friction was needed. By trying to make the robot walk with the printed plastic legs against the floor, a lack of friction was noticed. The legs would slide and not be able to perform the given gait. Friction was achieved by taping a part of sandpaper on each leg, which made sure that the robot did not slide undesirably.

Since this robot had 4 legs and was able to achieve static stability by having 3 legs on the ground while the fourth move, it did not need to be connected to the rod in the experimental setup. This rod was, as previously stated, used to be able to get precise measurements of the forward walking speed, as well as applying support if the robot was a bit unstable. Since this experiment just consisted of testing if the robot was able to create markable forward movement, no exact measurements was needed.

The robot was tested by placing it on the floor and running the gait 5 times. It was run multiple times to be sure that the result was representable

and not only a random outcome. This experiment showed that the robot was not able to create markable forward movement with the use of this static gait. The robot was able to move two or three steps forward, but the next step then moved the robot a great distance backwards. This was because the robot did not have the ability to lift the legs, so the swing of the leg that was supposed to happen in the air, happened with the leg on the ground, and with a lot of weight on it. This was the same result as in the simulation. Thus, the experiment confirmed the suspicions from the simulation. This result was used to develop a robot that was more fit to test the hypothesis, Version 2, which took advantage of the awareness of the problems discovered during the experiments on Version 1.

5.1.3 Version 2

The experiments on Version 2 were the first experiments that tried to test the hypothesis. The experiments consisted of testing if the ability to adjust the length of the legs was a useful parameter for optimization of a robot gait. Optimization of a gait with a fixed morphology was also tested, to compare the results. As previously stated, the fitness was measured in forward walking speed.

There was first found a reasonable length of the legs to use as the fixed length during optimization. This was done by testing evenly distributed lengths from across the available lengths to cover the whole spectre. A local search was run in the neighbourhood of the best solution to see if better solutions were found. The best solution found was then used as the length of the legs in the fixed morphology during optimization of the gait with the use of a GA with the angle on the steps as parameter. These results were then compared to the results of optimization of the gait with the angle on the steps and the length of the legs as parameters. To make the search space smaller, and the gait easier to handle, the same adjustment of the legs was done equally on both legs. Thus, the legs always had the same length.

Both EAs and local have a stochastic nature. This means that they could get trapped on a local maximum. Thus, the starting point could affect the result in great manner. Because of this stochastic nature, some of the experiments were run twice to check if the results could be trusted. This gives a more representable and valid result, and gets this thesis closer to a conclusion. The number of runs, fitness functions and so on is described more in the sections about each of the experiments.

The gait used on Version 2 was developed with inspiration from the simulation, but as previously stated, the gait was considerably changed to make it work on the real robot. This gait was used for all the experiments on Version 2 to make the results comparable. For the results to be comparable, and the gait to work properly, sufficient friction was needed to avoid the legs to slide undesirable. Nearly the same method to achieve friction as the method used on Version 1 was used. The difference was that the sandpaper was glued onto the underside of the foot, compared to the use of tape on Version 1. The glue helped create a tighter fit and made sure there was no movement in the sandpaper, which could lead to the sandpaper falling of.

The assembled Version 2 is shown during one of the experiments in figure 5.2.

Figure 5.2: Version 2 during an experiment

5.1.4 Version 3

The experiments on Version 3 were suppose to consist of the same experiments as for Version 2. Thus, local search on the length, GA with the angle of the steps as parameter and GA with the angle of the steps and the length of the legs as parameters. Because of a number of challenges that will be discussed in later sections, the experiments on Version 3 consisted of only selecting evenly distributed lengths and running a local search in the neighbourhood of the best.

Version 3 used the same technique to achieve sufficient friction. Thus, sandpaper was glued on the underside of the feet. This worked reasonably well during the experiments, but the friction is not comparable to the friction in the simulation. The forces in the real world makes sure that if the robot gets out of balance, the feet will find a way to slide. The assembled Version 3 is shown in figure 5.3.

Figure 5.3: Version 3 during an experiment

5.2 Optimization with local search

To be able to test the hypothesis, an optimization with a fixed morphology needed to be tested. To make the comparison fair, some optimization was done on the length of the legs. This optimization consisted of picking five evenly distributed lengths and testing their fitness. This made sure that lengths from the whole spectre were tested. Then, a local search with the hill climbing algorithm was run in the neighbourhood of the length with the best fitness. These experiments and their results are presented next.

5.2.1 Hill climbing experiments

The hill climbing algorithm is a local search algorithm that start in a random place and search in the neighbourhood for better solutions. More information about local search and the hill climbing algorithm could be found in the background chapter.

In these experiments, the hill climber was initialized with the position of the prismatic joint with the best fitness from the tests of the positions evenly distributed like 0/4, 1/4, 2/4, 3/4 and 4/4. The hill climber was also run on the position with the second best fitness to see if this neighbourhood had a better solution, and to cover some more positions to avoid being caught in a local optima. To do reasonable evaluations and reduce the randomness, each position of the prismatic joint was run 2 times. This was also to see if the experiments had a high margin of error. The algorithm then tried to

adjust the length of the joint in one of the two possible directions, either longer or shorter. After 2 new runs, the mean of these runs was calculated and compared to the fitness from the previous position. This comparison was used by the algorithm to decide if the adjustment was able to increase the performance. If the adjustment was able to increase the performance, the prismatic joint was adjusted in the same direction, and the same fitness test was repeated. If the adjustment did not increase the performance, the algorithm tried to adjust the length in the other direction. This went on until the algorithm was not able to find a better solution, or if 10 iterations were reached. The algorithm is described in pseudocode in figure 5.4.

```
begin
    start = random start length;
    current = fitness(start);
    best = 0;
    iterations = 0;
    while iterations < 10 do
        neighbour = find next neighbour length;
        next = fitness(neighbour);
        if next > current then
            current = next;
        end
        if current > best then
            best = current;
        end
        iterations ++;
    end
end
```

Figure 5.4: Pseudocode of the hill climbing algorithm

5.2.2 Version 2 results

The results from the experiments of the evenly distributed selection of leg lengths are shown in table 5.1. The results show that the position 32, which was the longest length available, had the best fitness. It also show a steady increase of fitness between the different lengths. With these experiments, the results show that the robot had good repeatability, by being able to reproduce the fitness without a big variation. Based on these results, position 32 and 24 were selected as starting points for the hill climbing algorithm.

	0/4	1/4	2/4	3/4	4/4
Position	0	8	16	24	32
Fitness 1	55	61	63	69	75
Fitness 2	58	60	65	70	74
Mean fitness	56.5	60.5	64	69.5	74.5

Table 5.1: Fitness of evenly distributed leg lengths on Version 2

The results from the hill climbing algorithm with position 24 as starting point are shown in table 5.2. The results show a very even distribution of the fitness. The results show that position 28 had the best mean fitness over 2 runs. It also had the two best runs individually. The variance is still small, which makes the results more credible.

Position	23	24	25	26	27	28	29
Fitness 1	68	69	69	73	73	76	73
Fitness 2	66	69	72	73	73	77	73
Mean fitness	67	69	70.5	73	73	76.5	73

Table 5.2: Results of hill climber around position 24 on Version 2

The results from the hill climbing algorithm with position 32 as starting point are shown in table 5.3. These results show that the fitness is very even between several of the best positions. By comparing these results to the results from the hill climber with starting position 24, position 28 turned out to have the best mean fitness over two runs. Thus, position 28 was selected as the static length of the legs to use in the fixed morphology used in the one of the GAs. This, and the rest of the optimization with GA is explained in section 5.3.

Position	32	31	30
Fitness 1	75	77	74
Fitness 2	74	75	74
Mean fitness	74.5	76	74

Table 5.3: Results of hill climber around position 32 on Version 2

5.2.3 Version 3 results

The results from the experiments of the evenly distributed selection of leg lengths are shown in table 5.4. The results show that position 24 had the best fitness in the selection. Thus, position 24 was used as the starting point for the hill climbing algorithm in the next experiment. It is worth noticing that the fitness for Version 3 was markable lower than for Version 2. The fitness of the runs on Version 3 were close to half of the fitness from Version 2. This was because the gaits and models were different, but it also show the difference between using a static gait compared to a dynamic gait. The fitness between Version 2 and Version 3 will not be further compared because of this difference, but this was not the point either. To test the

hypothesis, the important part is the different versions fitness compared to themselves.

	0/4	1/4	2/4	3/4	4/4
Position	0	8	16	24	32
Fitness 1	27	30	31	32	31
Fitness 2	28	27	29	31	30
Mean fitness	27.5	28.5	30	31.5	30.5

Table 5.4: Fitness of evenly distributed leg lengths on Version 3

The results from the hill climbing algorithm with the starting position 24 are shown in table 5.5. The result show a drastic increase in variance. This was related to a series of problems that showed up during the experiments. The most essential problem was that the servos were not able to perform all the commands equally every time. One of the servos could suddenly turn it self of, or move with a fraction of the speed it was supposed to. This problem, together with the rest of the problems will be addressed in detail in later sections. The results show that position 23 had the best fitness, but it also had the biggest variance. Since the problems with the experiments lead to no further experiments, this value was not critical for any other experiments.

Position	22	23	24	25
Fitness 1	30	29	32	23
Fitness 2	26	36	31	14
Mean fitness	28	32.5	31.5	18.5

Table 5.5: Results of hill climber around position 24 on Version 3

5.3 Optimization with GA

5.3.1 Genetic algorithm experiments

The hypothesis claimed that using adjustable morphology as a parameter in the optimization of a gait, would result in increased performance compared to optimization with a static morphology. To test this, optimization with adjustable morphology and optimization with a static morphology needed to be tested. This thesis ran this optimization with the use of GA. The GA used different parameters trough different experiments, and each of these parameters will be described with the respective experiment.

The GA used here, was based on the simple GA. The simple GA use a binary representation, roulette wheel parent selection, 1-point crossover, bit-flip mutation, and the offspring always places the parents. The same parameters where used here, but the size of the population, the mutation rate and the crossover rate were adjusted for each experiment. Since the roulette wheel parent selection allows duplicates, duplicates were also allowed in the random initialization. The termination condition that was

used, was if the number of generations reached 25, or if the algorithm did not find a better solution for 5 generations in a row. The pseudocode for the GA is shown in figure 5.5.

```
begin
    population = initialize with random population;
    failed, best, previousBest = 0;
    while (generations < 25) and (failed < 5) do
        foreach individual in population do
            fitness = Fitness(position);
            fitnessList += fitness ;
            if fitness > best then
                previousBest = best;
                best = fitness;
                failed = 0;
            end
        end
        if best == previousBest then
            failed ++;
        end
        parents = RouletteWheel(population, fitnessList);
        population = Crossover(parents);
        population = Mutation(population);
        generations ++;
    end
end
```

Figure 5.5: Pseudocode of the GA

5.3.2 Optimization with angle

The optimization of a gait with static morphology was done with the angle of the steps as parameter. To encode the angle to fit a bit-string representation, each angle could be from 0 to 31 degrees, and each angle was represented by the angle as a binary number. The length of the legs were decided by the hill climbing experiment, and for Version 2 it was set to position 28. The legs were adjusted ahead of the experiments and kept with this static length trough out the optimization with the angle as parameter.

The first optimization that was run, was a GA with a population with 6 individuals. The individuals were selected by a random generator, where, as previously stated, duplicates were allowed. The crossover rate was set to 70%, which is in the middle of the recommended values shown in section 2.2.1, and the mutation rate was set to %, which is 1 gene per offspring.

The results of the first optimization with angle as a parameter are shown in table 5.6. The results show that the best fitness was found in the third generation and the algorithm was therefore terminated after 8 generations.

It also show that the fitness got worse after it passed the top. This could be a sign of to much exploration, since it moves away from the good solutions. This lead to a reduction of the crossover rate, together with an increased size of the population, to see how the algorithm works with a larger population, since 6 individuals is rather few.

Gen		Ind 1	Ind 2	Ind 3	Ind 4	Ind 5	Ind 6	Tot	Max
	Angle	25	2	18	6	28	7		
0	Gene	11001	00010	10010	00110	11100	00111		
	Fitness	78	0	34	0	86	0	198	86
	Angle	18	25	18	16	24	13		
1	Gene	10010	11001	10010	10000	11000	01101		
	Fitness	34	78	34	16	72	0	234	78
	Angle	30	24	26	18	25	16		
2	Gene	11110	11000	11010	10010	11001	10000		
	Fitness	90	72	81	34	78	16	371	90
	Angle	10	25	19	30	26	29		
3	Gene	01010	11001	10011	11110	11010	11101		
	Fitness	0	78	59	90	81	87	395	90
	Angle	10	25	14	19	22	30		
4	Gene	01010	11001	01110	10011	10110	11110		
	Fitness	0	78	0	59	71	90	298	90
		11111							
	Angle	29	17	22	3	23	25		
5	Gene	11101	10001	10110	00011	10111	11001		
	Fitness	87	36	71	0	71	78	343	87
	Angle	23	15	31	23	1	21		
6	Gene	10111	01111	11111	10111	00001	10101		
	Fitness	71	7	77	71	0	66	292	77
	Angle	11	18	19	19	15	18		
7	Gene	01011	10010	10101	10101	0111	10010		
	Fitness	0	34	59	59	7	34	193	59

Table 5.6: GA1 - mutation rate 17% and crossover rate 70%

There was then run a new optimization on the gait with the angle of the steps as parameter. The GA still used a mutation rate of 20%, but the crossover rate was changed to 60 %. It was initialized with 10 random individuals chosen by a random generator. The results are shown in table 5.7. The results show that the best fitness was found in the second generation, and that the algorithm was able to find the same best fitness for the next five generation. Thus, it seemed like the use of a lower crossover rate worked. Even if there was only 32 possible solutions available, finding

the best solution with only creating one new generation was very fast. This could mean that the algorithm was stuck at a local optima, or very lucky. Therefore, the same algorithm was run again, to see if it was able to reproduce the results.

Gen		Ind 1	Ind 2	Ind 3	Ind 4	Ind 5	Ind 6	Ind 7	Ind 8	Ind9	Ind 10	Tot	Max
	Angle	5	11	14	2	1	24	13	6	31	11		
0	Gene	00101	01011	01110	00010	00001	11000	01101	00110	11111	01011		
	Fitness	0	0	0	0	0	72	0	0	77	0	149	77
	Angle	30	27	20	31	15	28	15	7	27	28		
1	Gene	11110	11011	10100	11111	01111	11100	01111	00111	11011	11100		
	Fitness	90	86	61	77	7	86	7	0	86	86	586	90
	Angle	28	28	22	30	31	11	26	28	29	18		
2	Gene	11100	11100	10110	11110	11111	01011	11010	11100	11101	10010		
	Fitness	86	86	71	90	77	0	81	86	87	34	698	90
	Angle	22	29	16	31	31	23	30	8	4	12		
3	Gene	10110	11101	10000	11111	11111	10111	11110	01000	00100	01100		
	Fitness	71	87	16	77	77	71	90	0	0	0	489	90
	Angle	28	31	30	21	30	23	31	29	31	23		
4	Gene	11100	11111	11110	10110	11110	10111	11111	11101	11111	10111		
	Fitness	86	77	90	71	90	71	77	87	77	71	797	90
	Angle	26	31	10	28	31	30	2	23	30	28		
5	Gene	11010	11111	01010	11100	11111	11110	00010	10111	11110	11100		
	Fitness	81	77	0	86	77	90	0	71	90	86	658	90
	Angle	30	28	31	26	30	6	28	10	28	22		
6	Gene	11110	11100	11111	11010	11110	00110	11100	01010	11100	10110		
	Fitness	90	86	77	81	90	0	86	0	86	71	667	90

Table 5.7: GA2- 1 - mutation rate 20 % and crossover rate 60%

The results from the second run with the same algorithm are shown in table 5.8. The results show that the algorithm got nearly the exact same results when it comes to the best fitness for each generation. The best found solution was found in the second generation, and the same solution was found in the next 5 generations.

Gen		Ind 1	Ind 2	Ind 3	Ind 4	Ind 5	Ind 6	Ind 7	Ind 8	Ind 9	Ind 10	Tot	Max
	Angle	15	26	2	25	18	17	26	7	26	1		
0	Gene	01111	11010	00010	11001	10010	10001	11010	00111	11010	00001		
	Fitness	7	81	0	78	34	36	81	0	81	0	398	81
	Angle	28	14	24	30	29	16	23	25	25	18		
1	Gene	11100	01110	11000	11110	11101	10000	10111	11001	11001	10010		
	Fitness	86	0	72	90	87	16	71	78	78	34	612	90
	Angle	22	25	31	8	28	26	30	25	20	30		
2	Gene	10110	11001	11111	01000	11100	11010	11110	11001	10100	11110		
	Fitness	71	78	77	0	86	81	90	78	61	90	712	90
	Angle	20	20	26	25	29	30	9	30	2	30		
3	Gene	10100	10100	11010	11001	11101	11110	01001	11110	00010	11110		
	Fitness	61	61	81	78	87	90	0	90	0	90	638	90
	Angle	16	30	26	23	29	30	30	29	31	18		
4	Gene	10000	11110	11010	10111	11101	11110	11110	11101	11111	10010		
	Fitness	16	90	81	71	87	90	90	87	77	34	723	90
	Angle	23	31	30	14	26	30	30	29	29	30		
5	Gene	10111	11111	11110	01110	11010	11110	11110	11101	11101	11110		
	Fitness	71	77	90	0	81	90	90	87	87	90	763	90
	Angle	30	15	14	13	25	31	10	30	30	23		
6	Gene	11110	01111	01110	01101	11001	11111	01010	11110	11110	10111		
	Fitness	90	7	0	0	78	77	0	90	90	71	503	90

Table 5.8: GA2- 2 - mutation rate 20% and crossover rate 60%

The GA that was used for optimization of both angle and length was planned to have a bit-string representation where the first part is the angle, while the second part is the length. In this case, the use off crossover could possibly cross all the bits of the angle with all the bits of the length from the other parent. This could lead to good exploration, but it was desirable to not allow this to happen. Thus, that GA was planned to use a crossover rate at 0%. Therefore, the optimization with only the angle as parameter, tested how the GA without crossover would perform compared to with the crossover. The results from the first run with 0% crossover rate are shown in table 5.9. The results show that the algorithm was able to find the same solution, in the same number of generations. Thus, a GA without crossover also seemed to work well. Because of the stochastic nature of EAs, this algorithm was also run twice, to see if there was a big difference between runs.

Gen		Ind 1	Ind 2	Ind 3	Ind 4	Ind 5	Ind 6	Ind 7	Ind 8	Ind 9	Ind 10	Tot	Max
	Angle	6	17	31	10	5	4	2	22	15	13		
0	Gene	00110	10001	11111	01010	00101	00100	00010	10110	01111	01101		
	Fitness	0	36	77	0	0	0	0	71	7	0	191	77
	Angle	31	7	31	22	22	18	30	31	30	5		
1	Gene	11111	00111	11111	10110	10110	10010	11110	11111	11110	00101		
	Fitness	77	0	77	71	71	34	90	77	90	0	587	90
	Angle	23	5	31	30	30	12	20	31	18	31		
2	Gene	10111	00101	11111	11110	11110	01100	10100	11111	10010	11111		
	Fitness	71	0	77	90	90	0	61	77	34	77	577	90
	Angle	26	12	20	23	4	30	27	27	20	31		
3	Gene	11010	01100	10100	10111	00100	11110	11011	11011	10100	11111		
	Fitness	81	0	61	71	0	90	86	86	61	77	613	90
	Angle	23	31	23	21	7	22	27	27	16	16		
4	Gene	10111	11111	10111	10101	00111	10110	11011	11011	10000	10000		
	Fitness	71	77	71	66	0	71	86	86	16	16	560	86
	Angle	6	26	21	23	7	26	31	29	29	31		
5	Gene	00110	11010	10101	10111	00111	11010	11111	11101	11101	11111		
	Fitness	0	81	66	71	0	81	77	87	87	77	627	87
	Angle	31	26	23	26	31	27	31	12	28	13		
6	Gene	11111	11010	10111	11010	11111	11011	11111	01100	11100	01101		
	Fitness	77	81	71	81	77	86	77	0	86	0	636	86

Table 5.9: GA3- 1- mutation rate 20% and crossover rate 0%

The results from the second run with no crossover, and still mutation rate of 20%, are shown in table 5.10. These results show that the algorithm was still able to find the same best solution, but this time it did not find it before the fourth generation. By looking at the initial population, it can be seen that this run had a starting population that was further from the end result. Thus, the algorithm did still do a good job, and this is a good evidence of how the algorithm works to get partially better. A graphical comparison of the 5 runs of the GA with different parameters is shown in figure 5.6.

Gen		Ind 1	Ind 2	Ind 3	Ind 4	Ind 5	Ind 6	Ind 7	Ind 8	Ind9	Ind 10	Tot	Max
	Angle	1	9	10	10	8	3	13	2	21	6		
0	Gene	00001	01001	01010	01010	01000	00011	01101	00010	10101	00110		
	Fitness	0	0	0	0	0	0	0	0	66	0	66	66
	Angle	21	4	20	21	21	7	31	21	23	24		
1	Gene	10101	00100	10100	10101	10101	00111	11111	10101	10111	11000		
	Fitness	66	0	61	66	66	0	77	66	71	72	545	77
	Angle	21	20	20	20	27	13	22	21	29	21		
2	Gene	10101	10100	10100	10100	11011	01101	10110	10101	11101	10101		
	Fitness	66	61	61	61	86	0	71	66	87	66	625	87
	Angle	21	22	20	30	3	29	5	5	22	4		
3	Gene	10101	10110	10100	11110	00011	11101	00101	00101	10110	00100		
	Fitness	66	71	61	90	0	87	0	0	71	0	446	90
	Angle	20	14	25	10	21	21	30	21	31	22		
4	Gene	10100	01110	11001	01010	10101	10101	11110	10101	11111	10110		
	Fitness	61	0	78	0	66	66	90	66	77	71	575	90
	Angle	21	27	4	9	18	20	14	21	30	17		
5	Gene	10101	11011	00100	01001	10010	10100	01110	10101	11110	10001		
	Fitness	66	86	0	0	32	61	0	66	90	36	437	90
	Angle	30	27	20	22	30	21	20	30	16	29		
6	Gene	11110	11011	10100	10110	11110	10101	10100	11110	10000	11101		
	Fitness	90	86	61	71	90	66	61	90	16	87	718	90
	Angle	30	22	22	14	25	16	10	30	14	21		
7	Gene	11110	10110	10110	01110	11001	10000	01010	11110	01110	10101		
	Fitness	90	71	71	0	78	16	0	90	0	66	326	90
	Angle	7	30	30	31	31	16	21	19	30	22		
8	Gene	00111	11110	11110	11111	11111	10000	10101	10011	11110	10110		
	Fitness	0	90	90	77	77	16	66	59	90	71	350	90

Table 5.10: GA3 - 2- mutation rate 20% and crossover rate 0%

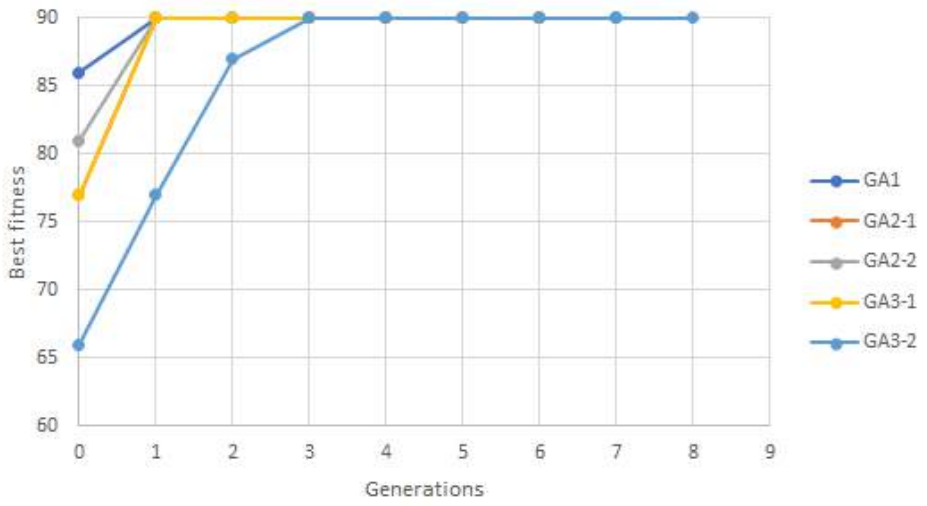

Figure 5.6: Comparison of the GAs run to optimize the gait with angle as parameter

5.3.3 Optimization with angle and length

The optimization of the gait with both angle and length of the legs as parameters was implemented with a GA. The representation was set to a bit-string with 5 bits for the angle and 5 bits for the length of the legs. To avoid mixing the angles and the lengths, the crossover rate was set to 0%. The mutation rate was set to 10% to keep the 1 gene per offspring ratio. The results from the optimization with angle and length are shown in table 5.11.

To test the hypothesis, these results needed to be compared to the results from the optimization with static morphology. The problem was that the results from the optimization with angle and length are not very representable. To show why they are not representable, a graph with the best fitness for each generation is shown in figure 5.7.

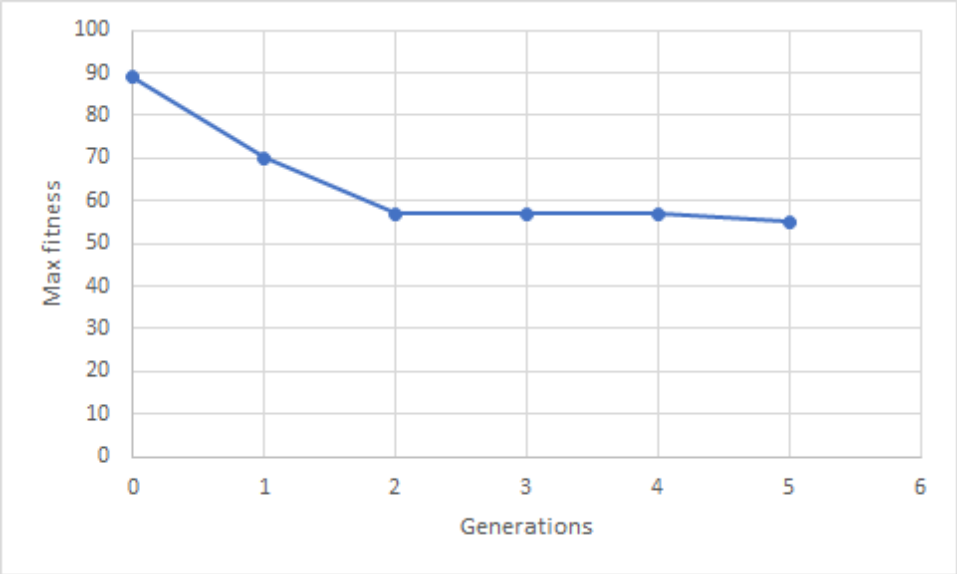

Figure 5.7: Max fitness for each generation for optimization with angle and length

Gen		Ind 1	Ind 2	Ind 3	Ind 4	Ind5	Indi 6	Ind 7	Ind 8	Ind 9	Ind 10	Tot	Max
	Angle, Length	23, 26	8, 11	23.28	19, 30	26, 5	20,10	15, 26	24,13	13, 22	16, 17		
0	Gene	10111 11010	01000 01011	10111 11100	10011 11111	11010 00101	10100 01010	01111 11010	11000 01101	01101 10110	10000 10001		
	Fitness	54	0	64	51	55	34	36	89	36	39	458	89
	Angle	15, 26	24, 5	18, 31	26, 5	1, 23	14, 27	15, 30	18, 21	20, 10	27, 31		
1	Gene	01111 11010	11000 00101	10010 11111	11010 00101	00001 10111	01110 11011	01111 11110	10010 10101	10100 01010	11011 11111		
	Fitness	36	70	15	55	0	31	30	57	34	0	328	70
	Angle	22, 17	28, 5	20, 2	18, 21	26, 5	18, 5	26, 13	15, 18	18, 21	14, 10		
2	Gene	10110 10001	11100 00101	10100 00010	10010 10101	11010 00101	10010 00101	11010 01101	01111 10010	10010 10101	01110 01010		
	Fitness	0	0	32	57	55	54	50	50	57	0	355	57
	Angle	27, 15	26, 13	14, 12	18, 14	22, 2	26, 5	16, 21	18, 17	18, 21	2, 7		
3	Gene	11011 01111	11010 01101	01110 01100	10010 01110	10110 00010	11010 00101	10000 10101	10010 10001	10010 10101	00010 00111		
	Fitness	52	50	0	24	40	55	12	55	57	0	345	57
	Angle	26, 15	18, 21	16, 6	26, 21	22, 30	22, 21	6.14	18, 21	22, 10	30, 21		
4	Gene	11010 01111	10010 10101	10000 00110	11010 10101	10110 11110	10110 10101	00110 01110	10010 10101	10110 01010	11110 10101		
	Fitness	0	57	5	27	29	20	0	57	17	0	212	57
	Angle	19, 21	30, 30	22.21	30, 30	18 ,17	26, 8	16, 21	22, 21	16, 4	19, 21		
5	Gene	10011 10101	11110 11110	10110 10101	11110 11110	10010 10001	11010 01000	10000 10101	10110 10101	10000 00100	10011 10101		
	Fitness	23	0	20	0	55	27	0	20	5	23	173	55

Table 5.11: GA - length and angle

Chapter 6

Discussion

6.1 General discussion

6.1.1 Challenges

6.2 Conclusion

6.3 Future work

Bibliography

[1] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. Springer, July 2015.

[2] C. R. Darwin, *On the Orngin of Species*. London: John Murray, 1859.

[3] R. Hecht-Nielsen, "Neurocomputing: picking the human brain," *IEEE Spectrum*, vol. 25, pp. 36–41, Mar. 1988.

[4] K. A. De Jong, "Genetic algorithms are NOT function optimizers," *Foundations of genetic algorithms*, vol. 2, pp. 5–17, 1993.

[5] J. H. Holland and D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[6] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies–A comprehensive introduction," *Natural computing*, vol. 1, no. 1, pp. 3–52, 2002.

[7] L. J. Fogel, *Artificial Intelligence Through Simulated Evolution.[By] Lawrence J. Fogel... Alvin J. Owens... Michael J. Walsh*. John Wiley & Sons, 1966.

[8] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*, vol. 1. MIT press, 1992.

[9] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, *Genetic programming: an introduction*. Morgan Kaufmann Publishers San Francisco, 1998.

[10] S. García, D. Molina, M. Lozano, and F. Herrera, "A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 special session on real parameter optimization," *Journal of Heuristics*, vol. 15, no. 6, pp. 617–644, 2009.

[11] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, pp. 67–82, Apr. 1997.

[12] R. Dawkins, "The Selfish Gene," *NewYorkCity: OxfordUniversityPress*, 1976.

[13] P. Moscato and others, "On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms," *Caltech concurrent computation program, C3P Report*, vol. 826, p. 1989, 1989.

[14] W. E. Hart, *Adaptive global optimization with local search*. PhD thesis, Citeseer, 1994.

[15] P. Merz, B. Freisleben, and others, "Fitness landscapes and memetic algorithm design," *New ideas in optimization*, pp. 245–260, 1999.

[16] J. M. Baldwin, "A new factor in evolution," *The american naturalist*, vol. 30, no. 354, pp. 441–451, 1896.

[17] G. E. Hinton and S. J. Nowlan, "How learning can guide evolution," *Complex systems*, vol. 1, no. 3, pp. 495–502, 1987.

[18] D. Whitley, V. S. Gordon, and K. Mathias, "Lamarckian evolution, the Baldwin effect and function optimization," in *International Conference on Parallel Problem Solving from Nature*, pp. 5–15, Springer, 1994.

[19] C. R. Houck, J. A. Joines, M. G. Kay, and J. R. Wilson, "Empirical investigation of the benefits of partial lamarckianism," *Evolutionary Computation*, vol. 5, no. 1, pp. 31–60, 1997.

[20] N. Krasnogor and J. Smith, "Emergence of profitable search strategies based on a simple inheritance mechanism," in *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, pp. 432–439, Morgan Kaufmann Publishers Inc., 2001.

[21] Y.-S. Ong, M.-H. Lim, N. Zhu, and K.-W. Wong, "Classification of adaptive memetic algorithms: a comparative study," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 36, no. 1, pp. 141–152, 2006.

[22] F. Neri and C. Cotta, "Memetic algorithms and memetic computing optimization: A literature review," *Swarm and Evolutionary Computation*, vol. 2, pp. 1–14, 2012.

[23] R. Meuth, M.-H. Lim, Y.-S. Ong, and D. C. Wunsch, "A proposition on memes and meta-memes in computing for higher-order learning," *Memetic Computing*, vol. 1, no. 2, pp. 85–100, 2009.

[24] J. C. Bongard, "Evolutionary robotics," *Communications of the ACM*, vol. 56, no. 8, pp. 74–83, 2013.

[25] K. Sims, "Evolving 3d morphology and behavior by competition," *Artificial life*, vol. 1, no. 4, pp. 353–372, 1994.

[26] K. Sims, "Evolving virtual creatures," in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pp. 15–22, ACM, 1994.

[27] J. B. Pollack and H. Lipson, "The GOLEM project: Evolving hardware bodies and brains," in *Evolvable Hardware, 2000. Proceedings. The Second NASA/DoD Workshop on*, pp. 37–42, IEEE, 2000.

[28] E. Samuelsen and K. Glette, "Real-World Reproduction of Evolved Robot Morphologies: Automated Categorization and Evaluation," in *Applications of Evolutionary Computation* (A. M. Mora and G. Squillero, eds.), vol. 9028, pp. 771–782, Cham: Springer International Publishing, 2015.

[29] R. A. Brooks, "Artificial life and real robots," in *Proceedings of the First European Conference on artificial life*, pp. 3–10, 1992.

[30] 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.

[31] N. Jakobi, P. Husbands, and I. Harvey, "Noise and the reality gap: The use of simulation in evolutionary robotics," in *European Conference on Artificial Life*, pp. 704–720, Springer, 1995.

[32] J. C. Zagal and J. Ruiz-Del-Solar, "Combining simulation and reality in evolutionary robotics," *Journal of Intelligent & Robotic Systems*, vol. 50, no. 1, pp. 19–39, 2007.

[33] J. Bongard, "Morphological change in machines accelerates the evolution of robust behavior," *Proceedings of the National Academy of Sciences*, vol. 108, no. 4, pp. 1234–1239, 2011.

[34] E. Samuelsen and K. Glette, "Some Distance Measures for Morphological Diversification in Generative Evolutionary Robotics," in *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO '14, (New York, NY, USA), pp. 721–728, ACM, 2014.

[35] E.-L. Ruud, E. Samuelsen, and K. Glette, "Memetic Robot Control Evolution and Adaption to Reality," 2016.

[36] P. G. de Santos, E. Garcia, and J. Estremera, *Quadrupedal locomotion: an introduction to the control of four-legged robots*. Springer Science & Business Media, 2007.

[37] V. R. Kumar and K. J. Waldron, "A review of research on walking vehicles," in *The robotics review 1*, pp. 243–266, MIT Press, 1989.

[38] S.-M. Song and K. J. Waldron, *Machines that walk: the adaptive suspension vehicle*. MIT press, 1989.

[39] M. J. Ballico, M. L. Sawley, and F. Skiff, "The bipolar motor: A simple demonstration of deterministic chaos," *American Journal of Physics*, vol. 58, no. 1, pp. 58–61, 1990.

[40] I. S. Lee, *Multi-phase bipolar brushless DC motor*. Google Patents, Nov. 1989.

[41] G. Gridling and B. Weiss, "Introduction to microcontrollers," *Vienna University of Technology Institute of Computer Engineering Embedded Computing Systems Group*, 2007.

[42] R. Bannatyne and G. Viot, "Introduction to microcontrollers. I," in *Wescon/98*, pp. 350–360, IEEE, 1998.

[43] C. Reas and B. Fry, *Getting Started with Processing: A Hands-On Introduction to Making Interactive Graphics*. Maker Media, Inc., 2015.

[44] D. Shiffman, *Learning Processing: a beginner's guide to programming images, animation, and interaction*. Morgan Kaufmann, 2009.

[45] D. Runberg, *The SparkFun Guide to Processing: Create Interactive Art with Code*. No Starch Press, 2015.

[46] M. Banzi and M. Shiloh, *Getting Started with Arduino: The Open Source Electronics Prototyping Platform*. Maker Media, Inc., 2014.