UiO : Department of Informatics
University of Oslo

Evolutionary optimization of robot morphology and control

Using evolutionary algorithms in the design of a six legged robotic platform

Tønnes Frostad Nygaard

Master's Thesis Spring 2014

Evolutionary optimization of robot morphology and control

Tønnes Frostad Nygaard

April 30, 2014

Abstract

Evolutionary robotics has yet to achieve mainstream status within the robotics research community, and among the reasons for this is the relative low maturity of the field. Most of the current research is still done in the scope of fundamental research, and has yet to be used to solve a wide range of real world problems, although much of it shows great promise. Conventionally designed and controlled robots solve an ever increasing number of tasks. For evolutionary robotics to catch up with, or even outperform traditional robotics techniques, a wide array of functional real life robots built on the foundation of evolutionary techniques is required.

This thesis proposes an evolutionary framework for evolving both morphology and control for a six legged robot. It features a parameterized 3D model with adaptable servo placement, base size, leg lengths, and a possibility for adding two more legs or tool holders to the front. It is also integrated into a simulation environment for evolutionary experiments. The algorithm tested is able to produce a varied set of solutions with different weights and speeds, and shows promise for solving more complex tasks or fitness functions.

The thesis also tests whether co-evolution of control and morphology is a feasible technique for robot design, by comparing the performance of a manually designed instance of the robot to two evolved models, both in simulation and reality. Machine learning is also used to lessen the reality gap present between the simulations and the physical experiments. Co-evolution of control and morphology shows a significant improvement over the manually designed morphology and gait, producing a robot which is 3% lighter and 49% quicker.

Contents

List of Figures

List of Tables

Acknowledgement

I would like to express my deepest gratitude to my supervisor, associate professor Kyrre Glette, for the continuous guidance and support throughout the thesis work. I would also like to thank PhD candidate Eivind Samuelsen for the great help and inspiration. The thesis would not have been the same without you two.

My sincere thanks to associate professor Mats Høvin for all your advice on the mechanics, to senior engineer Yngve Hafting for assisting with the practical work, and to all the other staff of the Robotics and Intelligent Systems research group for making a great learning environment the past two years.

I would also like to thank my fellow students, friends, family, and especially my significant other, Anniken Egeland Gaudernack, for the endless support.

Chapter 1

Introduction

1.1 Motivation

Robot design is an eclectic discipline, requiring insight into fields like mechanics, electronics, informatics, and mathematics. When designing robots, nature is often used as an inspiration. Consider for instance snake-like rescue robots [1], multi-joint robotic fish [2], spider inspired climbing robots [3], and humanoid robots designed for space maintenance and exploration [4]. Nature is an amazing source of inspiration, but solutions for all robot problems can not be found among the trees of the Amazon rainforest, in the depths of the Great Barrier Reef, or on the islands of Galapagos. Every product of natural evolution, be it plant or animal, has gone through millions of years of adaptation to its specific environment, and features thought to be advantageous in other situations might not be transferable to a robots morphology, control, and environment. Copying the processes of nature themselves, instead of trying to copy the results, offers an even greater reward, and is the basis for evolutionary robotics as a field of research.

Evolutionary computing, and specifically the field of evolutionary robotics, is slowly making its way into mainstream robotics research and development [5]. Most of the research on evolutionary robotics has been concerned with the automatic design of robot control [6, 7]. Goals for control evolution include object following or avoidance, herding, goal homing, foraging, hole avoidance, phototaxis, pursuit and evasion, object pushing, wall avoidance, circling, and many others [8]. This is evaluated on custom robots [9, 10, 11], general robot platforms [12, 13], or robot platforms made specifically for evolutionary experiments [14, 15]. A common denominator between most of the robot platforms used for evolutionary research is the lack of mechanical power and complexity required to perform relevant and useful real life tasks.

Although evolution of both control and morphology was proposed several years ago, current research focuses primarily on evolution of control alone [8]. Much of the work on evolution of morphology has been inspired

by the seminal work of Karl Sims' virtual creatures [16] and the evolved robots' leap into reality through the Golem project [17]. In contrast to the number of development platforms available for research on evolution of control systems alone, very few robot platforms exist for evolutionary experiments involving both morphology and control. Automatic design of both morphology and control is an exciting field, but is mostly limited to evolution in simulation [18, 19], or on custom robot platforms [20, 21]. Some research is, however, starting to appear using simple modular robot systems [22, 23]. The problem is, again, that these robots are made for very basic tasks, and lack the flexibility and adaptability for relevant real world use. A robot is only as good as its goal, or fitness objective, and as the field of evolutionary robotics matures, so will the goal of its robots. Evolvable robot platforms, capable of performing complex tasks and adapting to a wide range of environments, should be made available to researchers and the public to encourage research and use of the many opportunities given by new evolutionary techniques and tools.

1.2 Goal of the thesis

The first goal of this thesis is to make a robust framework for evolutionary experiments, both for gait development and optimization of design. Many of the robotic systems used for evolutionary experiments today are simple, and lack the flexibility and adaptability for more advanced goals and fitness functions. The developed platform should have a parametric physical design, and be customizable to suit a multitude of different goals and fitness functions, both simple and advanced. I hope to be able to make the robot and related work open source, and by doing this, contribute to the growing field of evolutionary robotics. In summary, the first aim of the thesis is to:

1. Design, implement, test and document a parameterizable legged robot for use in evolutionary experiments involving both morphology and control.

A second goal is to test whether using evolutionary algorithms to automatically improve parametric robot models is a feasible method for optimization of physical robots. Some research has been done on this in the past, but this has often been restricted to abstract robots or optimization of robots using few parameters. Optimization of both morphology and control system using a pre-defined parameterizable robot platform featuring 6 or 8 legs is an interesting case for evolutionary methods, due to the complex control requirements, and the adaptability to different real world objectives. In summary, the second aim of the thesis is to:

2. Investigate whether evolution of morphology and control is a feasible technique for realistic robot design.

1.3 Outline

The thesis is divided into seven chapters: introduction, background, tools and engineering processes, implementation, evolutionary experiments and results, physical experiments and results, and discussion.

Chapter 2 gives an introduction to the theory used in the thesis work, and a survey of the past work done. Chapter 3 contains an overview of tools and techniques used for making and testing the robots, or other practical aspects of the thesis work. Chapter 4 describes the implementation of both the evolutionary simulation environment, and the robotic platform used for the experiments.

Chapter 5 outlines the experiments done using the evolutionary simulation environment, and presents the results, along with a short discussion where appropriate. Chapter 6 describes the physical experiments done using the evolved robots, while chapter 7 contains a general discussion, along with a conclusion of the thesis and suggestions for future work.

Chapter 2

Background

This chapter attempts to give an overview of the field of evolutionary robotics relevant to this thesis. Robot and gait design are presented first, along with evolutionary algorithms, as they are both essential in evolutionary robotics, which is presented towards the end of the chapter.

2.1 Physical robot design

A popular place to get inspiration for robotic design is from nature. Countless robots have been designed with animals or nature as a guide, two of which can be seen in figure 2.1, but simply copying what is seen in nature has its pitfalls. A great part of an animal's body and mind may have evolved for part of its life separate from what a robot would experience. Simply copying an animal may therefore include features evolved as part of a tradeoff that does not exist for the robot, an example being secondary sexual characteristics, which most likely won't help a robot complete its task. By analyzing how and why animals evolved the way they did, one can isolate the features solving the problem at hand, although this can be very challenging as animal behavior and physiology are very complex systems. This is partly why evolutionary computing has become such a large research field - focusing on the processes of nature, instead of the results from those processes, yields results more directly transferable to new environments or goals.

Legged robots Legged robots have been the target of robotic research for many years, mainly due to their ability to traverse rough terrain or obstacles. Legged robots are, however, much more costly to develop and produce, and are generally slower and less energy efficient than their wheeled counterparts [26]. Earlier commercial robots, especially in the consumer market, were for these reasons mainly wheel-based. This is starting to change, and we see an increase in the market for legged robots, both in toys and for hobby use. Legged robotics, and especially gait generation, has also been used to study biological phenomena and learning. Legged robots are often defined by the number of degrees of freedom - the number of parameters defining the state of each actuator.

(a) An image of Salamandra Robotica 2 [24], a robot inspired by salamanders being developed at École Polytechnique Fédérale de Leusanne.

(b) One of several fish-inspired robots in the uwmfaus project [25] at the University of Washington.

Figure 2.1: Two robots both inspired by animals and the biological processes behind their movement and behavior.

2.2 Gait design

When designing gaits for robots, the first consideration is what type of actuators is used. DC motors require a voltage that is converted internally to a torque; a servo motor requires a goal angle that is sent to the motor control algorithm, while linear motors typically require a goal distance, in much the same way as a servo. A controller has to be designed to control the robot actuators, and serve as an interface between the physical layer and the control layer. Gait design typically involves both the design of the controller, and the optimization of the controller parameters, as seen in figure 2.2.

2.2.1 Gait types

Gaits are divided into two main groups, static and dynamic. A static gait is characterized by the walker being in balance throughout the gait sequence, while a dynamic walker will be out of balance during parts of the walk [27]. Whether the walker is in balance or not affects several different aspects of walking.

Static gaits An important trait of static gaits is that they can be paused at any time, with the robot still being in balance. A dynamic gait cannot be paused when not in balance, without continuing the current movement. This gives robots with static gaits more time to act on changes to tasks or from the environment, and is therefore less computationally expensive.

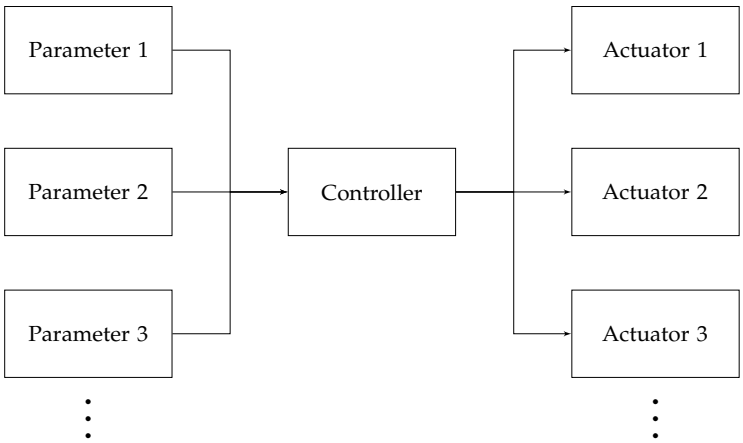


Figure 2.2: Flow chart of gait control.

Dynamic gaits One of the most important aspects of the dynamic gait is that a dynamic walker will, when not in balance, be under the influence of gravity. Dynamic gaits can therefore be made to exploit this fact, and let gravity do some of the work otherwise left up to the actuators of the robot. Because of this, most dynamic walkers expend less energy walking than their static counterparts. The addition of the force of gravity on the robot can also be exploited or higher speeds, if this is preferable over lower power consumption.

2.2.2 Controllers

There is an almost unlimited amount of different controllers available for robot control. Among the most used in evolutionary robotics, are variations of the pose controller, and the wave controller.

Pose controller The simplest way of programming the gait of a robot, is to record it as a set of poses with individual durations. This can be done by the jog-and-learn technique, driving each motor to a new position and recording it, the lead-and-learn technique, moving the robot manually and recording the new position, or using offline programming, utilizing a program or script to generate the robot poses [28]. These programs are often given parameters, and gaits generated can then be optimized by these parameters, instead of changing poses individually.

Wave controllers The problem with pose controllers is that they are hard to modify directly without having to change all motor values. A commonly used parameterizable controller is the wave controller. This takes the input of one or more wave signals (usually sine-waves), and calculates outputs to the motors by applying functions to the inputs with different parameters to modify the different output channels. These parameters can be set manually by engineers or designers, be set by optimization algorithms

before programming, or be continuously optimized during runtime of the robot using online optimization methods.

Other controllers Another way of generating walking gaits from parameters is to make individual mathematical functions for each limb angle, and simply apply these functions to the limb actuators. There are many ways to generate both functions and parameters other than traditional optimization algorithms, including learning algorithms [29], spline generators [30], neural networks [31], central pattern generators [32, 33], HyperNEAT [34], evolutionary algorithms [35], or a mix of techniques[36]. This is only some of the other techniques used for parameter optimization of robot controllers, but shows the diversity of tools available to a robot designer.

Multi direction controllers The problem with pose controllers, wave controllers, and specialized parameterizable controllers in general, is that they typically produce a gait moving the robot in a single direction, typically forward. This lowers the use for the gait in real-world problems considerably. General wave controllers which can control the robot in several directions have been made, but are typically much harder to hand design, learn, or evolve. Research has been done on making evolving or learning a collection of specialized controllers, instead of a single generalized controller. This enables shorter evolution or learning time, while still enabling the robot to walk in different directions or employ different walking techniques [37]. This is still a relatively new technique, and variants of this will most likely be used more extensively as robots become more mature, and the need for real world applicable controllers rise.

2.2.3 Forward/inverse kinematics

Forward and inverse kinematics are mathematical formulas used to calculate robot configuration or position. Forward kinematics are used to calculate position and orientation of the robot, given the angles or displacement of the joints. Inverse kinematics are used to get the angles or displacements of the joints, given a position and orientation.

Denavit-Hartenberg convention When calculating the forward kinematics, the Denavit-Hartenberg convention is a commonly used method for simplifying the kinematic analysis [28]. Each transformation is simplified to four basic transformations, given in equation (2.1).

$$A_i = Rot_{z,\theta_i} \times Trans_{z,d_i} \times Trans_{x,a_i} \times Rot_{x,\alpha_i} \tag{2.1}$$

Each parameter defines one physical feature of each joint; a_i defines link length, α_i (or r_i to more easily distinguish it from a_i) defines link twist, d_i defines link offset, while θ_i defines joint angle. Three are always static, while a fourth parameter is dynamic. Link length is dynamic for prismatic joints, where motion is linear, while joint angle is dynamic for joints with rotations,

called revolute joints. To enable a simplification like the DH convention is, two constraints must be applied to the axis systems, given in equations (2.2).

$$\begin{aligned} &\text{The axis } Z_i \text{ is perpendicular to the axis } X_{i-1} \\ &\text{The axis } X_i \text{ intersects } Z_{i-1} \end{aligned} \tag{2.2}$$

After setting up axes systems according to the constraints, the forward kinematics of all joints can be calculated by making the correct rotation and translation matrices and multiplying them together, resulting in a final forward kinematics matrix seen in equation (2.3).

$$\begin{matrix} cos(\theta_1) & -sin(\theta_1)cos(\alpha_1) & sin(\theta_1)sin(\alpha_1) & r_1cos(\theta_1) \\ sin(\theta_1) & cos(\theta_1)cos(\alpha_1) & -cos(\theta_1)sin(\alpha_1) & r_1sin(\theta_1) \\ 0 & sin(\alpha_1) & cos(\alpha_1) & d_1 \\ 0 & 0 & 0 & 1 \end{matrix} \tag{2.3}$$

Inverse kinematics Inverse kinematics calculate joint parameters based on position and orientation of the end point of the limb, and can be derived by many different techniques. Kinematics on simple robots may be determined by traditional geometric analysis, while more advanced kinematics may be solved by using approximations or numerical methods.

Using inverse kinematics for robot control The inverse kinematics only output the needed angles or displacements needed to reach a given end effectors position. To allow a robot to follow a calculated gait, several steps need to be completed. Generation of a gait is the first step, and involves analysis and calculation of the walking method. This generates a set of poses the robot needs to achieve. A path planning script calculates how the robot needs to move between these poses, and generates a number of intermediary positions for all limbs. These positions are fed into the inverse kinematics calculation to convert positions to actuator commands, which are sent to the actuators through a control interface. The cycle can be seen in figure 2.3.

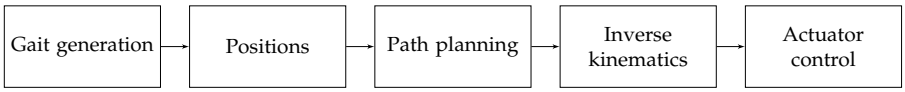


Figure 2.3: The basic work flow of using inverse kinematics for controlling revolute joints in executing a gait.

2.3 Evolutionary algorithms

Evolutionary computing has been steadily gaining popularity since it was created in the 1940s and 1950s [38]. It is a part of the field *artificial intelligence*,

and is mainly used in search and optimization problems. Evolutionary algorithms are parts of a class of algorithms that iteratively optimizes a problem, referred to as meta-heuristic optimization algorithms. It is a population-based approach, and uses operators like recombination and mutation on the individuals of the population to improve the current subset of solutions.

2.3.1 Overview of process

A typical run of an evolutionary algorithm is shown in figure 2.4, and starts with generation of an initial population. This can be done randomly, although initialization using problem-specific knowledge, or using previous solutions, are also common. Every individual or solution in the population is evaluated and assigned a fitness-value based on their quality. The main loop consists of a selection of solutions for recombination or mutation, most often based on their fitness values, but also age or other characteristics can be used. New individuals are generated based on the selected parents, and various recombination or mutation operators are applied to generate new solutions. Some of these solutions are then selected by the survivor selection, and a new population is made, often combined by the old parents and the new offspring. This loop runs until a stop-criterion, often based on time, age, change per n generations, or number of generations without improvement.

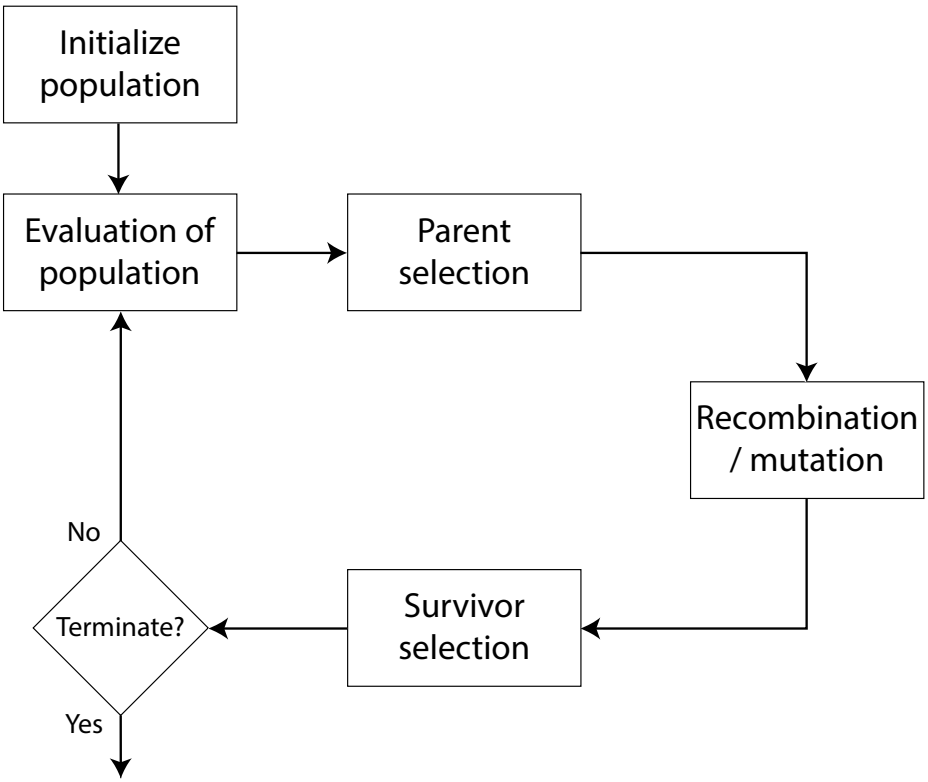


Figure 2.4: Flow chart of the basic EA process.

2.3.2 Genotype and phenotype

The distinction of *genotype* and *phenotype* is much used in evolutionary robotics, and stem from biology and genetics. Genotype is the hereditary information, while phenotype is the actual observed features of a solution. Each individual is represented in genotype space on the computer, and represents a solution in phenotype space. Each element of an individual in genotype space is often called an *allele* or *gene*. In evolutionary robotics, the term genotype space is used for the space where the actual search is taking place, often called the chromosome of the solutions. This is the digital representation of an individual, and is chosen by the designer of the algorithm. The phenotype space is the space of solutions in the original problem context, and involves the actual individuals posing a solution to the problem being solved. In a typical evolutionary robotics experiment, genotype includes binary information of control parameters, while the phenotype is the actual robots being represented by the genotype information.

2.3.3 Evolutionary operators

There is a multitude of different ways to generate new individuals. These are generally divided into two groups, depending on the number of inputs. Unary operators are often called *mutation*, and take one solution as input, and modifies it to produce a new individual. An operator with more than one input is often referred to as *crossover* or *recombination*, and is typically done on two parents to produce one or two new individuals, called offspring.

Mutation of floating point numbers There are many different mutation techniques available for mutation of computer representations of real values. One of the most common mutation techniques is the non-uniform mutation, using a fixed distribution. Each gene has a given mutation probability. If mutated, a number is drawn from a probability distribution with a given standard deviation, and added to the gene value. The normal (or Gaussian) distribution is the most commonly used statistical distribution for non-uniform mutation. This mutation operator ensures a non-zero chance of large mutations, while small mutations happening most of the time. The parameter controlling the amount of mutation is referred to as mutation step size, and can be the standard deviation of the normal distribution, or any other parameter controlling the statistical distribution from which the mutation is drawn from. An example of the mutation operator can be seen in table 2.1.

	Allele_1	Allele_2	Allele_3
Original individual	1.50	1.29	0.96
Drawn number	-0.04	0.35	0.07
Resulting individual	1.46	1.64	1.03

Table 2.1: Example of mutation of floating point numbers using the normal distribution with mean 0 and standard deviation 0.1.

Mutation of discrete values Mutation of floats with discrete values is normally not done the same way as continuous floats. Discrete values are typically represented in floating point numbers, or in integers and a factor or function that translates between integers and discrete values. A division size is often included to show the difference between discrete values, given a linear distribution. Non-uniform mutation could be used, and a simple truncation or rounding could be done to limit the numbers to the specified discrete values. An easier and more predictable way of mutating floats representing discrete values, is to step up or down to the next discrete step according to a given fixed probability, p. The probability of stepping up or down is the same, given by the mutation probability parameter p. If stepping occurs, the probability is applied again, and there is therefore a big probability of a small number of steps, while many steps are possible, yet unlikely. This is referred to as a one dimensional *lazy random walk* problem in statistics, an example of a Markov chain. An example of a probability distribution of this type of mutation is given in table 2.2. This technique is by many preferred for its ease of use, low computational cost, and statistically predictable behavior.

Steps	-4	-3	-2	-1	0	1	2	3	4
Prob.	.30%	1.11%	4.15%	15.51%	57.85%	15.51%	4.15%	1.11%	.30%

Table 2.2: Table showing a subset of the statistical distribution of mutation of discrete values, with $p = 0.5$.

Random reset mutation Random reset mutation can be used regardless of representation, and involves resetting one or several alleles to a new random value. The probability is typically chosen to be low, since this mutator has the power to move individuals very far in the search space, and hence the algorithm might quickly lose much of the exploitation aspect of the search. A common way of performing this mutation is to use an individual probability for each allele, but other variations including a probability of performing a fixed number of allele resets are also commonly used.

Uniform crossover for non-discrete floats Among the simplest and most commonly used recombination techniques, is the uniform crossover. This technique works on each individual allele, and has a chance p to inherit the gene from one parent, and the chance 1-p to inherit from the other parent.

Other crossover techniques may exhibit *positional bias*, varying chances of including genes based on the position. Uniform crossover does not have a positional bias, but exhibits *distributional bias*, a bias towards selecting a certain number of genes from each parent (given by the parameter p). An example of the mutation operator can be seen in table 2.3.

Parent 1:	0.60	-0.75	2.3	1.1	-1.05	1.35	1.40	0.25
Parent 2:	1.25	-1.25	1.3	-2.1	-2.30	-1.31	0.85	-1.55
Random:	0.91	0.42	0.65	0.98	0.60	0.86	0.15	0.25
Offspring 1:	1.25	-0.75	1.3	-2.1	-2.30	-1.31	1.40	0.25
Offspring 2:	0.60	-1.25	2.3	1.1	-1.05	1.35	0.85	-1.55

Table 2.3: Table showing an example of uniform crossover with $p = 0.5$.

Arithmetic recombination for non-discrete floats When working with floats, an alternative to selecting one of each allele for the new offspring, is taking an average value between the different values. This sets the new allele somewhere along the geometric line in genotype space between the two alleles of the parent. Whole arithmetic recombination sets all alleles to a weighted sum with some parameter α deciding where on the line between the two parents, the offspring should be placed. Another way of doing arithmetic recombination, is having a chance p of doing recombination on an allele, and a parameter α deciding where on the line the new offspring should be placed (typically also random, or 0.5). This results in a function (2.4) being run a number of times given by p.

$$\text{Allele}_{\text{child}_n} = \alpha \times \text{Allele}_{\text{parent n}} + (1 - \alpha) \times \text{Allele}_{\text{parent m}} \tag{2.4}$$

Parent 1:	0.60	-0.75	2.3	1.1	-1.05	1.35	1.40	0.25
Parent 2:	1.25	-1.25	1.3	-2.1	-2.30	-1.31	0.85	-1.55
Random p:	0.7	0.4	0.4	0.5	0.1	0.8	0.6	0.2
Random α_1:	0.80	0.99	0.25	0.36	0.31	0.67	0.26	0.35
Random α_2:	0.67	0.26	0.47	0.58	0.44	0.72	0.57	0.44
Offspring 1:	0.73	-0.75	2.3	-0.95	-1.05	0.47	0.99	0.25
Offspring 2:	1.04	-1.25	1.3	-0.76	-2.30	-0.57	1.09	-1.55

Table 2.4: Table showing an example of arithmetic crossover, with $p = 0.5$.

2.3.4 Parent and survivor selection schemes

Parent selection There is a multitude of different methods for selecting individuals from the population that is then used as parents for offspring generation. The main role of parent selection schemes is to select the best individuals from the population to perform the evolutionary operators on. Most algorithms use a probabilistic function to give a small chance for low-scoring individuals to advance, while still rewarding individuals with high

fitness scores with high probabilities of advancement. This ensures a higher degree of exploration than simply selecting the best individuals, and helps fight premature convergence.

Survivor selection Survivor selection schemes often rely on fitness value, age, diversity, or other measurements. Survivor selection is most often deterministic, and does not rely on probabilities, like parent selection. Keeping the best individuals unchanged for the next generation is referred to as elitism, and is often applied to inhibit losing the best solutions. This ensures that a number of the best solutions are always carried on to the next iteration, and is generally considered beneficial [39].

2.3.5 Parameter control and tuning

Evolutionary algorithms are still not (perhaps excluding modern evolutionary strategies) considered mature enough to be used effectively for black box optimization by many scientists. One of the main reasons for this is the lack of thoroughly tested or widely used methods for tuning of the parameters controlling the evolutionary run[1] [40]. Methods for improving parameters of evolutionary algorithms work on one of two areas, called parameter tuning and parameter control. Parameter tuning is the act of finding good parameters before the run is performed, and keeping the values fixed during execution of the search. Parameter control is, in contrast, a technique where parameters are changed during a run, either deterministically, adaptively or self-adaptively [41]. Figure 2.5 shows an overview of methods of parameter tuning and control.

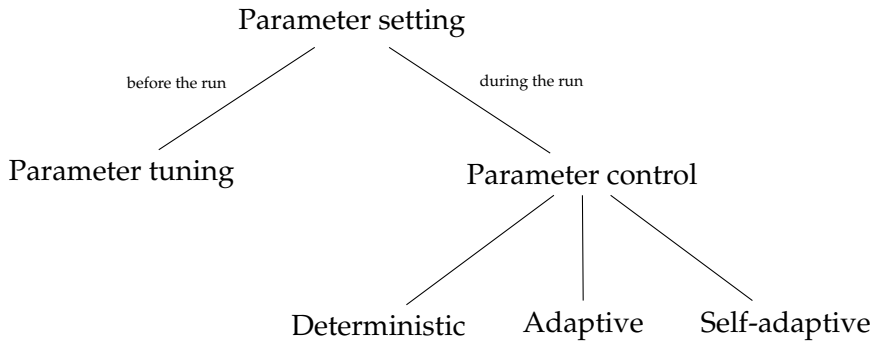


Figure 2.5: Flow chart of parameter optimization methods.

2.3.6 Exploration and exploitation

Any search or optimization algorithm may be classified by their degree of *exploration* and *exploitation*. *Exploration* involves testing solutions from new areas of the search space, and a random search is an excellent example of this,

[1] As in parameters controlling mutation or crossover types and probabilities, not problem-related parameters.

where new random solutions from the whole search space are selected for each iteration. *Exploitation* involves using previously acquired information to improve solutions. Good examples of exploitation heavy algorithms are hill climbing algorithms, which iteratively improves solutions by searching close to already known solutions.

No free lunch The no free lunch theorems state that finding optimal parameters for a wide range of problems is impossible [42], which shows the need for techniques for both parameter control and tuning. This is also why it is generally considered hard to set initial values for parameters, because good values for a seemingly similar problem might not perform well for the other. There has been done research on grouping problems into classes by various statistical methods, which in theory should share some common optimal and sub-optimal parameters and operators [43]. If these techniques continue maturing, they may one day be able to give a set of starting parameters based on what works for statistically similar problems, or at least lessen the need for manual parameter tuning.

2.3.7 Fitness

The quality of the solutions or individuals in a population is referred to as their individual fitness. A fitness function is a function returning the fitness score by estimating how close a solution is to a given goal or objective[8]. It is generally considered important that the fitness function gives an accurate evaluation of solutions. It is, however, for some problems more important that it is quick and easy to calculate, to lessen computational requirements. This results in the use of fitness approximation functions, which uses an approximate fitness model when the correct model is too computationally expensive, or if an explicit fitness function doesn't exist [44].

Fitness landscape The fitness landscape is often referred to when talking about the performance of search or optimization algorithms. This is the geometric space made up by the quality of the different solutions, and can be expressed as a function of the genotype space. Optimizing a problem can be seen as traversing this fitness landscape and finding a maximum or minimum point. Different variations of evolutionary algorithms are better or worse at escaping local optima, ridges, plateaus and other performance inhibiting effects. The fitness landscape is, of course, highly problem-specific, and this is one of the main reasons for the high number of different algorithms in use for optimization problems today. There is a wealth of different fitness functions in use for different problems, and which one to use has to be carefully chosen to maximize solution quality [8].

2.3.8 Diversity

Diversity is the measure of difference, spread, or variance within an evolutionary population [45]. High diversity shows a generation with individuals spread out over large areas of the genotype space. Low

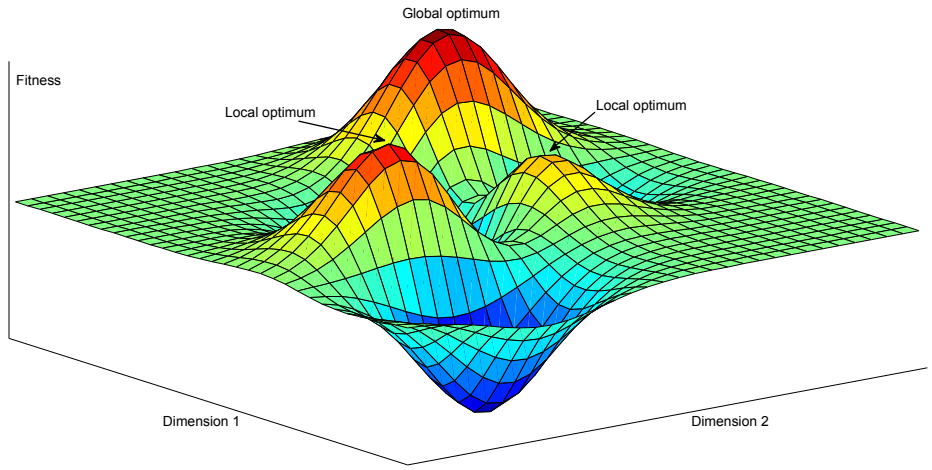


Figure 2.6: An example of a fairly simple three dimensional fitness landscape, including two local optimums.

diversities typically comes from grouped individuals in a fairly small part of the genotype space. An early loss of diversity can end in premature convergence on local optima [46, 47], so ensuring high diversity throughout the run can be beneficial. Methods such as analyzing neighborhoods, islands, niches, crowding and sharing are all techniques used for encouraging diversity through selection, replacement or mating [48]. Careful use of mutation and crossover methods can also ensure a wanted diversity development throughout the run. There is, however, no standard way of measuring diversity in a population, and the success of methods for calculation of diversity is problem specific [49].

2.3.9 Multi objective optimization

Many problems involve more than one measurement of success or quality, and several objectives may need to be defined for the algorithm to find optimal or near-optimal solutions. The problem with using multiple objectives is that for most problems, no single optimal solution exists. There are normally several solutions that feature different trade-offs between the different objectives, and a manual analysis of the results is more often required than with conventional single objective problem (except for evolutionary aided design, described in section 2.4.1). Since multi objective algorithms produce a multitude of different solutions, which might not explicitly outperform each other, some advocate for its use in what has traditionally been considered single objective problems. *Innovization* is a technique that uses multi objective algorithms and an analysis of resulting individuals as a design tool, and it is argued that conflicting objectives and the analysis of the solutions to these often produce better results than solutions typically found by single objective optimization [50].

Scalarisation An easy way of including several objectives within one fitness function is to merge several fitness functions together, using a weight for each function. This is often referred to as *scalarisation*. This is a quick way of expanding a simple fitness-based evolutionary algorithm into a multi objective solving algorithm, but it has several drawbacks. The most important drawback is that the weights must be decided before the runs, and setting weights before being aware of possible solutions is hard, and might damage the quality of final solutions.

Dominance One way to solve the limit of a single fitness function is to introduce the concept of dominance. A solution is said to dominate another solution if all objectives are at least equal, and at least one objective is better. With this concept, several fitness functions can be active at the same time without having to merge them using weights. This enables the algorithm to keep solutions with different trade-offs between objectives. The set of non-dominated solutions is referred to as the *Pareto Front*. See figure 2.7 for an example of a Pareto front.

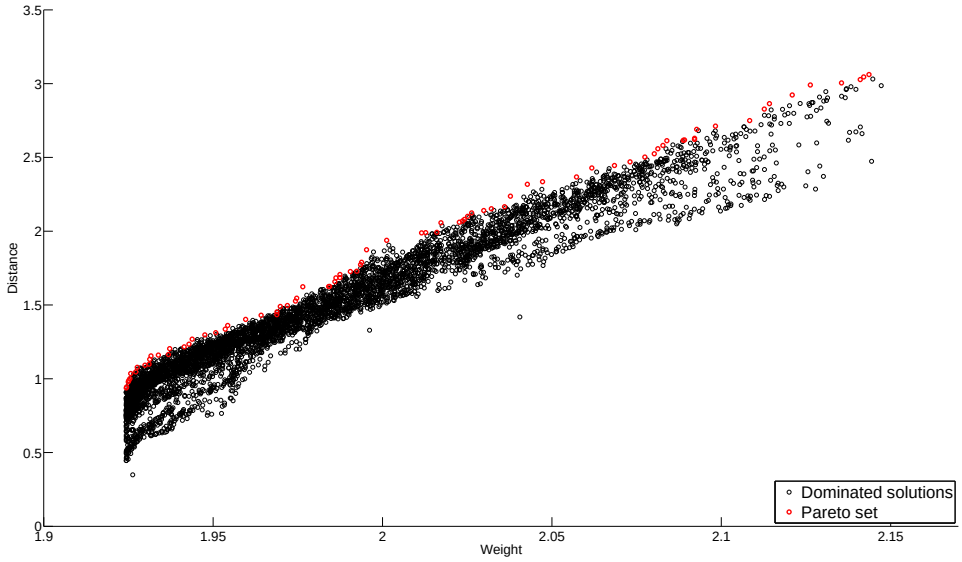


Figure 2.7: An example of a two dimensional Pareto front, where minimization of weight and maximization of distance is the goal. All points that are part of the Pareto set are colored red, while every dominated point is colored black.

NSGA-II An example of one of the most well known multiobjective algorithms in use today, is the nondominated sorting genetic algorithm 2, NSGA-II [51]. Before NSGA-II, no single multiobjective evolutionary algorithm was widely accepted, but rather a handful of different algorithms were in use for different types of problems. Among them was the original NSGA algorithm, which was among the top performing algorithms at the time [39]. There were several problems with previous algorithms,

including high computational complexity, a need for specifying the sharing parameter used for the diversity function, and a lack of elitism for some of the algorithms. NSGA-II fixed all these [51], and by doing so, established itself as one of the most widely used multiobjective algorithms today, along with SPEA-2 [52] and MO-CMA-ES [53]. The algorithm selects survivors from both parents and offspring based on both fitness and spread, without the need for parameters defining the diversity function.

2.3.10 Other algorithms from evolutionary computation

There is a variety of different techniques and extensions in evolutionary computing, in addition to evolutionary algorithms, used for a wide array of different problems. Three of the most commonly used in evolutionary robotics, are the evolutionary strategies, simulated annealing, and memetic algorithms.

Evolutionary strategies Evolutionary strategies were introduced in the 1960s [54], but changed greatly with the addition of self-adaption in 1977 [55], which has now grown to be one of the defining features of evolutionary strategies. Self-adaption involves placing some of the parameters of the search algorithm into the genes, which are then co-evolved with the solutions. This typically involves mutation step sizes of recombination probabilities, and enables an adaptation of these parameters throughout the runs. This enables evolutionary strategies to customize parameters for the fitness functions, and even enables efficient search through changing fitness landscapes [56]. Evolutionary strategy algorithms are often defined by their survivor selection scheme, represented as $(\mu \ / \ \rho \ \dagger, \lambda)$. ρ represents the number of parents used for the creation of one offspring, and is often omitted from the function if no recombination is done. μ is the size of the parent pool, while λ is the size of the offspring pool. Plus is used in the function to signify survivor selection from both parent and offspring pool, while comma signifies a selection of survivors from the offspring pool alone.

Simulated annealing Simulated annealing is a probabilistic generate-and-test search technique [57]. It uses the Boltzmann selection mechanism, which changes the selection pressure according to a pressure schedule, which enables the balance between exploration and exploitation to change during the search. The idea is that exploration is more essential early in the search, and that the reward from exploitation rise as a larger part of the search space is tested. Simulated annealing works by testing a neighborhood solution, and moving to the new individual if it is an improvement. Inferior solutions are accepted in accordance with a probability given by a temperature function. Parameters typically defined for simulated annealing, is the cooling function with initial temperature value, the number of moves performed at each temperature called the epoch length, the function for finding new individuals, and the probability of allowing inferior solutions.

Memetic algorithms Problem-specific algorithms often surpass evolutionary algorithms in performance, but are often tuned to very specific scenarios or problems, and can be very time consuming to develop and test. They are also more dependent on engineering principles and the humans designing and programming the algorithm. A balance between a general solver and the use of problem-specific knowledge can be achieved by using memetic algorithms. Memetic algorithms are the combination of evolutionary algorithms with problem-specific algorithms, often in the form of smart initialization, problem specific operators, or local improvement of solutions by various search algorithms, as seen in figure 2.8.

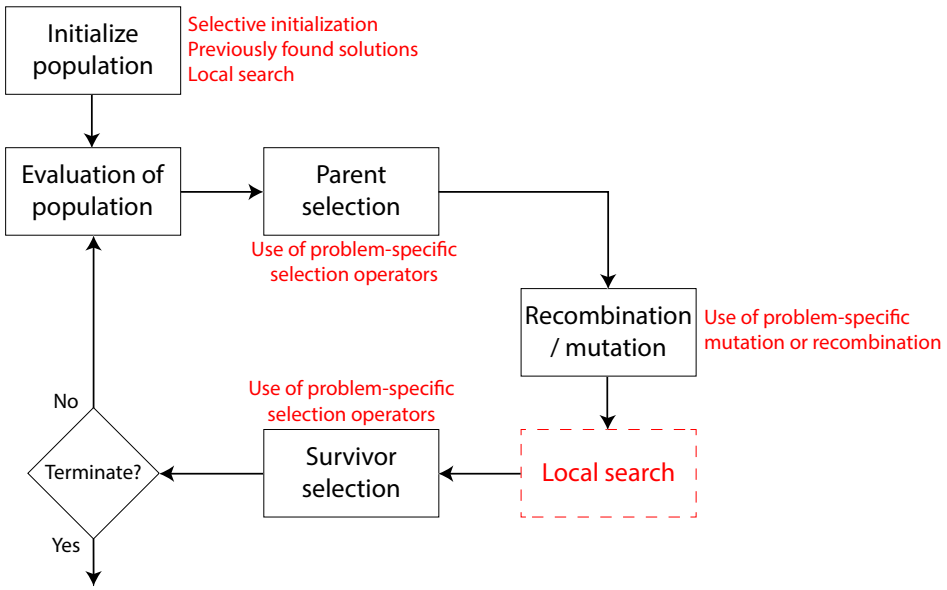


Figure 2.8: The memetic algorithm process, as compared to figure 2.4, with changes indicated in red. Various variations of local search algorithms are often chosen as problem specific algorithms.

2.3.11 Evaluation of EA performance

Evolutionary algorithms are most often non-deterministic. This is because of the randomness of the operators used, and results will therefore vary between different runs. This is an intended feature, since better results could potentially be produced by doing several runs. A problem with the random nature of evolutionary algorithms is that comparing results can be challenging. The law of large numbers describes that as more experiments are conducted, the closer the results get to the expected value. This theorem transfers to evolutionary algorithms by requiring several runs of each algorithm to be able to draw a conclusion on performance with any degree of certainty. Many scientists do several runs, and compare average performances, but without at least including a confidence interval, a realistic conclusion is hard to defend. More advanced statistical methods need to be employed, to conclude on performance with a high degree of certainty.

Box plot The box plot is a much used way to visualize data from evolutionary runs to make it easier to compare results, see figure 2.9 for the general structure of the plot. It shows the three quartiles, the set of points dividing the data set into four, while showing maximum and minimum values using whiskers on each side of the box. It also filters the data by classifying all points beyond 1.5 times the interquartile range below or above the first or third quartile as outliers, commonly referred to as a Tukey filter. These whiskers are often referred to as fences in statistics. Whether points beyond the fences are actually considered as outliers depends on the problem, the testing environment, and the degree of noise in the measurements. A notch is sometimes added around the median with size being proportional to the interquartile range of the sample, and inversely proportional to the size of the sample. It is typically used to distinguish whether differences in medians are statistically significant or not by comparing differences in notches, not medians directly [58].

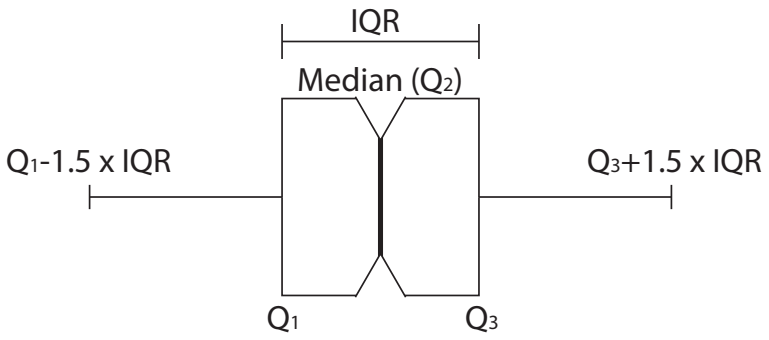


Figure 2.9: An overview of the box plot structure.

Rank-sum test The Wilcoxon rank-sum test, also called the Mann-Whitney U test, is a test of the hypothesis that two populations are the same. It does not assume any distribution or structure of the data, and is therefore considered non-parametric, in contrast to the much used Student's t-test. Non-parametric tests are generally considered better suited for evolutionary robotics than parametric tests [59], and the Wilcoxon rank-sum test is among the most widely used tests, in addition to the analysis of variance test (ANOVA) [60]. The method involves sorting the observations, and for each observation from sample one, counting and adding the number of observations from sample two with lower ranks. This result in a U statistic that indicates at what certainty the null hypothesis of similar populations can be rejected. A P value can be calculated, and indicates the possibility of obtaining a result at least as extreme as the one that was observed, assuming the null hypothesis is true. The null hypothesis can then be rejected if this value is less than a certain significance level, typically 0.01. The rank-sum

test does, however, only allow comparison of medians of two unknown distributions, and can not be used for comparing the mean difference or other interesting parameters, or comparing more than two populations simultaneously [61].

Peak and average performance A problem faced when comparing results from several runs of evolutionary algorithms, is whether to compare peak or average performance. A high average performance is typically the goal if the algorithm developed needs to be used in an environment or for a problem that only allows a single run of the algorithm before a solution is needed. This is typically the case for repetitive problems, like calculating postal routes throughout the day, or the real time placement of stock in a warehouse. The problem, however, arises when there is time for several runs of an algorithm, and it is easy to select the best performing individual. This is the case in most scientific research, and in design problems in the industry with high computational resources. Peak performance is less statistically significant since it is based on a single data point from each evolutionary run, while the average performance is based on all individuals from the last generation. It is therefore harder to compare peak performance throughout the work on designing and tuning an evolutionary algorithm. It is therefore important to be aware of whether the algorithm will be used for design problems or repetitive problems, and compare and develop algorithms and operators accordingly.

2.4 Evolutionary robotics

Manually designing robot controllers or morphology can be very challenging. Automatic design and optimization is a growing field, especially as the complexity of robotic systems increase, along with demands for solving more advanced tasks. The field of evolutionary robotics is mainly concerned with using evolutionary computation for creating robot controllers. Optimization of artificial neural networks, central pattern generators, wave controllers, or a variety of different parameter-based controllers is typically done with evolutionary techniques.

2.4.1 Evolutionary robotics processes

Doncieux et al. [62] distinguishes four different uses of EAs in robot development:

- parameter optimization
- evolutionary aided design
- online evolutionary adaptation
- automatic synthesis

Parameter optimization Parameter optimization is the most common way of using evolutionary algorithms in ER, and consists of optimizing one or more parameters towards one or more pre-defined goals. Common parameters to optimize are control parameters [12, 37, 63], design parameters [20, 9, 35], or parameters for artificial intelligence or learning algorithms [11, 64, 65, 36] running on the robot. Parameters found are often transferred to the control system or physical robot, and is typically not analyzed further before transferral.

Evolutionary aided design Evolutionary aided design shares several features with parameter optimization, but differs in how the results from the algorithm is applied to the robot or control system. While a parameter optimization process usually involves using the evolutionary results directly, evolutionary aided design instead uses the parameters found to aid analysis and optimization of the problem - often done by experienced engineers, before manually designing the final solution. This is a more complicated process than parameter optimization, but often leads to better results [50]. This process is also often used for analyzing Pareto fronts to decide on design strategies early in the process, and help engineers understand the different objectives and their relative requirements.

Online evolutionary adaption Online evolutionary adaption is most commonly done on the actual robot after design and manufacture. It is done to customize control or morphology to new environments, situations, or goals. This is normally done less aggressively than both parameter optimization and evolutionary aided design, since the performance of the robot while optimization is running might be of importance. The goal is often a slow adaptation and improvement of the working robot, while not inhibiting the day-to-day operation to a large extent. This is still a relatively new use of evolutionary robotics, but promising results are starting to emerge [66, 67].

Automatic synthesis Automatic synthesis is the process of designing the whole robot, including morphology and control, completely by evolutionary algorithms. Parts or modules are often hand designed before the algorithm combines these into different morphologies and tests different control systems. Human intervention is minimized, and generated robots can also be subject to an online evolutionary adaption. This is a field that, according to many, started with the previously mentioned work of Karl Sims [16], but has not since seen many major breakthroughs.

2.4.2 Evolving control

The most commonly evolved part of robots is the controller. Among evolved controller types are neural networks [31, 68, 69], gait parameter controllers [9, 12, 37], evolvable hardware (FPGAs) [70, 71], fuzzy logic controllers [11, 72], and spline controllers [30, 73], which is only a small selection of the

controller types evolved in ER research. There is a wide range of objectives and fitness functions used, and several robot platforms have been made specifically for evolutionary experiments [14, 15, 74]. Most of the research done on evolving robot controllers use parameter optimization, though the use of evolutionary aided design and online evolutionary adaption has grown the last couple of years.

2.4.3 Evolving morphology

Among the early uses of evolutionary computation for robotics, was the seminal research done by Karl Sims [16], in which he used evolutionary algorithms to develop morphology and control. These virtual creatures were fairly abstract, and accomplished tasks like walking, following, jumping and swimming. The Golem project [17] took evolutionary robotics and the virtual creatures into the physical world by using rapid prototyping to build the machines and test them in real life. The robots were rather abstract, but the work contributed to the initiation and growth of morphological evolution within evolutionary robotics.

Robots for evolution of morphology Much of the research in evolution of morphology since, use custom modular robot systems, like the fairly abstract robotic life forms of Lipson and Pollack [21] or Samuelsen [20]. Other researchers try to use openly available general robot platforms like the tinkerbot robots [22] or lego-servo modular robots [23], which enables evolution of simple modular systems. There is, however, a lack of robotic platforms enabling evolution of morphology for common robot designs, which produce legged robots able to solve real world objectives and complex tasks.

2.4.4 Search complexity

Many techniques in ER require many iterations to produce good results, and continuous testing on a physical robot can be very time consuming, and often suffer from noisy measurements caused by external causes. One of the greatest advantages of simulating is that you can exclude external disturbances that might otherwise serve as noise in your test data. Simulation ensures consistent test data, and removes elements like worn out servos or parts, temperature, environmental conditions etc, from the evaluation process. Earlier evaluations within an evolutionary run can be done using a lowered resolution or tolerance to speed up the search, while not significantly degrading final results [75].

Environment The environment in which a robot is simulated or tested can have an impact on the resulting complexity on both controller and morphology. Higher complexity of the environment leads to a higher evolutionary pressure, which again leads to more complex controllers or body forms [76]. Auerbach et al. also posit that more degrees of freedom or more sophisticated controllers reduce the need for complex morphologies

when given the same goals in the same environment. This shows that as the evolutionary robotics field mature, and both environments and goals of the robots gets more complex, search spaces might grow exponentially, to facilitate the added complexity to both controllers and morphology. There are, however, several techniques to avoid searching through massive search spaces. Evolving for novelty, and completely disregarding fitness, has been shown to work by not misdirecting the search towards suboptimal solutions [77]. This is often referred to as diversity-based evolution. Another technique involves an incremental approach to evolution, where the problem is divided into sub problems which can be solved more easily, then combined [78]. This is only two of a wide array of techniques that will likely be used more as robotics technology mature.

Reuse and modularity Many robot morphologies used for evolution has been divided into building blocks that together form a complete robot. Using a hierarchy of a group of evolvable parts enables simpler construction of the finished robot, enable reuse of modules across different projects, and enables single modules to be changed without having to modify the rest of the system. Using an evolvable hierarchy of parts adds a tradeoff between modularity and regularity, modularity reducing the amount of information by duplicating parts, while regularity increasing the extent of changes by affecting all copies of the module on the robot [79].

2.4.5 Reality gap

Reality gap is the difference between simulated fitness, and the fitness experienced in the real world [80]. A perfect simulation of the reality is practically impossible, and errors early in a simulation cause discrepancies between simulation and reality that grow exponentially as time passes by, as given by chaos theory. There are several methods to reduce the significance of the reality gap.

Reality gap reduction One of the easier methods of crossing the reality gap, although not as effective, is to add random noise in specific ways to all aspects of the simulation to ensure robots that work well in simulation also work in reality [81]. Usage of the physical robot performance for adaptation of the simulation parameters has also been successful [82, 83]. A more recent approach that has shown great promise, it to calculate a transferability score for a subset of solutions, approximate a transferability function, and use this to favor solutions that exhibit a low difference in simulator behavior to real world behavior [84]. This might, however, discourage solutions that utilize movements that are hard to simulate accurately, but that might perform well in reality.

Chapter 3

Tools and engineering processes

This chapter attempts to give an introduction to the different tools and processes used throughout the thesis. The background chapter lays the theoretical groundwork, while this chapter focuses on the physical work done, and the engineering parts of the thesis, including 3D design of the physical robots, choice of parts, simulation in evolutionary systems, and the testing, validation, and verification methods used in the thesis. Table 3.1 shows an overview of tools and software used.

Tool	Name	Version
FDM 3D printer	Fortus	250mc
Photopolymer 3D printer	Objet	500
Servos	Dynamixel	AX12 and AX18
Battery	Turnigy	Nano-tech 2200mah
Wireless module	Xbee	Series 1, 1mw
Microcontroller	Arduino	Uno R2
Microcontroller	Arbotix	RoboController
Microcontroller programming	Arduino	1.0.5
Data capture and analysis	Processing	2.1
CAD package	Solidworks	2013 Educational version
Slicer	Insight	9.1
Motion capture software	Arena	1.7.3000
Statistics and graphing package	Matlab	2013a

Table 3.1: Table showing the tools used in the work on the thesis. The first part shows hardware used, while the second part includes software.

3.1 3D design

Solidworks[1], made by Dassault Systèmes, is a fully featured 3D computer aided design package (CAD), including simulation and rendering toolboxes.

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

It is among the most popular CAD packages, both for full 3D modeling, 2.5D modeling, and 2D drawing. 2.5D modeling involves sketching features in 2 dimensions, and projecting this into the third dimension. This limits the available geometry slightly, but makes the models far easier to both 3D print and mill. Solidworks also feature parametric modeling through design tables, linked dimensions, and equations. Parametric models enables the design files to be edited and customized without in-depth CAD knowledge, which in turn encourages open source models and easier verification of research by enabling scientists to duplicate experiments involving changing variables or conditions.

3.2 3D printing

Use of 3D printing The use of 3D printing has increased in popularity the last couple of years, and is used more and more in robotics research and development. 3D printing is an additive process used to construct parts or assemblies out of different materials, plastics like ABS or PLA being most common. It allows a developer to quickly produce physical parts from 3D models and drawings. The ability to do rapid prototyping lowers the price and time constraints of many projects, and affects the entire design and manufacture process. 3D printing also encourages collaboration between developers and scientists by enabling different people or groups to quickly and easily replicate parts or setups in a very similar fashion.

Model preparation Once the designs are made with practically any 3D CAD package, it has to be converted into commands for the 3D printer. The process of generating tool paths for the 3D printer from a 3D model is called slicing, because the first process of generating tool paths includes slicing the complete model into horizontal layers. Examples of popular free slicers available today are Slic3r[2], Cura[3] and Kisslicer[4]. Each perimeter is then calculated, and these paths make out the shell of the component. After calculating the perimeter of the part, the infill has to be generated. There are several different infill types. Solid infill fills the model with material, and gives a strong and heavy model that takes a lot of time to print. Other infills typically fill a given percentage of the area, and give a quicker print with close to full strength. Popular non-solid infill patterns include lines or rectilinear patterns. Honeycomb, Hilbert curves or different types of spirals are also popular.

Printing process There are a number of different types of 3D printers in use, but the most common, both in hobby use and for professionals, are FDM printers - fused deposition modeling (or FFF, fused filament fabrication). FDM is an additive method that works by extruding a plastic or metal wire

[2]http://slic3r.org/
[3]http://wiki.ultimaker.com/Cura
[4]http://kisslicer.com/

and melting the material onto the model. Models are built one layer at a time in succession. Since the process of 3D printing consists of depositing strands of material, models with features hanging in the air, like bridges or crosses, are hard to print. This is often done by adding support under the affected areas, either by using another material that can more easily be removed, or by printing in the same material, but using a pattern that doesn't attach to the final model itself. This enables a 3D printer to print features in mid-air, but requires removal of the support after printing. As with infill, there is a multitude of different patterns and techniques for generating support material, two of which can be seen in figure 3.1.

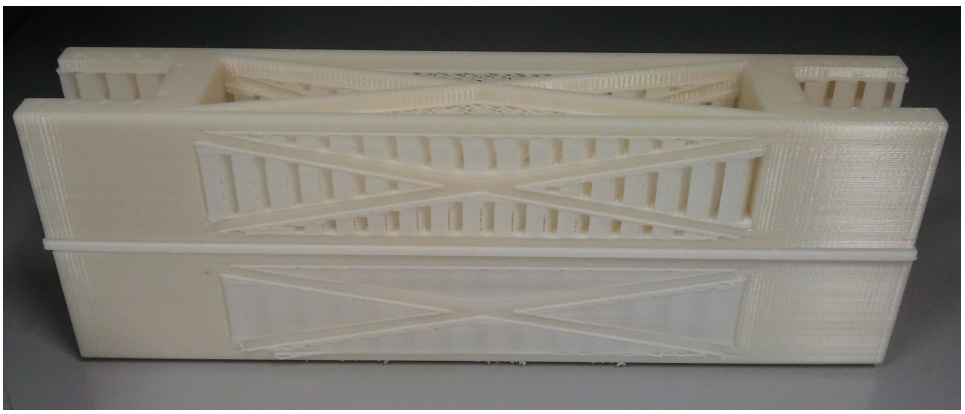


Figure 3.1: An example of two different support patterns for the same part, with *breakaway* style on the top and *smart* style on the bottom.

3.3 Electronics and mechanics

Microcontroller Arduino[5] has, since the release of the first board in 2005, become among the most popular and widely used single-board microcontrollers available for hobby use. Arbotix[6] is an Arduino compatible robot controller designed to be easy to interface to motors and servos, and is compatible with Arduino, an already widely used system for most robotics departments or hacker spaces.

Servos There is a massive amount of different servos on the marked, and they each target one of many different market segments. Professional servos are typically at a price range not available to hobby or scientific use, but high end hobby servos exist, and perform comparable to the cheapest professional servos. Evolutionary robotics often include closed loop controllers, where sensory data is fed back into the controller, so the ability to read servo positions is vital for many projects. Several servos include a way of outputting the positions through PWM on the data lines, but this is not easy to read with the Arbotix (or any other Arduino board),

[5]http://arduino.cc
[6]http://www.vanadiumlabs.com/arbotix.html

and a digital bus-style servo is preferred in most cases. The Dynamixel[7] range of servos from Robotis features low cost, high power, and an easy to use bus interface highly compatible with the Arbotix or other Arduino-based system. The servos can be controlled by position, speed and torque, and work in continuous rotation mode if needed, which enables a wide range of controller outputs. Position, speed, load and temperature can also be read from each servo through the digital bus, which is a big plus when using closed loop controllers, since this reduces the need for other external sensors. The Dynamixel range of servos is considered an excellent choice when both precision control and dynamic operation are needed [85], and is extensively used in robotics research, especially for small humanoid robots[86, 87, 88].

Wireless communication When it comes to control of legged walking robots, wireless control is the norm. This can be achieved by many different technologies or techniques, but use of xBee[8] wireless RF Modules are certainly among the most popular among hobby and research users. They are normally used in a point to point network as a direct replacement for a TTL serial cable, but can also be used in other network topologies like star, mesh or cluster tree networks. The most popular xBee boards work on 2.4GHz, 900Mhz, bluetooth or wireless computer networks. All modules have a common pinout and are interchangeable, given the correct supply voltage. Popular alternatives for the xBee modules include simple radio frequency transmitter/receiver pairs on 315MHz, but these typically involve more work, as modulation and a communication protocol with checksum calculation has to be implemented to get good performance.

Power When considering power needs, an analysis of the power requirements of all individual components involved has to be conducted. Typical voltages required for motors and servos range from 5-12v, while microcontrollers typically run on 3.3v-5v. Several different voltages can be combined in the same system, given a common ground connection. When doing extensive testing of robots, a tethered connection with communication and power is mostly used. This removes error sources like wireless communication and power systems while testing. When running the system wirelessly, a battery has to provide power for the whole system. If different voltages are required, a voltage regulator is normally added to reduce the complexity and added weight of carrying several different batteries. Among rechargeable batteries (often referred to as secondary cells), lead acid, lithium, NiCad and NiMH are the most common. Use of NiCad batteries is declining because of memory effects, a loss of energy capacity after charging cycles, and the toxic cadmium core. NiMH is equally declining due to high self-discharge rates. Lead acid batteries have been used extensively for bigger robots due to the ease of recharging, low price, relatively low internal resistance (and therefore large current output), and high availability. They have, however, lost marked shares the last couple of years due to

[7] http://www.robotis.com/xe/dynamixel_en
[8] http://www.digi.com/xbee/

high weight and size, and the rise of better lithium batteries. Lithium batteries, often called LiPo or Li-ion, have been steadily gaining market shares, especially in areas of RC use and robotics. They feature comparable energy storage capacity to NiMH batteries, comparable output current to NiCad batteries, no memory problems, and considerably lower weight than all previously mentioned batteries. They are, however, held back by high price and complex charging requirements.

3.4 Testing

Validation and verification are both important to consider when it comes to product development. Validation relates to whether the product developed meets the needs of the intended users, while verification involves an evaluation of whether the product satisfies the given requirements or specifications. Verification of robot design plans often include a structural analysis of all parts, showing stress or deformation under various loads, and potentially uncovers material failure, structural instability or inefficient designs. Verification of finished robots is typically done by comparing the performances to the predefined goals of the development.

3.4.1 FEM simulation

Finite element analysis is a function in Solidworks that utilizes the finite element model (FEM) to calculate strain, stress, and displacement on parts, both under internal and external forces. This is used to verify the model against known external forces inflicted under typical use. Especially important is the generated stress graphic, which shows stress intensity throughout the model. The three principal stresses in each axis is combined to form what is referred to as Von Mises Stress, which can be compared to the yield stress of a material to predict material failure. This is used to reinforce the model where needed, and gain a better balanced form. Factor of safety, or FOS (or factor of ignorance, as some refer to it as) is a calculated factor that states how much stronger the model is than it has to be, for the simulated load (3.1). Any number under one indicates it would not be able to handle the load specified (given a perfect simulator), while any number above one gives a margin of safety equal to the factor of safety minus one (3.2).

$$\text{Factor of safety} = \frac{\text{Material Strength}}{\text{Design Load}} \tag{3.1}$$

$$\text{Margin of safety} = \text{Factor of safety} - 1 \tag{3.2}$$

Problems with using FEM on 3D printed models The problem with doing a FEM simulation on 3D printed objects is that the objects are not uniform. The printer deposits the print material in a continuous path, and does not necessarily have 100% infill for the whole model, as seen in figure 3.2. Models are also much stronger in the plane they were printed in, as

more force is needed to break the continuous lines of plastic, than to split the model between two layers. This causes yet another source of inaccuracy between simulation and real life. In theory, the software to prepare the model for 3D printing could output the model with the right paths, and simulation could be done on the new model, but that is not yet supported in any major CAD package.

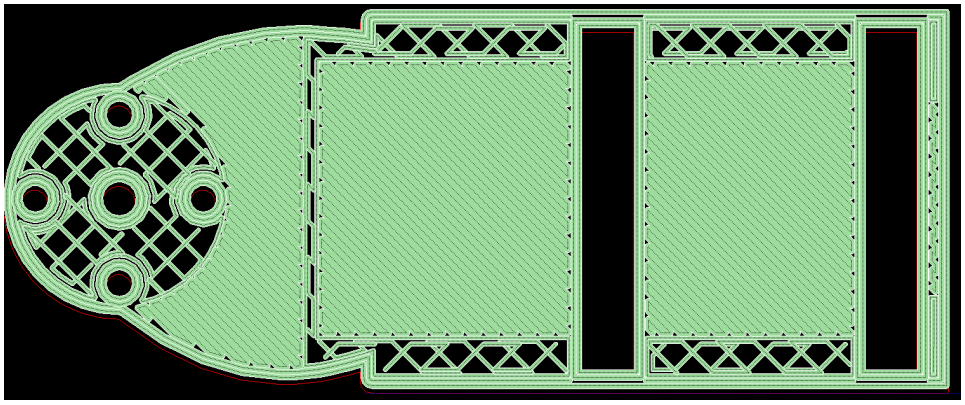


Figure 3.2: Drawing of the print path for a given layer of a printed model.

3.4.2 Motion capture equipment

There are various methods of testing the movement of robots. One of the most popular methods among researchers and professional users is using motion capture equipment to record position and orientation of the robot. Typical professional equipment gives a precision in the scale of millimeters to sub-millimeters. Motion capture systems work on a variety of different technologies, but most are optical systems that rely on either passive or active markers. Passive markers are reflective balls that reflect light at a higher ratio than the other materials on the scene, so a simple threshold of the incoming light can decide the position of the markers. A problem with passive markers is the difficulty in differentiating between different markers. This is not the case with active markers, where the markers typically consist of light emitting diodes that can be switched on sequentially. Several cameras capture the scene at the same time, and the positions of the markers on each 2D image is used to generate the 3D positions of the markers.

Setup at UiO The motion capture setup used at the University of Oslo consists of eight OptiTrack Flex 3 cameras with infrared lights, and passive reflective markers. Data recording and analysis is performed by version 1.7.3 of the Arena software package, which handles all point cloud calculations. The cameras run at 100 frames per second, sending a 640×480 precision grayscale image to the software by USB. Rigid body orientation and position is streamed from Arena to the simulation environment.

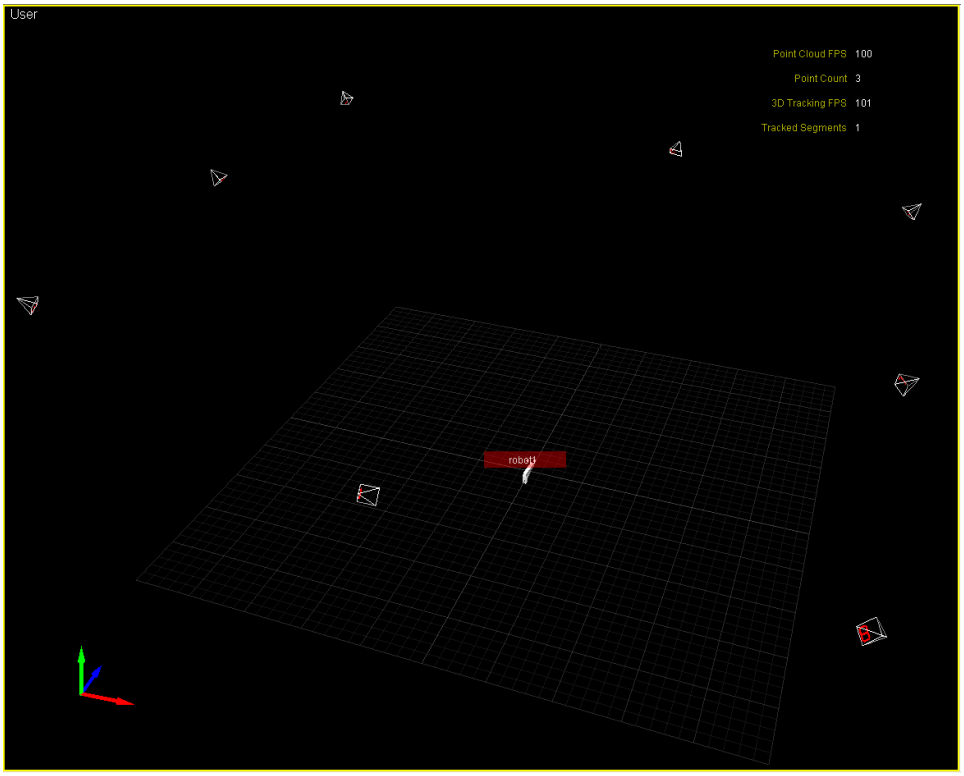


Figure 3.3: Screenshot from the motion capture software showing all cameras and the rigid robot body.

Alternatives Alternatives to motion capture equipment when dealing with robotics applications vary, depending on the need for accuracy. Using an accelerometer and a gyro to estimate position by integrating linear acceleration twice gives low accuracy, but may be improved by using kalman filters or other techniques. Many researchers have used onboard sensors like ultrasound, laser range finders or stereo vision systems to successfully estimate movement. Other alternatives include use of light, sound, or magnetic fields. Wireless signals like RFID or normal computer-based wireless networks can also be used. To get an accurate indoor positioning system, a hybrid algorithm combining different techniques should be used, and fusion and analysis of data from different sensors has grown into a big research field.

3.5 Simulation environment

The evolutionary simulation framework developed at the University of Oslo relies on PhysX for physics simulations, and ParadisEO for the evolutionary functions. It enables distributed simulations across the network, and has been used in several ER research projects [20, 35].

Physx The real-time physics engine PhysX[9] is released and maintained by Nvidia, one of the top two developers of graphic cards, along with AMD. PhysX and its source developer kit is free for both commercial and non-commercial use on windows, and is in use by a wide range of computer games and other projects requiring physics simulations. It allows developers to rely on a framework for physics simulations that enables simple high level coding to integrate high end physics simulations with hardware acceleration support into existing projects without particular knowledge in physics or simulations.

ParadisEO ParadisEO[10] is a free white-box object oriented metaheuristic framework implemented in C++. It also includes a variety of probability functions, and includes support for hybrid, parallel and distributed metaheuristic calculations.

ROBIN framework overview An overview of the framework can be seen in figure 3.4. When doing evolutionary runs, a set of randomized initial encodings are made. These generate a controller and a simulation model, which are given as specifications to the PhysX simulator. The simulator returns one or several fitness scores to the evolutionary framework, which uses ParadisEO to modify the encodings, and start the cycle again for new generations. A single initial encoding is the starting point for hardware runs, and this encoding generates a single hardware model with controller, which is sent as a specification to the hardware control. The controller then sends signals to the robot. The motion capture equipment sends back scores, which is used by the learning system to modify the controller of the hardware model. This is repeated for a number of evaluations defined by the learning algorithm used. Every red box contains model-specific code, and needs to be changed when new robots and controllers are added.

[9] http://www.nvidia.com/object/physx_faq.html
[10] http://paradiseo.gforge.inria.fr/

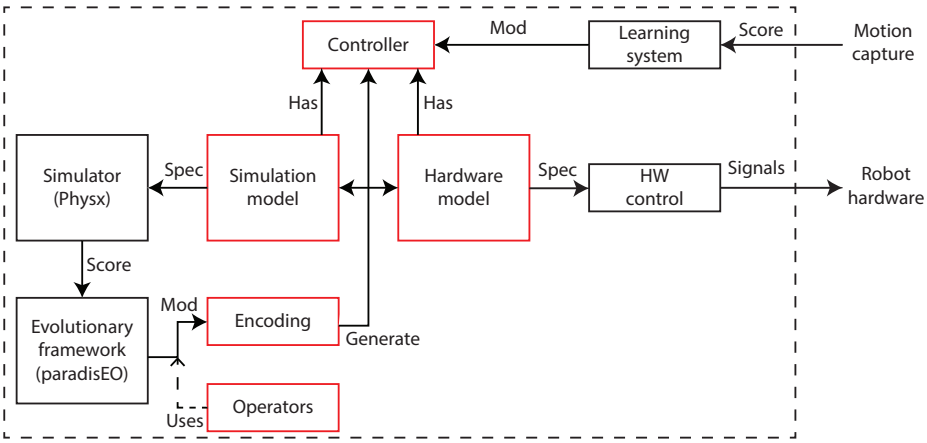


Figure 3.4: Flow chart of the evolutionary simulation environment. All custom parts are colored red.

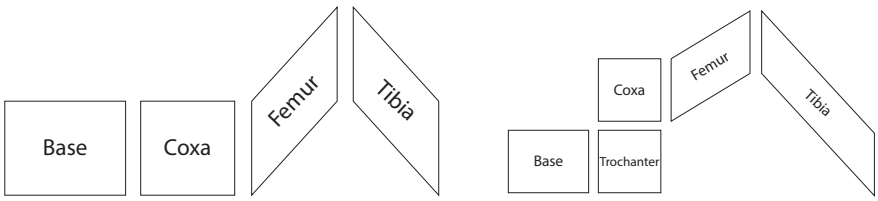
Chapter 4

Implementation

This chapter gives an overview of the work done to develop the evolutionary system, including the parameterizable model, the physical robots with gaits, the simulation environment, and the evolutionary framework. Some section might contain minor experiments and results, but these are small, and considered a part of the implementation phase of the work. The main goal of the design was to make a legged robot suitable for evolutionary experiments involving both morphology and control. This requires parameterization of all critical dimensions of the robot. To be adaptable to a wide range of objectives and environments, a six legged design was chosen. This enables high manoeuvrability and ability to walk in harsh terrain, and gives good lifting capability, balance, and speed, as opposed to two or four legged robots, or their wheeled counterparts. The chapter begins with the design of the robot platform. Manual choice of parameters for the initial model is then shown, before gait design and the evolutionary setup is discussed.

4.1 Design of robot platform

Even though the final robot models are generated using the evolutionary system, the parameterizable model has to be made manually and parameters has to be chosen for a manually designed version to be used as a reference when testing performance. Two different leg configurations were made to give a wider field of application for the robot. All link names are taken from biology, specifically arachnid legs. Three degrees of freedom (DOF) is considered standard for legged robots, and is shown in figure 4.1a. Another DOF is added in the leg showed in figure 4.1b. Only the 3DOF legs are used in this thesis, and the axis system is defined in section 4.3, where gait and kinematics are calculated. The first sketch of the robot can be seen figure 4.2, which includes servo IDs for the optional front legs.



(a) Drawing of the 3DOF leg.

(b) Drawing of the 4DOF leg, typically used in front.

Figure 4.1: Drawings of the two leg configurations.

4.1.1 Choice of evolvable parameters

The parametric robot model needs to be customizable, while still containing as few parameters as possible, to reduce computational requirements, driven by the curse of dimensionality. Lengths for the femur and tibia of each leg were chosen as parameters. Servo sizes and angle limits were assumed to be static (but could be changeable if several servo models were available), and minimum lengths for all parameters were chosen to be as small as possible while still having a robot with feasible real world relevance. Maximum lengths were chosen as the largest parts still able to fit on a typical professional 3D printer tray, to enable other researchers or collaborators to more easily reproduce the model and research. Lengths for all femurs and tibia resulted in 16 parameters.

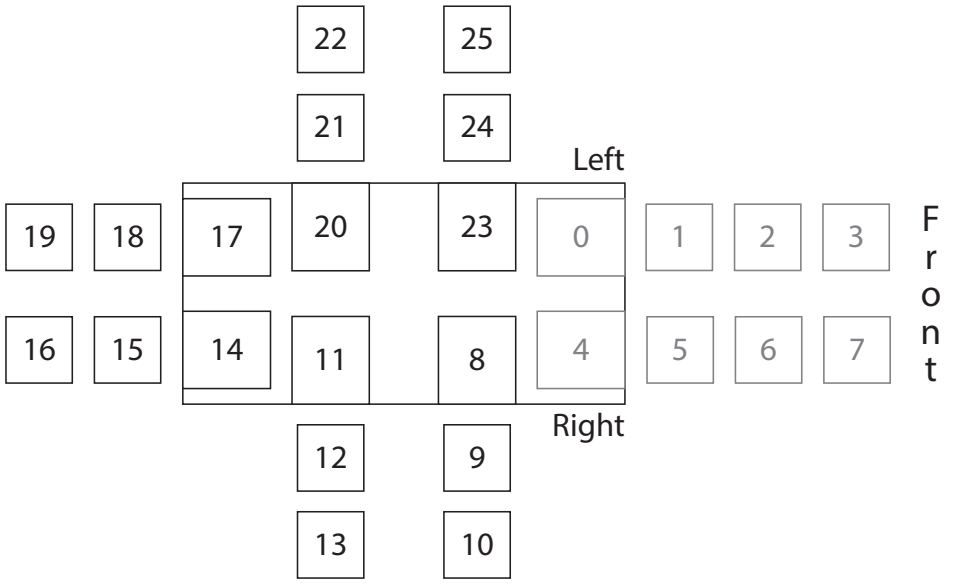


Figure 4.2: Drawing showing servo placement and IDs. Front legs are colored gray as they are not used in this thesis.

Base To enable a wider range of robot morphologies and robot configurations, the base plate was also made customizable. It hosts eight servos, and

the parameters chosen were distances between the centers of certain servo horns, seen in figure 4.3. Six of the side servos were chosen to hold legs, while the front servos were not added in this thesis, due to the increased complexity of simulations and computational requirements. This parameterization results in 5 different parameters, hopefully making the base plate adaptable to a wide range of different scenarios and morphologies. An additional limit was set on the length parameters to simplify the creation of the 3D model and the calculation of the weight. This states that servos 9, 12, 18, 21 and 24 are always at the sides, or in other words, that parameter B1 defines the width of the base plate, and B0 can only occupy the given space. This results in the limit given in equation (4.1).

$$B0 + 21.5mm \times 2 < B1 \tag{4.1}$$

Base servo orientation 21.5mm in equation 4.1 comes from the fact that servos 1 and 5 are rotated 90 degrees compared to the servos on each side, and the centre of the servo horns on the side is therefore further from the edge of the base plate than the front servos. More parameters could be added to allow for arbitrary rotation of servos on the ends of the plate, but this configuration was chosen to limit the number of parameters.

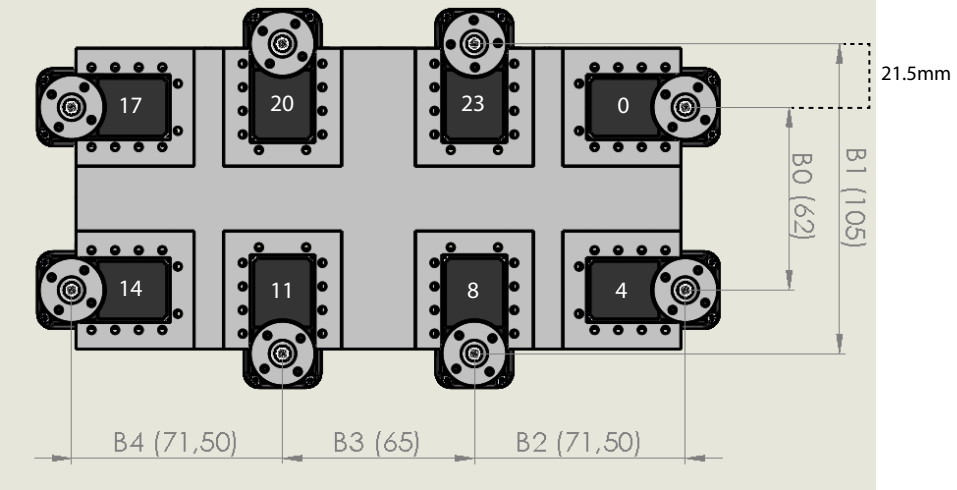


Figure 4.3: Drawing of the base, showing all parameters.

Movement sectors One disadvantage to locking the servo orientations is that the evolution has less control of the available space for each servo movement. Most traditional gaits require legs to alternate between backward and forward positions in opposite phases, so leg paths cannot intersect, or else the legs would crash. This has typically been solved by assigning each leg its own *sector*, as seen in figure 4.4. The sectors can then be used to calculate angle limits to restrain each leg from exiting its own sector. Sectors of maximum size can be achieved with a circular robot, but a circular robot is often considered sub-optimal, perhaps evident from the

lack of large circular animals in nature. Because so many six- or eight legged walking robots feature a configuration of legs on two sides, it was decided to lock the servo orientation to keep the solution space minimal. Individual sectors can still be modified by different gaits, but this will simply move or resize the sectors relative to each other, not increase total movement space across all sectors.

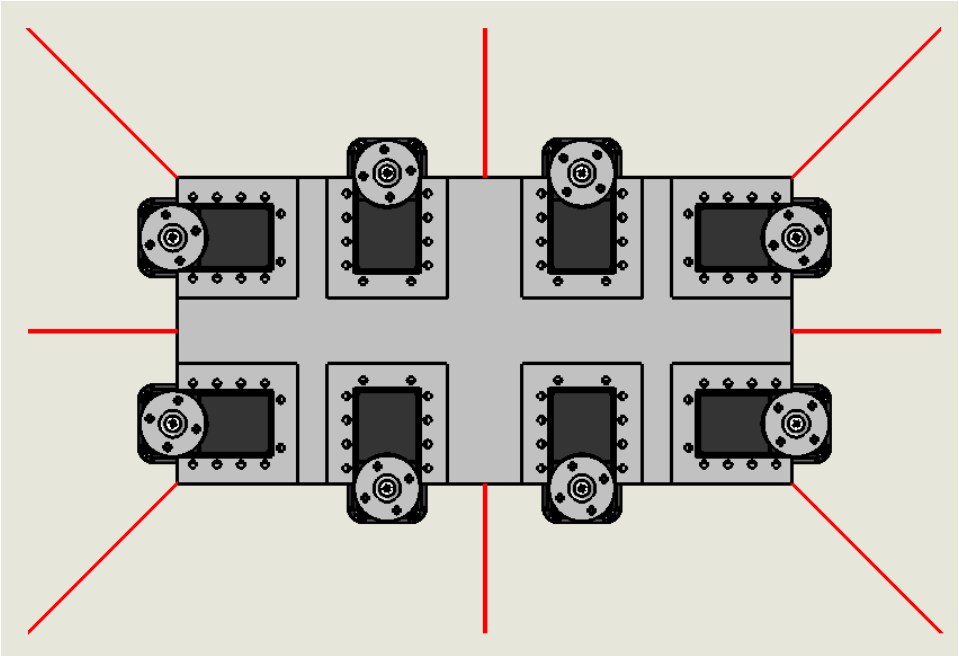


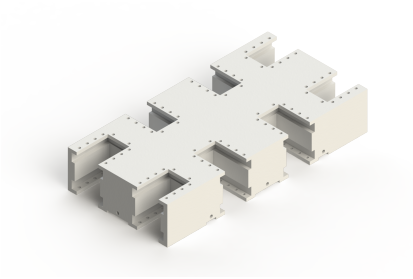
Figure 4.4: Drawing of a possible sector diagram showing the available servo movement space for each servo.

4.1.2 Design of parts

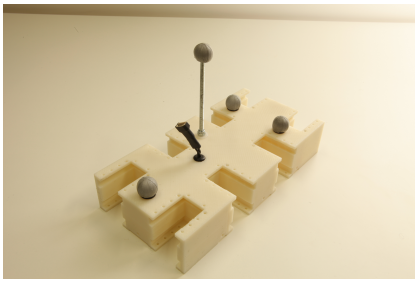
Each part was designed in Solidworks 2013 using simple 2.5D techniques (see section 3.1 for a description of 2.5D design). The design was done from the center of the robot and outwards, starting with the base, then coxa, femur, tibia, and then the trochanter used only in the front.

Base plate

The base plate was designed to be lightweight, and to provide a stable foundation for the 8 servos. This allows six legs plus two servos holding sensors or tools, or a total of eight legs, giving maximum strength and agility to the robot, allowing traversal of rough terrain or carriage of heavier loads.



(a) Rendered image from Solidworks



(b) Image of the printed base, with motion capture markers and cable gland.

Figure 4.5: Images of the 3d printed robot base.

Coxa

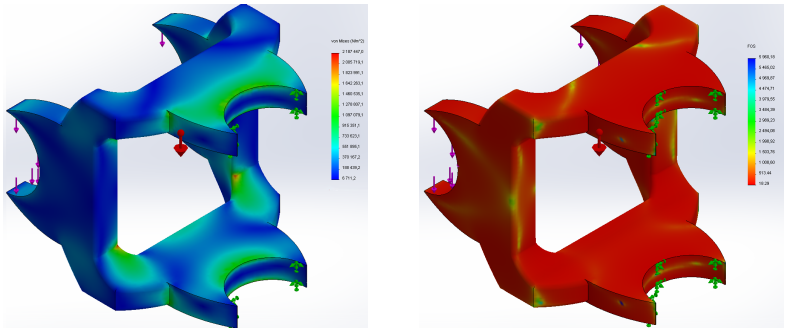
The coxa serves as the interface between the trochanter and the femur in the front legs, or the base and the femur in the other six legs. It serves as a link between two servo horns, and gives a rotation of 90 degrees, as seen in figure 4.6. A model featuring the required rotation, while not restricting servo movement or adding too much material and weight, can be challenging. It was, for this reason, decided to test the design by doing a FEM simulation.

(a) Rendered image from Solidworks.

(b) Image of the printed coxa.

Figure 4.6: Images of the 3d printed coxa.

Coxa FEM simulation A FEM simulation was done on the Coxa to ensure that the right strength and rigidity was achieved, as seen in figure 4.7. The study was done with 3kgf force on one end, and a static fixture on the other. It resulted in a factor of safety of 18.3 and fairly even Von Mises stress, indicating a structurally strong and stable design.



(a) Von mises stress for the coxa. Blue areas show the lowest level of stress, while red areas show the highest levels.

(b) Factor of safety for the coxa. Red areas show the lowest FOS, while green and blue show the highest.

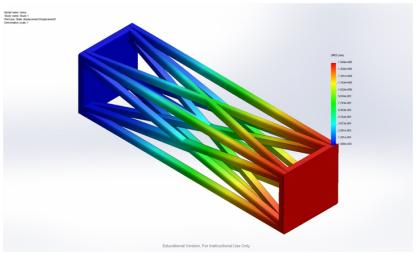
Figure 4.7: FEM simulation results for the coxa.

Femur

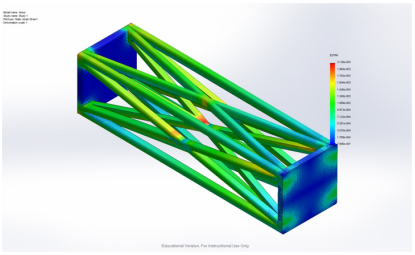
Several different configurations were tested during the design of the femur. A simple rectangular cuboid or a circular cylinder would suffice mechanically, but the overall aim of the design was to minimize weight by using as little material as possible. This also contributes to a lower cost and less print time, encouraging the use of this robotic framework for projects with lower budgets. Designs were based on truss techniques - commonly used when constructing bridges, electricity pylons, buildings, or other structures. The different designs were compared by weight, displacement, strain, and factor of safety. A custom material was used in Solidworks, derived from the standard material ABS. This is not an exact match to the non-uniform material of the parts printed by the 3D printer. It is, however, a good measure for relative performance. The test was done with a femur middle length of 150mm, standard gravity, and 5kgf applied at one end of the femur.

Model	Weight	Displacement	Strain	FOS
Reference	43.97g	1.55mm	**0.00214**	**6.91**
Pratt truss	39.52g	1.58mm	0.00566	2.08
Whipple truss	**38.54g**	1.66mm	0.00434	3.93
Brown truss	42.12g	**1.41mm**	0.00369	4.42

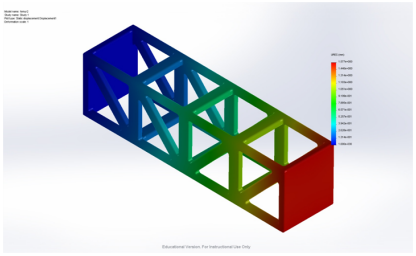
Table 4.1: Performance of the different femur designs.



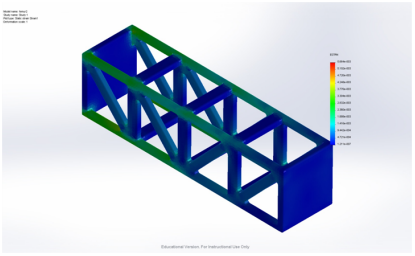
(a) Displacement of reference model



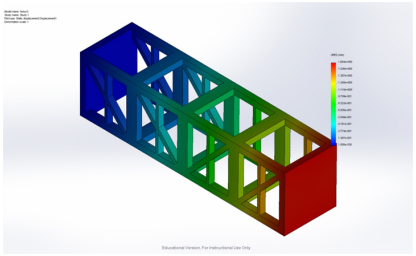
(b) Equivalent strain of reference model



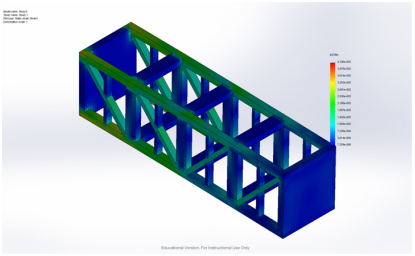
(c) Displacement of Pratt truss design



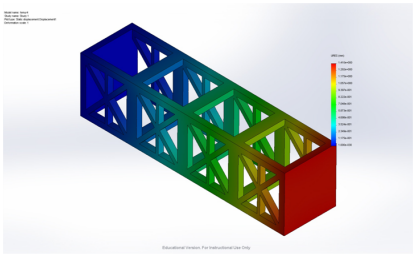
(d) Equivalent strain of Pratt truss design



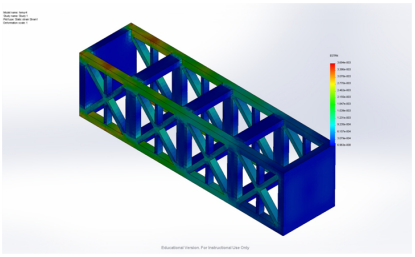
(e) Displacement of whipple truss design



(f) Equivalent strain of whipple truss design



(g) Displacement of Brown truss design



(h) Equivalent strain of Brown truss design

Figure 4.8: FEM simulation results

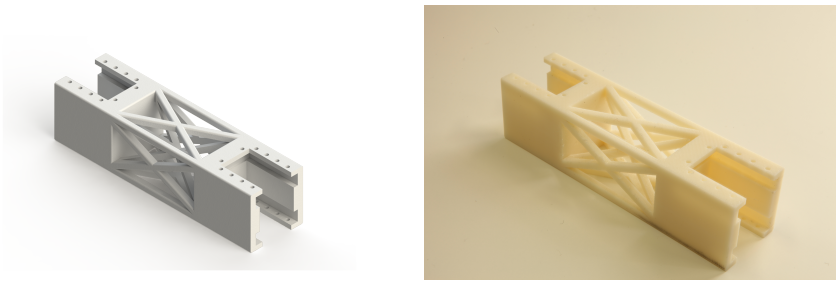
Reference model A hand designed model was made without using any common truss technique, to get a base to compare the traditional truss designs with. The design was drawn with simple engineering intuition, without any reference designs or truss design experience. The performance can be seen at the top of table 4.1. Figure 4.8a shows the displacement graph, and figure 4.8b the strain graph.

Pratt truss There is a variety of different truss techniques, and one of the most commonly used techniques is the Pratt truss. Figure 4.8c shows the displacement graph, while figure 4.8d shows the strain graph. Compared to the custom model, weight is lower; displacement is higher, while factor of safety is far less. The relatively low decrease in weight (less than 5g, or 40g for the whole robot) does not justify the increase of 165% in equivalent strain and a loss of factor of safety in the area of 70%.

Whipple truss An extension of the Pratt truss is adding vertical nodes through the diagonal members. The implementation seen in figures 4.8e and 4.8f shows that the implementation is not a perfect Whipple truss, since it is missing an angled member at each end. This should, however, not affect the performance to a high degree. This design features an even lower weight than the Pratt truss design, but there is still an unacceptable increase in equivalent strain and loss of factor of safety, even though strain improved when compared to the Pratt truss.

Brown truss The Brown truss was the last truss type chosen for testing, and can be seen in figures 4.8g and 4.8h. This truss type is known for its economical use of materials, and should therefore give a light model when compared to structural support. The weight loss is less significant than the other two models, but the displacement is better than both the reference model and the other truss types. Equivalent strain and factor of safety are better than the other two truss designs, but still considerably worse than the reference model.

Final choice of femur design A more detailed analysis could have been done if the differences between the models were greater. A continued analysis would have included redesigning the models so they all had the same weight, and thereby enabling a more precise and direct comparison. Since the weight difference between the best and worst was less than the weight of a single servo given six legs, the hand designed model was chosen for the high factor of safety, and was made parameterizable.



(a) Rendered image of the femur (b) Image of the printed femur

Figure 4.9: Images of the 3d printed femur design.

Tibia

The tibia is at the end of the leg, and connects to servo hubs. It was designed to be as light weight as possible, and feature three holes in each horizontal beam for mounting of tools, sensors, or other objects.

(a) Rendered image from Solidworks

(b) Image of the printed part

Figure 4.10: Images of the 3d printed tibia.

Trochanter

The trochanter was designed to serve as the interface between the servo horn at the base, and the coxa. It enables rotation of the legs so they can serve as grippers or dynamic tool holders, and can be seen in figure 4.11. The trochanter is only needed for the optional front legs, and is not used in this thesis. It was, however, designed to enable others to use it for future research and experiments.

(a) Rendered image from Solidworks.

(b) Image of the printed trochanter.

Figure 4.11: Images of the 3d printed trochanter.

4.1.3 Parametric weight

Since the simulator relies on weights for the parameterized parts, functions for the weight of both femur and tibia have to be determined. If the simulator used high resolution 3d models, a simple mass density would likely suffice, but since the simulator works on a simplified robot model for speed and efficiency, an individual mass function call is used to apply the correct

Part	Weight in real world ($\pm$0.1g)	Weight in Solidworks
AX12 servo	54.7g	-
AX18 servo	57.8g	-
Servo holder	19.1g	22.81g
Trochanter	29.5g	38.93g
Coxa	24.2g	32.91g

Table 4.2: Weights of all static robot parts

weight to all parts. Static weight is defined in table 4.2, while parametric weight is calculated for the femur, tibia, and base. (See chapter C in the appendix for a table of weights of the different parts at various phases of the project.)

Femur

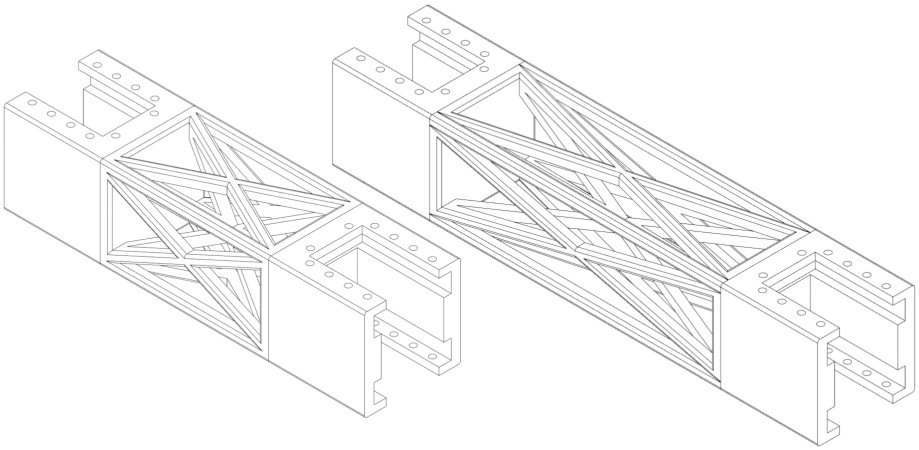


Figure 4.12: Drawing of the femur, with a total length of 150mm to the left and 200mm to the right.

Printing two different lengths and then interpolating the weights would have been the preferable method of making a weight function, but since printing is costly and takes time, weights were interpolated using Solidworks, and then multiplied with a Solidworks to physical world constant, calculated from one printed sample. This results in a low accuracy weight, but higher accuracy would be impossible without also including the tool path generating software for the 3D printer, as parts are non-uniform due to the printing process. Weights given in table 4.3 are the weights of the middle part of the femur recorded in Solidworks. Only the middle part of the model scales with length, while servo holders stay static, as seen in figure 4.12.

Middle length	Total length	Solidworks weight	Printed weight
70mm	150mm	29.00g	-
120mm	200mm	38.32g	-
150mm	230mm	43.97g	35.9g
174mm	254mm	48.5g	-

Table 4.3: Weights of the femur from Solidworks.

After using linear regression, the following function is generated:

$$\text{femur\_weight}_{SW} = 0.187g/mm \times x + 15.858g/mm$$

The printed femur has a total length of 230mm, which results in the following calculation to get the weight of the middle part:

$$\begin{aligned} &74.1\text{g} - 2 \times \text{servoholders\_weight} \\ =&74.1\text{g} - 2 \times 19.1\text{g} \\ =&35.9\text{g} \end{aligned}$$

This leads to a weight in the real world (rw) to weight in Solidworks (sw) constant of:

$$\begin{aligned} C_{rw/sw} &= \frac{35.9g}{43.97g} \\ &= 0.81647 \end{aligned}$$

This in turn makes a final function for the weight of the femur:

$$C_{rw/sw} \times ((\text{length} - 80mm) \times 0.187g/mm + 15.858g/mm) + 2 \times 19.1g$$

Tibia

The tibia is the second of the two parametric parts used in the legs. As with the femur, only one printed sample was available at time of calculation. Only the outer part of the tibia is scaled when the length is increased, while the inner part stays constant at 66mm long, as seen in figure 4.13. Calculations are done identically to the femur calculations.

$$\text{tibia\_weight}_{solidworks} = 0.136g/mm \times x + 13.58g/mm$$

$$\begin{aligned} C_{rw/sw} &= \frac{40.8g}{46.22g} \\ &= 0.88273 \end{aligned}$$

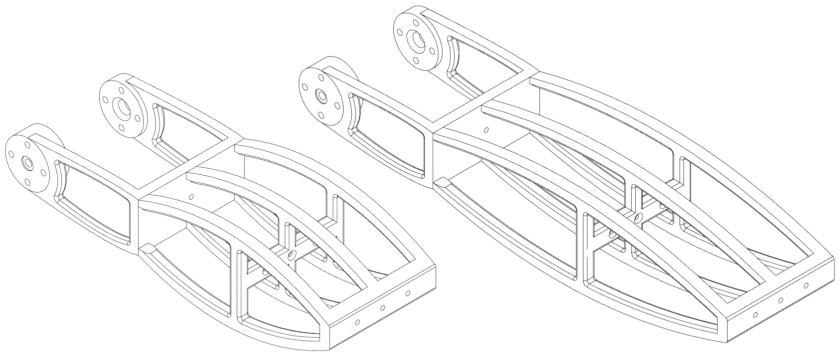


Figure 4.13: Drawing of the tibia, with a total length of 150mm to the left and 200mm to the right.

End length	Total length	Solidworks weight	Printed weight
84mm	150mm	33.96g	-
134mm	200mm	40.81g	-
174mm	240mm	46.22g	40.8g
188mm	254mm	48.10g	-

Table 4.4: Weights of the tibia from Solidworks.

The calculations result in a final function for the weight of the femur:

$$C_{rw/sw} \times ((length - 55mm) \times 0.136g/mm + 13.58g/mm)$$

Base plate

As can be seen in figure 4.14, the base plate is of a fairly simple construction, with 8 servo holders and a thin plastic construction covering the empty spaces. All five parameters are also seen. Five dimensional regression analysis is not a viable technique due to a high requirement of data points, but since the construction is so simple, a calculation of the total area is a feasible method, and should be very accurate, even when only given a single printed sample.

$$\begin{aligned} A_{\text{blue}} &= (B1 - 83mm) \times (B2 + B3 + B4 + 69mm) \\ A_{\text{red}} &= 152mm \times (B2 + B3 + B4) - 24776mm^2 \\ A_{\text{total}} &= (B1 - 0.083m) \times (B2 + B3 + B4 + 0.069m) \\ &\quad + 0.152m \times (B2 + B3 + B4) - 0.024776m^2 \end{aligned} \tag{4.2}$$

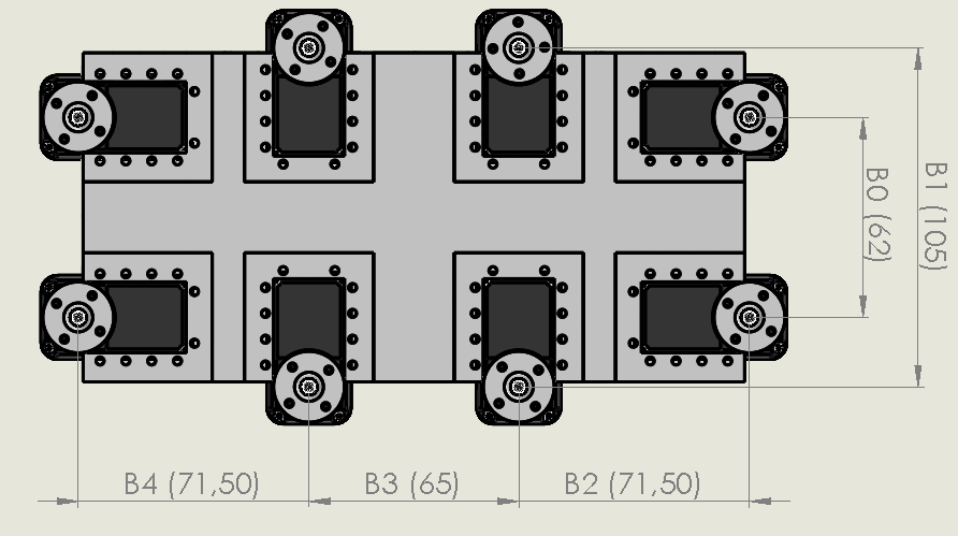


Figure 4.14: Drawing of the base plate, with lengths from robot1.

For the initial printed base model, the function yields the following result:

$$\begin{aligned} &22mm \times 277mm + 152mm \times 208mm - 24776mm^2 \\ =&6094mm^2 + 31616mm^2 - 24776mm^2 \\ =&12934mm^2 = 0.012934m^2 \end{aligned}$$

Printed weight is 177.4g, but that includes 8 servo holders. Total weight is given by:

$$177.4g - (8 \times \text{weight}_{\text{servo holder}}) = 177.4g - 8 \times 19.1g = 24.6g$$

This means the constant for weight to area can be calculated by the function below (4.3). This constant is used together with the function for area (4.2) and the weight of the servo holders, to define the total weight of the base plate. See figure 4.15 for the colored zones.

$$24.6g/0.012939m^2 = 1901.23g/m^2 \approx 1.9kg/m^2 \tag{4.3}$$

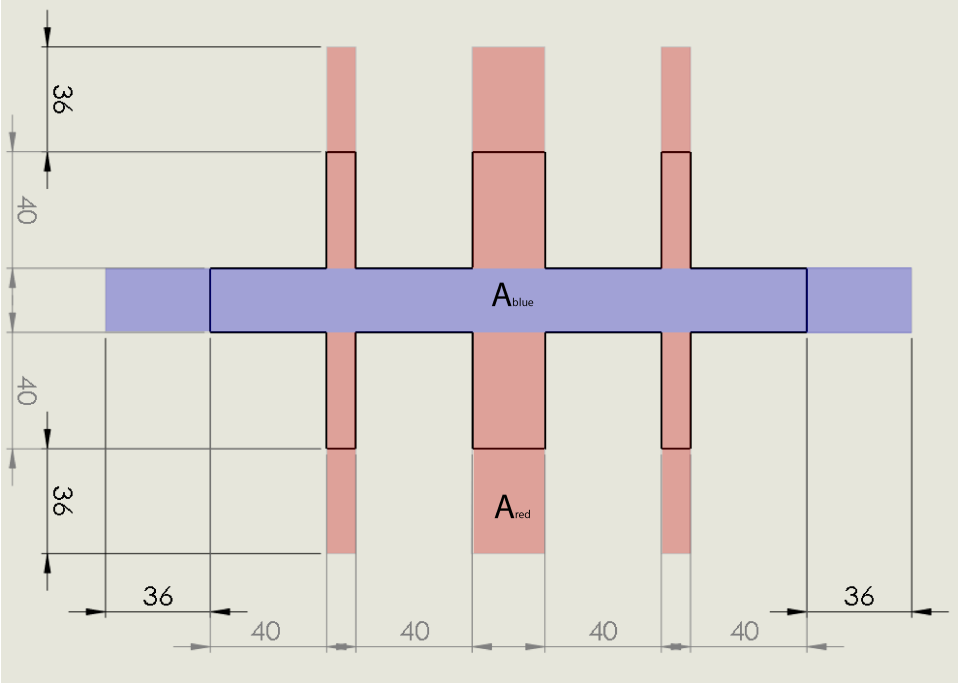
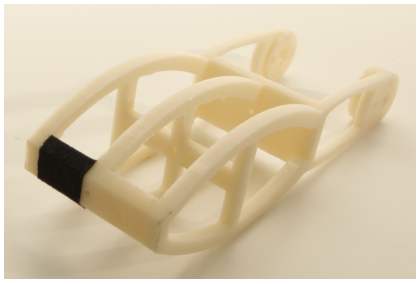


Figure 4.15: Drawing of the base plate, with marked area calculation zones.

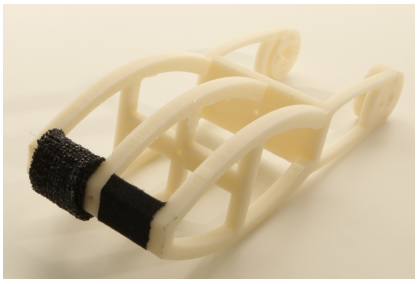
4.1.4 Increasing friction

Grip tape The ABS plastic used for the parts of the robot has a low friction on both the floor at the robotics lab, and the carpets used in the motion capture lab at UiO. A low friction is typically harder to simulate than a high friction, and there are several different approaches to increasing friction. The first and quickest way used to increase friction, was to apply both friction tape and grip tape to the end of the legs. Friction tape is a thin tape used for hockey sticks and other sporting goods, while grip tape is a thicker, coarser tape, which is used in hobby projects and DIY. The robot legs were designed to make it easy to attach different tools or objects, so the tape was applied without any effort, as seen in figure 4.16. Friction tape did not offer much difference on the floor, but showed promising results on the carpet. The thick grip tape increased friction significantly on both surfaces, though not as much as originally hoped.

Testing setup Since different solutions to the friction problem were to be tested, a friction testing setup was made. The setup was only used for comparing different friction increasing designs, so an absolute friction force measurement was not required. The main goal of the test was therefore a relative score, with good precision, while accuracy was a secondary objective. Dynamixel servos were the motors readily available, and allow digital control of the torque, which was used actively throughout the testing. A simple carriage was designed to hold two batteries for weight, and had the same hole placement as the robot legs to enable similar attachment. A pulley was designed, printed, and attached to the servo, pulling the rope



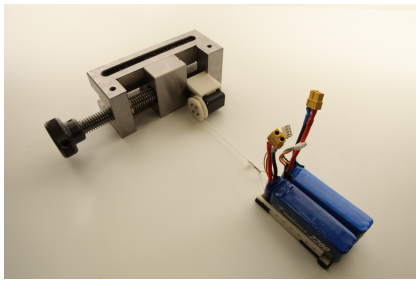
(a) An image showing the friction tape applied on one side of the tip of a tibia, with bare abs plastic on the other side.



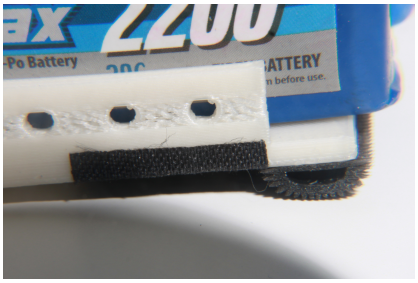
(b) An image showing the grip tape applied on one side of the tip of a tibia, with friction tape on the other side.

Figure 4.16: Images of the two types of friction increasing tape used.

attached to the carriage, as seen in figure 4.17a. The Arbotix microcontroller[1] was programmed to run the motor system. Logging of the position of the carriage was done through a Processing[2] sketch. This allowed for a gradual increase of the torque of the motor, and recording of the movement of the system at the same time. The test was done up to five times for each solution, and returned a relative fitness score. When running the actual test, the vice was holding the servo to a table leg, while the carriage was pulled across the floor.



(a) An image showing the testing setup, including the carriage, servo, pulley and vice.



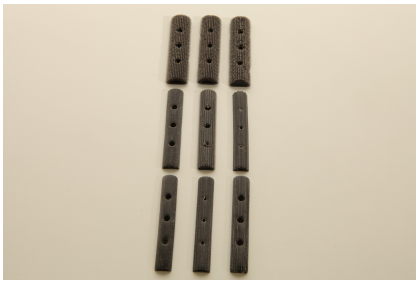
(b) An image showing a friction pad test up close.

Figure 4.17: Images of the friction pad testing setup.

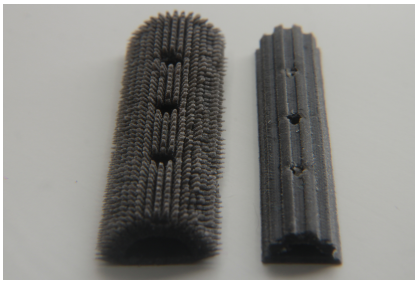
Friction pads The photopolymer 3D printer we have at the University of Oslo is able to print in a flexible material called Tango Plus[3], which offers much higher friction on both the floor of the university and the carpet in the motion capture lab, compared to untreated printed ABS plastic. Several different designs of friction pads were made and printed in pure Tango black plus, and a friction testing setup was made to test and compare the

[1] http://vanadiumlabs.com/arbotix.html
[2] http://processing.org/
[3] http://www.stratasys.com/materials/polyjet/rubber-like

performance of the different models and materials. This resulted in a friction pad that can be attached directly to the legs of the robot, as seen in figure 4.18. These showed a significant increase in friction, both on the floor and on the carpet. As seen from figure 4.17b, the friction pads were flexible enough to deform and give a higher area of contact for increased friction, much like the tires on a car.



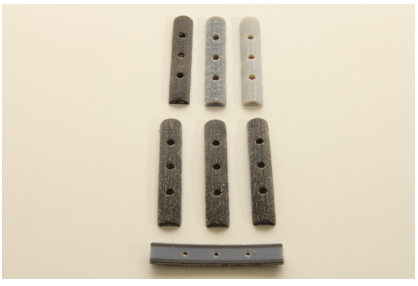
(a) An image showing nine of the different friction pads.



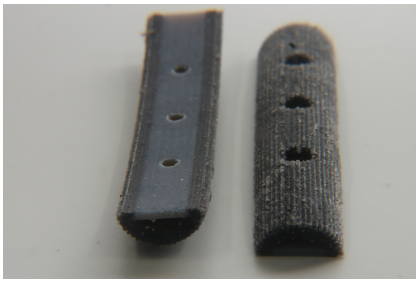
(b) A close up image of two friction pads, printed in Tango black plus, showing the details of the texture.

Figure 4.18: Images of the single material friction pads.

Multi material friction pads It became apparent after the first friction tests that the 100% Tango black plus material produced pads too weak for continuous use on the robot. While providing good friction, the soft model tore where the screws and washers attached it to the end of the robot legs. The 3D printer is able to print in different materials for parts of the model, and even using an arbitrary mix of different materials. New models were designed where the whole part was printed in a mix of hard and flexible materials, producing a single semi-flexible part. Parts with flexible domes and non-flexible back plate were also printed and tested.



(a) An image showing multi material friction pads. Three different material mixes are used on the three topmost pads, while the back of the multi material pads can be seen in the bottom of the image.



(b) A close up image showing how the hard back plate is attached to the flexible front, and the difference in the size of the holes on the dome and the back plate.

Figure 4.19: Images of the multi material friction pads.

Conclusion The single material friction pads of 100% Tango black plus performed best, but were too weak for use on the robot. The Multi material friction pads with a flexible dome and non-flexible back plate performed slightly worse on friction, but showed no signs of deterioration through five tests of each pad. Availability of the Tango black plus material did, however, stop the manufacture of a full set of friction pads. Grip tape was therefore used for the duration of the thesis, with friction tape below, to provide better grip to the ABS plastic parts.

4.2 Manual choice of parameters

After choosing evolvable parameters and designing the parts, values for the first iteration of the robot had to be chosen. No optimization or simulation was used, as this robot serves as the manually designed baseline for comparisons to evolved robots. Sizes were limited by the available 3D printer tray size, but also by the strength of the servos. Any static gait needs at least three legs in contact with the ground at all times. As long as the center of mass is within the triangle defined by the three legs, the robot will be in balance, and the gait will be specified as static. Any additional legs in contact with the ground extend the area of the triangle, and give a higher number of available static poses. Each servo controlling the femur is rated at a stall torque of 18.3kg/cm force. Given a static gait on a robot weighing about 3kg, each servo has to support at least 1kg each. This results in a maximum length between the inner servo of the femur to the end of the tibia of 18.3cm.

Calculation of forces for different lengths Given a maximum length between femur and tibia of 18.3cm, a minimum force required to stand and a maximum servo angle can be calculated. The calculations are done in chapter A.1 in the appendix, but the results have been shortened and presented in table 4.5. Sizes of 230mm were calculated first. It showed a minimum force required fairly close to the stall torque, so smaller sizes of 150mm were also tested. This resulted in a minimum torque required of approximately half the stall torque, and was chosen as initial lengths for the legs. Base parameters were chosen intuitively, giving a relatively small distance between all servos, and can be seen in table 4.6.

Femur	Tibia	Distance_{min}	$\text{Torque required}_{min}$	Angle_{max}
230mm	230mm	147mm	14.7kg$\times$cm	133°
150mm	150mm	94mm	9.4kg$\times$cm	105°

Table 4.5: Leg lengths and corresponding force requirements to stand on only 3 legs, given a robot of 3kg.

B0	B1	B2	B3	B4	Femur	Tibia
62mm	105mm	71.5mm	65mm	71.5mm	150mm	150mm

Table 4.6: Parameters chosen for the manually designed robot.

4.3 Gait design

Designing a gait from a path planning program by making use of the inverse kinematics of each leg was done to produce a manually designed gait that can serve as a base line when evaluating the performance of evolving morphology. The evolutionary setup, including simulation model, hardware control, and control system, was implemented to enable evolution of both morphology and control.

4.3.1 Forward kinematics

The forward kinematic matrix was derived using the Denavit-Hartenberg conventions, as described in section 2.2.3. The forward kinematics were calculated to establish the axis system for all links, to enable mathematical verification of the inverse kinematics, and to use for experimentation on the robot platform before path planning and inverse kinematics were implemented. The full calculation can be seen in section A.2 of the appendix.

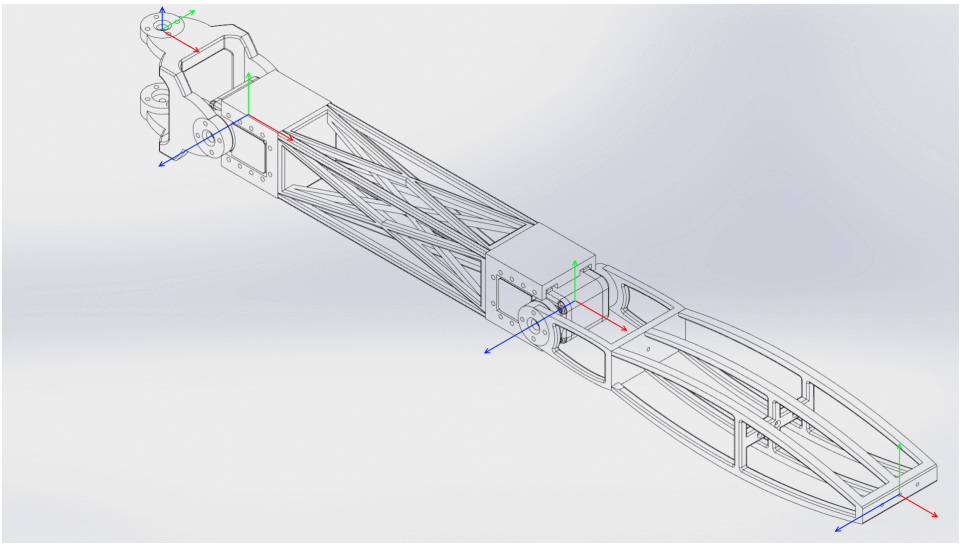
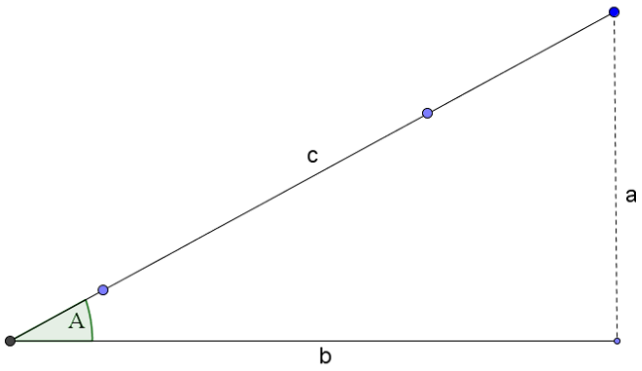


Figure 4.20: The Denavit-Hartenberg compatible coordinate system chosen for the robot legs. Axis colors are the standard (X, Y, Z) = (Red,Green,Blue).

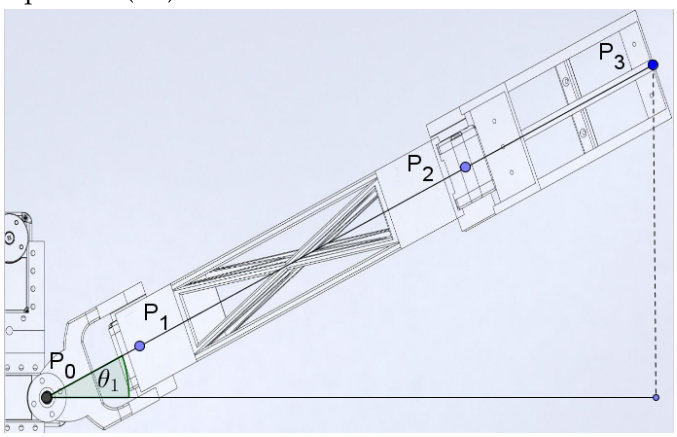
4.3.2 Inverse kinematics

Inverse kinematics are generally harder to calculate than forward kinematics, as there is no widely used method of calculation. The kinematics were derived for the robot using simple geometrical functions.

Calculation of θ_1 Calculating the coxa angle is the easiest, since the leg can be reduced to two dimensions by looking at the leg from the top. This makes the problem a simple trigonometry problem, solved by calculating the inverse tangent, as seen in the equations (4.4) and figures 4.21.



(a) This is the reduced problem of finding θ_1, solved by equations (4.4).



(b) A sketch of the inverse kinematics problem of finding θ_1, showing a robot leg from above. P_0 is the inner coxa hub center, and P_1 is the outer hub center. P_2 is the inner tibia hub center, and P_3 is the center of the end of the tibia, as seen in figure 4.20.

Figure 4.21: Drawings of the inverse kinematics problem of θ_1.

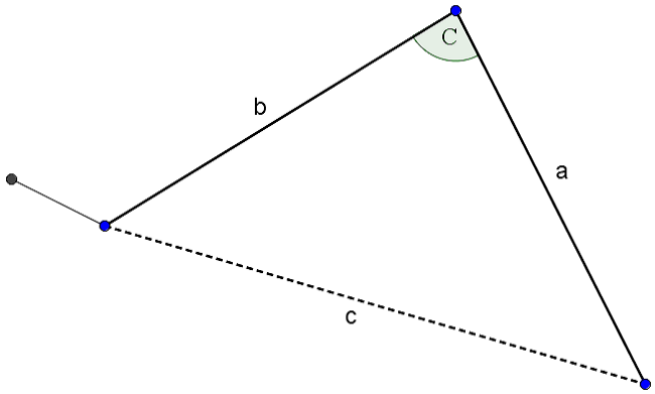
$$\begin{aligned} \tan(A) &= \frac{a}{b} \\ \tan(\theta_1) &= \frac{P_{3y}}{P_{3x}} \\ \theta_1 &= \tan^{-1}(\frac{P_{3y}}{P_{3x}}) \end{aligned} \tag{4.4}$$

Calculation of θ_3 The calculation of the two femur angles is slightly more complicated than the coxa angle. The geometry is seen from the side of the

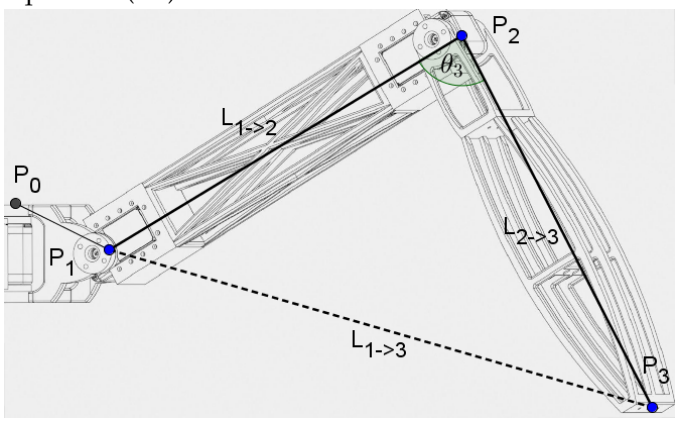
leg to simplify the problem, and is therefore considered two dimensional. Firstly, the three dimensional Euclidean distance $L_{1\to3}$ is calculated, and then used in the law of cosines to calculate the angle θ_3, as seen in equation (4.5) and figures 4.22.

$$L_{1\to3} = \sqrt{(L_{1\to2})^2 + (L_{2\to3})^2 - (L_{1\to3})^2} \tag{4.5}$$

$$\cos(C) = \frac{a^2 + b^2 - c^2}{2ab}$$
$$\cos(C) = \frac{(L_{2\to3})^2 + (L_{1\to2})^2 - (L_{1\to3})^2}{2 \times L_{1\to2} \times L_{2\to3}}$$
$$\theta_3 = \cos^{-1}\left(\frac{(L_{2\to3})^2 + (L_{1\to2})^2 - (L_{1\to3})^2}{2 \times L_{1\to2} \times L_{2\to3}}\right)$$



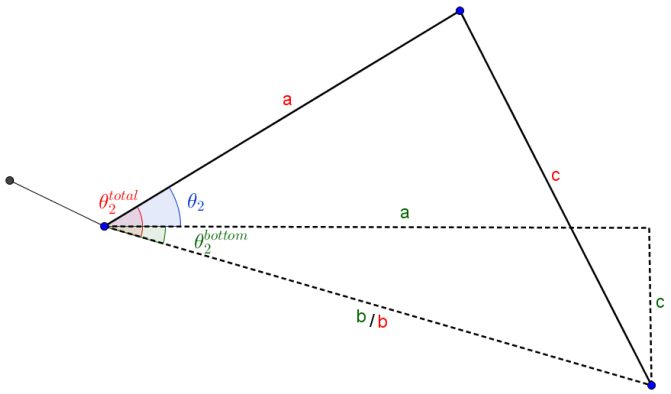
(a) This is the reduced problem of finding θ_3, solved by equations (4.5).



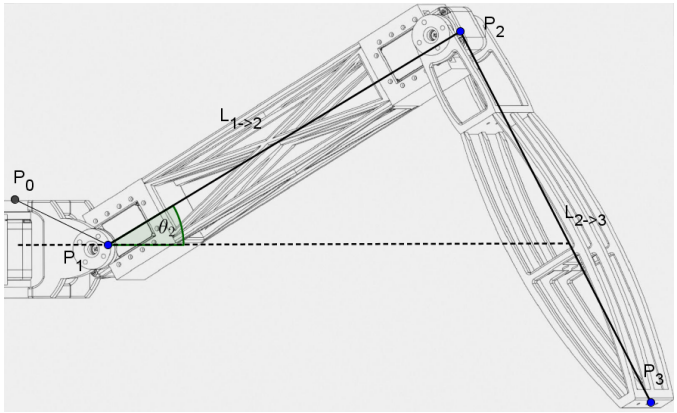
(b) A sketch of the inverse kinematics problem of finding θ_3, showing a robot leg from the front of the robot. Coordinate systems are as defined in 4.20.

Figure 4.22: Drawings of the inverse kinematics problem of θ_3.

Calculation of θ_2 The calculation of θ_2 is the most complicated of the three angles. Two angles are created to aid the calculation, θ_2^{bottom} and θ_2^{total}. See figure 4.23a for the problem sketch. Calculating θ_2^{bottom} can be done using the inverse tangent function, since a right triangle is used (see equation (4.6)). θ_2^{total} is calculated using the law of cosines, as used when calculating θ_3 (see equation (4.7)). θ_2 is then given by equation (4.8), as seen in figures 4.23.



(a) This is the reduced problem of finding θ_2, solved by equations (4.8).



(b) A sketch of the inverse kinematics problem of finding θ_2, showing a robot leg from the front of the robot. Coordinate systems are as defined in 4.20.

Figure 4.23: Drawings of the inverse kinematics problem of θ_2.

$$\begin{aligned}
&\tan(\theta_2^{bottom}) = \frac{c}{a} \\
&a = \sqrt{(P_{3x} - P_{1x})^2 + (P_{3y} - p_{1y})^2} \\
&\tan(\theta_2^{bottom}) = \frac{P_{1z} - P_{3z}}{\sqrt{(P_{3x} - P_{1x})^2 + (P_{3y} - p_{1y})^2}} \\
&\theta_2^{bottom} = \tan^{-1}(\frac{P_{1z} - P_{3z}}{\sqrt{(P_{3x} - P_{1x})^2 + (P_{3y} - p_{1y})^2}})
\end{aligned} \tag{4.6}$$

$$\begin{aligned}
&\cos(\theta_2^{total}) = \frac{a^2 + b^2 - c^2}{2 \times a \times b} \\
&\cos(\theta_2^{total}) = \frac{L_{1\to2}^2 + L_{1\to3}^2 - L_{2\to3}^2}{2 \times L_{1\to2} \times L_{1\to3}} \\
&\theta_2^{total} = \cos^{-1}(\frac{L_{1\to2}^2 + L_{1\to3}^2 - L_{2\to3}^2}{2 \times L_{1\to2} \times L_{1\to3}})
\end{aligned} \tag{4.7}$$

$$\theta_2 = \theta_2^{total} - \theta_2^{bottom} \tag{4.8}$$

4.3.3 Manual gait from inverse kinematics

A gait based on the inverse kinematics was calculated for the manually designed robot. The inverse kinematics were implemented in Matlab, and a script was written that generates a conventional tripod-gait[89]. The robot has three legs on the ground at all times, as seen in figure 4.24. This provides a triangle shaped base of support for the robot. The line of gravity is within these bases at all times, and ensures a stable gait. Parameters were added for defining leg height over the ground, stride length, leg starting positions, and a safety margin between legs. A forward skew was also added, since legs are mounted on the back and sides of the robot only. Different parameters were tested and optimized. Resulting parameters and code can be seen in section B.7 in the appendix.

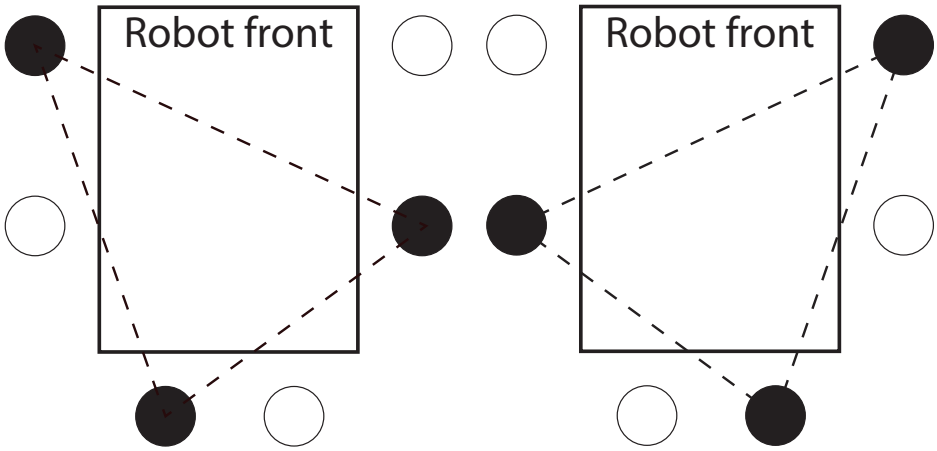


Figure 4.24: Graph showing the tripod gait. Legs on the ground are drawn with black circles, while legs in the air are white. The triangled base of support is shown with the dashed lines.

4.4 Evolutionary setup

Implementing the evolutionary setup is an important part of the thesis, and involves defining fitness functions, implementing robot control system, definition of genotype, and implementation of evolutionary operators and parameters.

4.4.1 Fitness functions

A fitness function is a function returning a fitness score by estimating how close a solution is to a given goal, and is described in more detail in section 2.3.7. The C++ code for all fitness functions can be seen in the appendix (section B.1). The main objective of the robot development was to make the robots move as fast as possible. Several fitness functions were, however, implemented as part of the thesis work.

Total weight A fitness function returning weight is used as a helper objective to ensure the evolutionary setup is able to generate a wide range of different morphologies to choose from. It could also, however, be used for evolving small robots to save on material costs or print time. Even though the morphology evolved in the robots used in this thesis have the same number of servos and legs, it was decided to develop the fitness function for general use. This includes looking at the total weight of the robot, including servos, other actuators, electronics, and other physical features. The function works by calling a robot-specific function in the simulation environment. This enables different robots to implement different weight functions. The robots in this thesis simply return the sum of weight of all parts from PhysX, as this is parametrized and includes all parts of the robot. A search is typically run with minimization of this function as a goal.

Forward movement The fitness objective rewarding speed was chosen to be the distance moved in the robot's forward direction at the end of the eight seconds of simulation. This was done as continuously deriving position to get speed, and then calculating the average speed throughout the simulation, is computationally expensive, and prone to rounding errors or noise. This does not explicitly favor straight gaits by punishing movement in other directions, although it implicitly does so by only rewarding positive movement along the forward axis for eight seconds.

Sideways movement A fitness function for sideways movement was also implemented, and is implemented in the same way as the forward movement function. The only difference is that this function rewards movement in the axis perpendicular to the length of the robot. Which way to be favored, left or right, can be changed by switching between maximization and minimization of this function.

Total movement A fitness function for total movement was also implemented, and measures the Euclidean distance travelled in the ground plane at the end of the eight seconds of simulation. This function can be minimized when no movement is wanted, as it is used when evolving turning gaits, or maximized, to enable evolution of gaits that walk in all directions.

Turning The last fitness function implemented in the work of this thesis, is a function designed for evolving turning gaits. It calculates the total counter clockwise(CCW) rotation of the robot throughout the evaluation. Since it returns the total turning angle CCW, a high negative value signifies a high performing clockwise turn. It can therefore be used with both minimization and maximization for evolving gaits that turn the robot either way.

4.4.2 Control system

The objective of the robot optimization is a high forward movement speed, which should be able to do with a simple repeating gait. It was decided to implement an open-ended controller, as closed-loop controllers are significantly more complex than open-loop versions, and require simulation of the inputs or feedback to the control system as well.

AmpPhase controller A controller that has previously proved effective in developing gait for six legged robots is a wave controller with amplitude and phase parameters [35, 63, 68]. Since natural gaits typically keep legs on the ground for some time, a simple wave controller based on sine waves is inefficient. This would mean constant movements, and low contact times between each leg and the ground. A solution to this was proposed by Koos et al. [63], and results in a control function that enables more stable gaits by restricting the movement to a smaller part of the gait period. The function consists of two parameters for each controlled joint, defining the control function γ (4.9), where t is time, α is amplitude, and ϕ is phase shift.

$$\gamma(t, \alpha, \phi) = \alpha \times tanh(4 \times sin(2 \times \pi \times (t + \phi))) \tag{4.9}$$

AmpOffPhase controller The AmpPhase controller generates a function that alternates between the center of the two angle limits of the servo, and this may reduce the range of robots this controller can be used for. To generalize, an offset can be added to enable the controller to control movements symmetrical around any given angle. A modified function κ is given (4.10), including an offset β. Any robot gait achievable by the modified control function could also be achieved with the original function, but the center would have to be set manually outside of the control functions or in hardware. The new modified function (4.10) includes an offset, and therefore enables the center of movement to be more easily included in evolutionary runs or other optimization techniques, as a feature of the control system. The two control systems can be seen in figure 4.25.

$$\kappa(t, \alpha, \phi, \beta) = \alpha \times tanh(4 \times sin(2 \times \pi \times (t + \phi))) + \beta \tag{4.10}$$

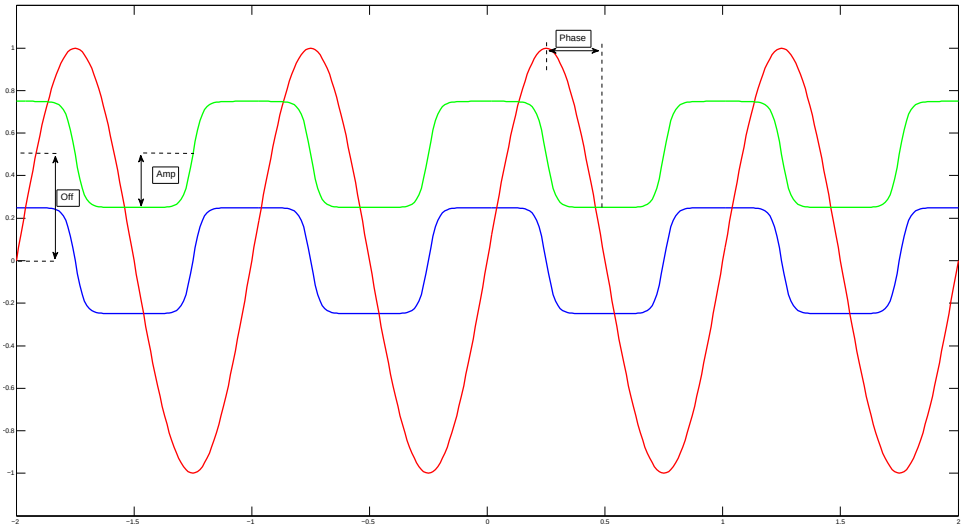


Figure 4.25: Graph showing the AmpPhase controller (4.9) in blue with amplitude $\alpha = 0.25$ and phase shift $\phi = 0.25$. The AmpOffPhase controller (4.10), in green, features the same amplitude and phase, but includes an offset β of 0.5. A reference sine is included in red, given as $\sin(t)$.

MinMaxPhase One of the difficulties of controlling a robot with the AmpOffPhase controller, is that limiting the control function to a given set of limits effectively, may be hard. Randomizing the parameters and then limiting them may result in a near-zero amplitude since the offset is near the limits, or amplitude above the available range of motion beyond the offset. If mutation pushed the movement beyond the limits, it is also unclear whether amplitude or offset should be affected, and how this should be done effectively without affecting the statistical distribution of the mutation.

I therefore propose yet another modified controller, that is easier to limit, and displays the same result as the original controller (4.9). Instead of amplitude and offset, two parameters ν and ω serve as the two angles of end movement. This simplifies limitation of actuation, since as long as the two parameters ν and ω are within the limits, the movement will be as well. To enable simple interaction with previous projects, controllers based on the original AmpPhase or AmpOffPhase can be converted to the MinMaxPhase controller by applying the functions given in equation (4.11). A comparison of the control functions can be seen in figure 4.26. The functions can be used to convert between the new amplitudes and offsets and the two limit parameters, or the complete function χ (4.12) can be implemented directly in the control system.

$$\begin{aligned} \alpha &= \frac{(\nu - \omega)}{2} \\ \beta &= \frac{(\nu + \omega)}{2} \end{aligned} \tag{4.11}$$

$$\chi(t, \nu, \omega, \phi) = \frac{(\nu - \omega)}{2} \times tanh(4 \times sin(2 \times \pi \times (t + \phi))) + \frac{(\nu + \omega)}{2} \tag{4.12}$$

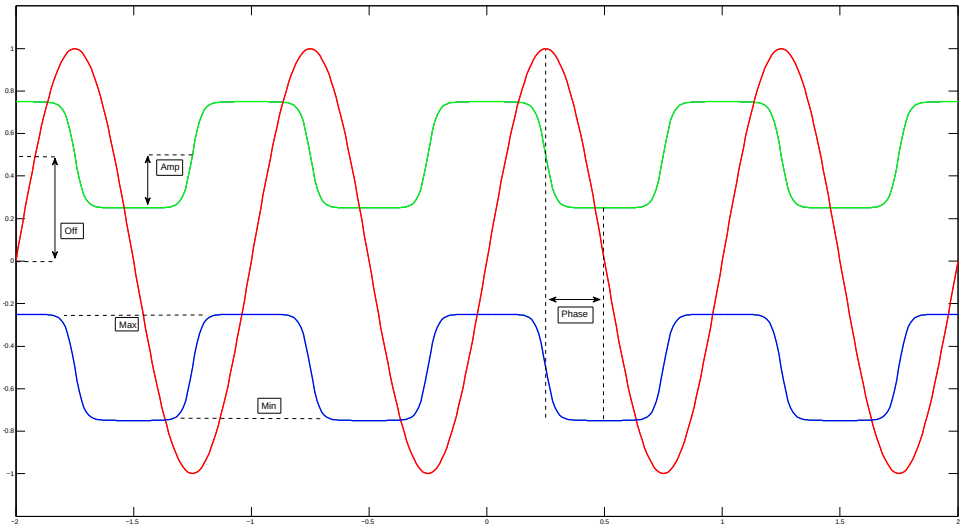


Figure 4.26: Graph showing the MaxMinPhase controller compared to the AmpOffPhase controller. The AmpOffPhase signal is the same as figure 4.25, while the MaxMinPhase has been generated with $\nu = -0.25$ and $\omega = -0.75$. They are able to produce identical output, but the parameter values needed are different. A reference sine is included in red, as in figure 4.25.

4.4.3 Reduction of the search space

The search space for the morphology alone is very large for this robot, even in terms of typical evolutionary robotics applications [90]. 6 legs with two parameters for lengths yield 12 parameters, along with 5 parameters

from the base. 18 servos, each requiring several control parameters, makes the search space for control much larger than for morphology. Table 4.7 shows the size of the search space, with min and max values for all parameters. The high number of dimensions makes any attempt at brute-force infeasible, and even evolutionary algorithms, which only test a fraction of the search space, may suffer. Large search spaces give a need for longer evolutionary runs, and yields a lower probability of finding near-optimal solutions.

Parameter	#	Lower	Upper
Tibia length	6	0.080	0.254
Femur length	6	0.080	0.254
Base length$_1$	1	0.05265	0.094
Base length$_2$	1	0.083	0.284
Base length$_3$	1	0.0615	0.254
Base length$_4$	1	0.05265	0.254
Base length$_5$	1	0.0615	0.254
Tibia movement	6	-2.49	2.49
Coxa movement$_1$	4	-0.81	1.64
Coxa movement$_2$	2	-1.64	1.64
Phases	18	$-\pi$	π

Table 4.7: Table showing search space for morphology at the top and control at the bottom. Coxa movement$_1$ is movement along the side of the robot (servos 8,11,20,23), while Coxa movement$_2$ is movement on the edges of the robot (servos 0,4,14,17)

Reduction of dimensions Most of the parameters are related to control of the legs, so this was the focus area of size reducing measures. Gaits rewarded for moving in straight lines have been shown to perform better when the morphology of the robots themselves are symmetric [65, 91], and all legs were therefore divided into pairs of servos, as seen in figure 4.27. The problem with symmetric gaits is that several symmetries can be used, and choosing one limits the number of resulting gaits greatly. This can be solved by using encodings or methods encouraging or imposing symmetry [92, 91, 34]. A simple side mirror symmetry was used for this robot, as the goal of the thesis was not to achieve the best results, but test a method of design. This increased the parameter requirement from three to four, but the four parameters now control two servos. This reduced the number of dimensions used for control from 54(18×3) to 36($18/2\times4$), a decrease by 1/3.

4.4.4 Design of robot in simulator

Introducing the robot to the simulation environment requires coding of a simplified three dimensional model. The Nvidia PhysX library works on solid bodies and links, and the whole morphology of the robot was designed

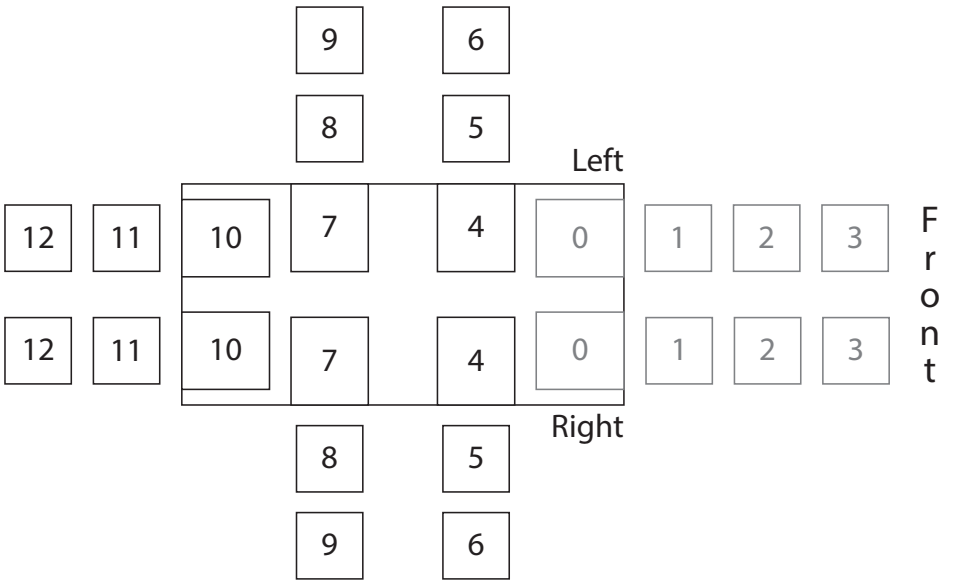
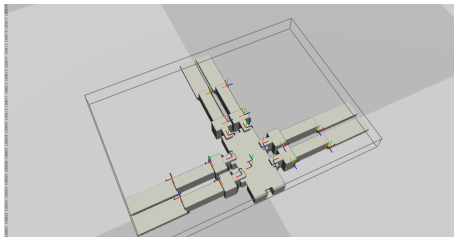
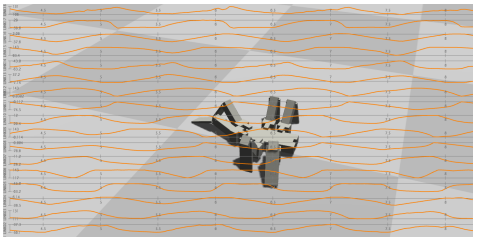


Figure 4.27: Drawing showing the servo pairs. Front legs are colored gray as they are not used in this thesis.

using simple three dimensional boxes. All boxes were given appropriate positions, sizes, and weights, and connected to each other by links. Servo forces for both types of servos were calculated, and applied to all links.



(a) Image of robot1 in simulator with bounding box and axes for each part.



(b) Image of robot1 walking in simulation, with an overlaid graph of joint movements.

Figure 4.28: Images of the simulation environment.

4.4.5 Operators and parameters

No functions for evolutionary operators were available in the evolutionary framework that was used, so all operator functions were made from scratch using C++. They can be seen in sections B.2 and B.3 in the appendix.

Mutation operators Non-uniform mutation was implemented for both discrete and non-discrete floats (see section 2.3.3 for description of float types and operators for evolutionary algorithms). Random reset mutation

was also implemented for both types of floats. The parameters for these operators were decided through experiments, seen in chapter 5.

Crossover operators There is a wide range of different crossover operators, and a small subset was implemented in code. The simplest was the uniform crossover, with custom probability and parent distribution. Simple-, single-, and whole arithmetic recombination were also implemented, along with a customized crossover operator named servo pair swap recombination. This crossover operator chooses a random pair of servos from each parent, and swaps all control signals for the chosen legs between the parents.

4.5 Result interpretation tools

Interpreting results from evolutionary runs can be a time consuming task. Tools for calculations of statistical values are essential, and Matlab was used for its statistical packages and since the integrated graphing possibilities would be beneficial for analysis of the results. The evolutionary framework produces plain text files with fitness values for all individuals of each generation of the run. A Matlab function to iterate through all text files and extract fitness information was implemented, and functions for comparing whole runs, or even groups of runs, were also written. This enables easy comparisons of both individual runs using the same parameters, and groups of runs with different evolutionary parameters. This is used in the experiments to decide between which parameters to use, and the effects of their change.

Pareto graph A function to graph the Pareto set from a group of runs was implemented. It features automatic classification of the Pareto front, has features for marking individual runs for analysis of sub-fronts, and shows which run the individuals came from, and which file it resulted in. This enables the user to see differences between individual runs, and gives insight into the Pareto front of each individual run in relation to the global Pareto front. An example of a graph generated by the script can be seen in figure 4.29.

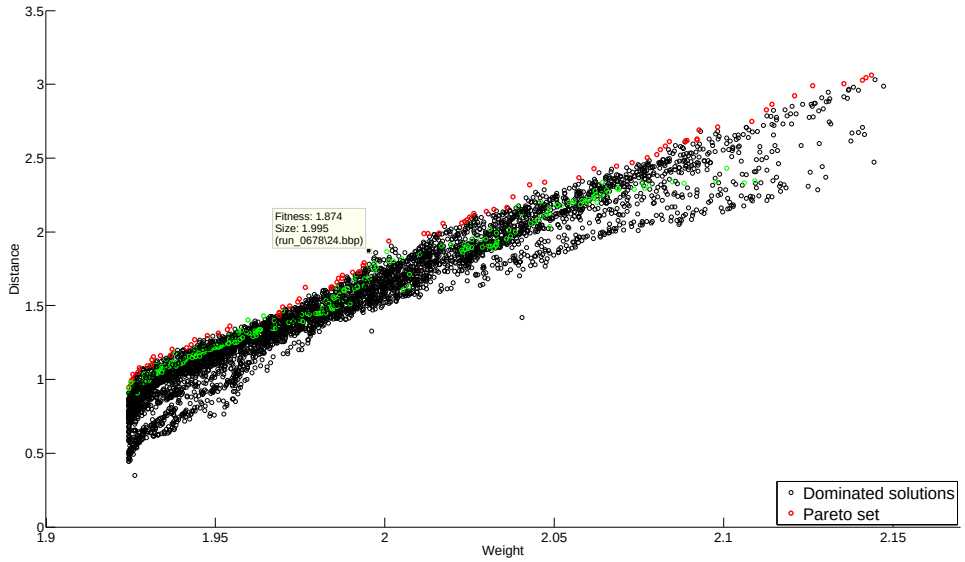


Figure 4.29: An example of a plot generated by the Pareto graph script. A single individual is chosen, with a datatip showing the file containing the individual. All other individuals from the same run are colored green.

Chapter 5

Evolutionary experiments and results

Experiments using the evolutionary framework were the first experiments done, and included improving parameters of the evolutionary algorithms used, the simulation environment, and the robot models themselves. The first section describes the parameter search done for evolving the control system, while the next section describes the parameter search for co-evolution of both control and morphology. The third section shows the experiments done on evolving the new morphologies, while new control systems for the chosen evolved morphologies are evolved in section four. The last sections evolves turning and sideways walk, as a test of the versatility of the evolutionary framework. The appendix contains table D.1 showing parameters of all runs, and table D.2 for the results. Distance walked was used as the main objective, while total weight served as a helper objective, which was not analyzed to the same degree as distance (see section 4.4.1 for more details on the fitness functions).

5.1 Parameter search for evolving control system

When applying evolutionary algorithms to a new problem, a variety of different parameters need to be chosen. There are several methods for parameter search for evolutionary algorithms, but no one method has gained enough use to be considered standard, and as such, a simple manual search was conducted (see section 2.3.5 for background on parameter search).

5.1.1 Uniform crossover experiments

Evolution of	**Optimization of**
Control	Uniform crossover value p
Objectives	**Other parameters**
Distance travelled	Non-uniform mutation probability: 1
	Non-uniform mutation step size: 0.3

The first crossover operator to be tested was chosen to be a uniform crossover operator, described in section 2.3.3. A p value of 0.5 is the most common, as this combines approximately half of each parent. Parameters less than 0.5 means the offspring consists of a bigger part of one of the parents, and a parameter close to 0 gives an offspring approximately equal to only one of the parents. As it was infeasible to run a search using only a crossover operator, the test was done with a static non-uniform mutation operator, to ensure a minimum of diversity in the population.

Run	Param.	Rank	$\text{Dist}_{\text{arithmetic median}}$	$\text{Dist}_{\text{mean}}$	$\text{Dist}_{\text{mean(max)}}$	Dist_{max}
180-216	0	2	0.9031m	0.9336m	1.5275m	1.9363m
217-252	0.5	3	0.7946m	0.8281m	1.4104m	1.7580m
253-274	0.25	3	0.7918m	0.8265m	1.4618m	2.2075m
275-291	0.1	4	0.7919m	0.8239m	1.3416m	1.4631m
292-295	0	1	0.9146m	0.9445m	1.5343m	1.6628m
296-304	0.05	4	0.7870m	0.8201m	1.3586m	1.6104m
305-318	0.02	3	0.7981m	0.8306m	1.3606m	1.4751m

Table 5.1: Results from uniform crossover experiments showing that the reference runs performed statistically significantly better than all parameters tested.

Results As seen from table 5.1, all parameters tested resulted in significantly worse average performance than the reference runs, given by the Wilcoxon rank sum tests with significance levels 0.01. Runs 292-295 were made to ensure no changes in the simulator code had affected the performance negatively, as the results were considerably lower than without recombination. This was shown not to be the case, and the experiment was completed with 5 different parameters being tested against no crossover.

Analysis Since no parameter was found where uniform crossover performed better than no recombination, it was decided to pursue a mutation only evolutionary setup. Testing other types of crossover operators might have given better results, but the low performance of this relatively commonly used operator indicates a low reward for the added exploration offered by recombination, which does not justify the added time and computational requirements of testing other crossover operators. An interesting feature of the results is the increase in max distance travelled of parameter 0.25, even though the mean of all max values is lower than the runs without recombination. This might only be caused by the low number of runs, but may indicate that recombination is better suited for an algorithm where peak performance was the goal, while average performance is highest without the operator. Due to higher statistical significance on average performance, and the time and computational constraints in this thesis, average performance is given higher priority in this thesis.

5.1.2 Non-uniform mutation experiments

Evolution of	**Parameter tuning of**
Control	Non-uniform mutation step size
Objectives	**Other parameters**
Distance travelled	Non-uniform mutation probability: 1

When it was decided that only mutation were to be used, the non-uniform mutation step size used before had to be optimized (see section 2.3.3 for an overview of non-uniform mutation and step size). The size was kept at 0.3 throughout the testing of recombination, so values both above and below this was tested.

Run	Step size	Rank	$\text{Dist}_{\text{median}}$	$\text{Dist}_{\text{arithmetic mean}}$	$\text{Dist}_{\text{mean(max)}}$	Dist_{max}
180-216	0.3	5	0.9031m	0.9336m	1.5275m	1.9363m
319-330	0.15	4	1.2785m	1.2942m	1.8727m	2.0576m
331-337	0.45	6	0.8134m	0.8486m	1.3966m	1.6027m
338-357	0.6	7	0.7569m	0.7908m	1.3123m	1.5659m
358-377	0.05	2	2.0297m	2.0755m	2.3122m	2.9099m
378-397	0.1	3	1.6463m	1.6125m	2.0286m	2.2995m
398-417	0.025	1	2.0204m	2.0943m	2.2343m	2.8969m
418-437	0.038	2	1.9661m	2.0758m	2.2707m	2.9648m

Table 5.2: Table of results from experiments with different non-uniform mutation step sizes.

Results As seen in table 5.2, a mutation step size of 0.025 performed best, according to the Wilcoxon rank-sum test with a significance level of 0.01. The original value of 0.3 performed fifth, only beating two other values tested.

Analysis A small mutation step size results in a smaller search radius during the exploitation part of the evolutionary run, while a high size widens the search area. Mutation step sizes allowing long jumps in solution space can add an exploration effect to mutation, even though mutation is originally considered more exploiting in nature by many, as it typically features shorter jumps than recombination, given high diversity. Very small mutations can give artificially good results, by having populations centered on small local optima, driving the mean and median fitness values up. Without recombination or high mutations, though escaping these local optima is unlikely, and long runs have a tendency to stagnate early. Since both step sizes with rank two have the highest max distance (0.038 and 0.05), it was decided not to continue trying lower values, and instead use the highest ranked step size, 0.025. It seems to be a good tradeoff between the exploitative nature of small mutations, while still being high enough to escape local optima and guide the search to new parts of the solution space.

5.1.3 Random reset mutation experiments

Evolution of	**Parameter tuning of**
Control	Random reset probability
Objectives	**Other parameters**
Distance travelled	Non-uniform mutation probability: 1
	Non-uniform mutation step size: 0.025

Random reset mutation was tested as a way of increasing the degree of exploration in the search.

Run	Prob.	Generations	Rank	$\text{Dist}_{\text{median}}$	$\text{Dist}_{\text{arth. mean}}$	$\text{Dist}_{\text{mean(max)}}$	Dist_{max}
398-417	0	200	6	2.0204m	2.0943m	2.2343m	2.8969m
438-457	0.02	200	2	2.5711m	2.4695m	2.6454m	3.1241m
458-477	0.04	200	4	2.3266m	2.3048m	2.5192m	2.8275m
478-497	0.01	200	5	2.3032m	2.2506m	2.4258m	2.9616m
498-517	0.03	200	3	2.3246m	2.2982m	2.4977m	3.1748m
518-537	0.02	1024	1	2.8052m	2.6587m	2.7813m	3.0964m

Table 5.3: Table of results from experiments with different random reset probabilities. Dist. is short for the distance traveled. Note that simulations 518-537 were run for 1024 generations, while the others were run at 200 generations.

Results As seen in table 5.3, all probabilities tested gives an operator resulting in better individuals, according to the Wilcoxon rank-sum test with a significance level of 0.01. Different probabilities were tested, and a group of runs were conducted at the end of the experiment with 1024 generations. This was done to see if the heightened degree of exploration would extend well for more generations. Median and mean distances were improved with more generations, but max distance found was not.

Analysis The non-uniform mutation experiments resulted in a relatively low mutation step size, which, in addition to the lack of crossover, makes the search use a higher degree of exploitation than exploration. It was suspected that an operator increasing the degree of exploration in the search might improve the quality of solutions, and this was confirmed when all parameters tested resulted in an increase in solution quality. The improvement in median and mean from running the search for 1024 generations instead of 200 was significant. The max distance found was, however, not improved. This shows that 200 generations is a good tradeoff between quality and computational requirements, especially when the runs are only used for relative comparisons between different parameters. The increase in mean max distance does, however, suggest the need for more than 200 generations when doing final runs for control systems.

5.1.4 Discrete mutation experiments

Evolution of	**Parameter tuning of**
Control	Division $\text{size}_{\text{phase}}$
	Division $\text{size}_{\text{position}}$
Objectives	Non-uniform mutation probability
Distance travelled	Number of forced non-uniform mutations
	Random reset probability
Other parameters	

Because of the massive size of the solution space, any decrease in size of genotype should lessen the computational requirements of the search. Several features have been added to facilitate this, one of them being support for setting a specified division size of all solution parameters. Equation (5.1) was used for limiting the parameters, with a special case for zero step_size, which returned an unchanged parameter. This was used to allow disabling of the rounding function via the parameters. The division size was initially set to a low number, to try to not exclude too many solutions from the search or alter the continuity and roughness of the fitness landscape. Since addition of division sizes to parameters affects the genotype so drastically, a separate non-uniform mutation for parameters with discrete values was used instead of the mutation operator used with non-discrete float values (see section 2.3.3 for a description of the operator). The new operator uses a recursive mutation probability, instead of a mutation step size. Separate division sizes were added for phases and positions.

$$\text{parameter}_{\text{discrete}} = \text{round}\left(\frac{\text{parameter}_{\text{original}}}{\text{division\_size}}\right) \times \text{division\_size} \tag{5.1}$$

Run	Div. $\text{size}_{\text{phase}}$	Div. size_{pos}	$\text{Prob}_{\text{non-uniform}}$	$\text{Num}_{\text{forced}}$	$\text{Prob}_{\text{reset}}$	$\text{Distance}_{\text{mean(max)}}$
438-457	0	0	0	0	0.02	2.2343m
525	0.001s	0.001^{rad}	0.15	3	0.02	0.8319m
526	0.001s	0.001^{rad}	0.15	5	0.05	0.7170m
527	0.001s	0.001^{rad}	0.25	5	0.05	1.1770m
528	0.0001s	0.0001^{rad}	0.5	5	0.05	1.2959m
529	0.01s	0.01^{rad}	0.1	5	0.05	1.5887m

Table 5.4: Table of results from experiments using discrete parameters.

Results After only a couple of generations of the first run featuring discrete parameters, the entire population was made up of only a few different individuals. The problem causing this was a fairly high probability of zero mutation, and therefore cloning of individuals. The experiment was therefore stopped, and several actions were taken to prevent this. A parameter called $\text{num}_{\text{forced}}$ was added to do a minimum number of mutations on each individual, heightening the probability of different parent and offspring drastically. Mutation probability was also increased while

division size was lowered, to increase the chance of mutation happening, while lowering the degree of change done by the operator on the individual. This enabled runs to go on for 200 generations without any individuals taking over all of the population. There were still many similar individuals after run 525, consisting of only three to five unique individuals in the population. $\text{Num}_{\text{forced}}$ was increased by two, and the probability of random reset mutation was increased substantially as well. This had some effect, but there were still several duplicates, and the performance of the search was far below what experienced without discrete parameters. An even larger increase of mutation probability for run 527 did not help much, and while a mutation probability of 50% removed all duplicates, performance was still far below non-discrete float representation, as seen in table 5.4. Another run with far courser division sizes was also tested, but did not show results deemed good enough to pursue through further testing.

Analysis The problem with adding discrete parameters is that the conversion to discrete values effectively removes a large number of solutions from the search space. Many of the optimal or near-optimal solutions might be removed, but even removal of sub-optimal solutions might affect the fitness landscape so much that the search is not able to traverse the solution space effectively [90]. Large irregularities or lack of continuity because of the removed solutions might make exploitation ineffective by not having the required connection between position in space and quality, and might leave exploration as the only working search tactic. Since computational power is fairly easily available, and the results are significantly worse than runs on non-discrete floats, use of discrete parameters was not pursued further.

5.2 Parameter search for evolving morphology

Evolving morphology in addition to the control system increases the search space immensely, but does not necessarily require new evolutionary parameters (see section 4.4.3 for more information on search space size). Since an optimal process is not required, a quick parameter search was performed to assure that feasible solutions were generated with the evolutionary parameters found during the parameter tuning of the control evolutionary runs. Individual mutation step sizes were used for mutation of lengths, given by (M) in the tables used throughout the section. (C) denotes control-specific parameters. A number of runs were first done to test the overall evolution of morphology, and several tweaks were done. Evolution of control system was again tested with the new settings before final parameter tuning. Some errors were also found and fixed, before the final morphology runs were conducted.

5.2.1 Initial parameter tuning

Evolution of	**Parameter tuning of**
Control	Static friction
Morphology	Dynamic friction
	Servo forces
Objectives	**Other parameters (control)**
Distance travelled	Random reset mutation probability (C): 0.02
Total weight	Non-uniform mutation probability: 1
	Non-uniform mutation step size (K): 0.025
	Non-uniform mutation step size (M): 0.0001

The first runs including evolution of morphology were runs 530-549. Mutation of base parameters, femur, and tibia lengths, were selected based on the performances of parameters on the control system, so a non-uniform mutation with a standard deviation of 0.001 was chosen as the initial parameter, corresponding to 1mm on the robot. Both dynamic and static friction were raised after the initial runs and a number of single evolutionary runs were then done to ensure whether the raised friction had the wanted effect.

Results A table of results is not shown, as qualitative comparisons were done exclusively during this experiment, in contrast to the other parameter tuning experiments, which included quantitative data on distance travelled. Both static and dynamic friction constants on the robot base were initially raised from 0.3 to 1 to encourage lifting of the base from the ground, along with a slight decrease in servo forces. The friction constants were later increased incrementally, to a final value of infinity. This showed a significant improvement in inspected individuals, which walked with the base off the ground, and with more intuitively correct servo power.

Analysis Setting the friction constants to infinity might seem excessive, but Nvidia's PhysX library clamps the value internally, so friction constants of infinity ensure the highest friction possible. Friction was not raised on the floor, as this would require new friction constants for the rest of the robot parts, and setting friction constants for the base material seemed to solve the problem. Setting the servo force lower than previously calculated produced more plausible gaits, and is also supported by the fact that the servos do not perform as well as their specifications [85].

5.2.2 Re-testing control system evolution

Evolution of	**Parameter tuning of**
Control	
Objectives	**Other parameters**
Distance travelled	Random reset mutation probability: 0.02
	Non-uniform mutation probability: 1
	Non-uniform mutation step size: 0.025

By changing such a large part of the simulation parameters, any result from previous runs was not comparable to new simulations done after the adjustment. A robot simulated with the old settings would not be affected by the same frictions or have the same motor forces as a robot in the new environment. After seeing morphology runs 554-559 producing feasible results, 20 runs of evolving the controller for robot1 was performed. This was done for two main reasons. The first and most important reason was to see that the changes done to the simulator to improve morphology runs did not affect control evolution negatively. The second reason was to get a new baseline score to compare it with both earlier scores and all scores for new morphologies.

Results The results from two groups of runs with similar parameters, but on different sides of the simulation adjustments are listed below in table 5.5. As seen in the table, the distance travelled in simulation went down considerably with the changes. When inspecting the performance of the individuals in the simulation viewer, gaits seemed more plausible, and had similar solutions as found before the simulation adjustment, only now having the base above the ground and more feasible motor forces.

	Run	$\text{Distance}_{\text{med}}$	$\text{Distance}_{\text{mean}}$	$\text{Distance}_{\text{mean max}}$	$\text{Distance}_{\text{max}}$
Old:	438-467	2.5711m	2.4695m	2.6454m	3.1241m
New:	560-579	1.6016m	1.5885m	1.7192m	2.1586m

Table 5.5: Table of results from evolutionary runs with the same parameters, but different simulation settings.

Analysis Lower distances were expected as servo forces were lowered while friction was increased, both affecting the distance travelled negatively. Seeing similar solutions before and after the adjustments was encouraging, and indicates that having the base on the ground does not affect the rest of the gait to a large degree. This also supports the notion that the near-optimal parameters found before are still performing well, and that there is no need for a new parameter tuning cycle for the control system evolution.

5.2.3 Final parameter tuning

Evolution of	**Parameter tuning of**
Control	Non-uniform mutation step size (M)
Morphology	Generations
Objectives	**Other parameters**
Distance travelled	Random reset mutation probability (C): 0.02
Total weight	Non-uniform mutation probability: 1
	Non-uniform mutation step size (C): 0.025

A group of morphology runs were conducted after the simulator adjustments and subsequent control system evolution to check the feasibility of generated solutions and see if there was a need for additional parameter tuning. A number of evolved morphologies and gaits were inspected, and all seemed feasible. The mutation step size for morphology still needed a tuning, and a new value of 0.01 was tested. Since the parameters were set for both control and morphology evolution during this experiment, runs to test for the right number of generations was also done as part of this experiment.

Results The new mutation value performed statistically significantly better than the value of 0.001, as shown in table 5.6, and was therefore chosen as the new mutation step size. Runs were then conducted for 256, 512 and 1024 generations, to see how more generations affect the quality of evolved individuals. The progress of the run with 1024 generations can be seen in figure 5.1. Not until 1024 generations do we start to see a flattening of the average best distance travelled. As figure 5.1 shows, 1024 generations gives an increase to the average best distance of about 33.4% over 256 generations, and an increase of 16.9% of 512 generations over 256. The gain of doubling the run time from 512 to 1024 only gives a fitness increase of approximately 14.1%.

Run	Mutation size	$\text{Dist}_{\text{median}}$	$\text{Dist}_{\text{arithmetic mean}}$	$\text{Dist}_{\text{mean(max)}}$	Dist_{max}
580-599	0.001m	1.0014m	1.0305m	1.7968m	2.7665m
600-619	0.01m	1.1540m	1.1887m	1.9338m	2.7257m

Table 5.6: Table of results from evolutionary runs with different morphology mutation step sizes.

Analysis Since the adjustment of the mutation step size resulted in such a low increase in mean and median distance travelled, it was decided to finish the parameter tuning, and select the better of the two. The results of the last runs showed that the improvement of running searches for more than 1024 generations would most likely be low, since both average distance and average best fitness were close to flat at 1024 generations. It was therefore chosen to use 1024 generations for final morphology runs.

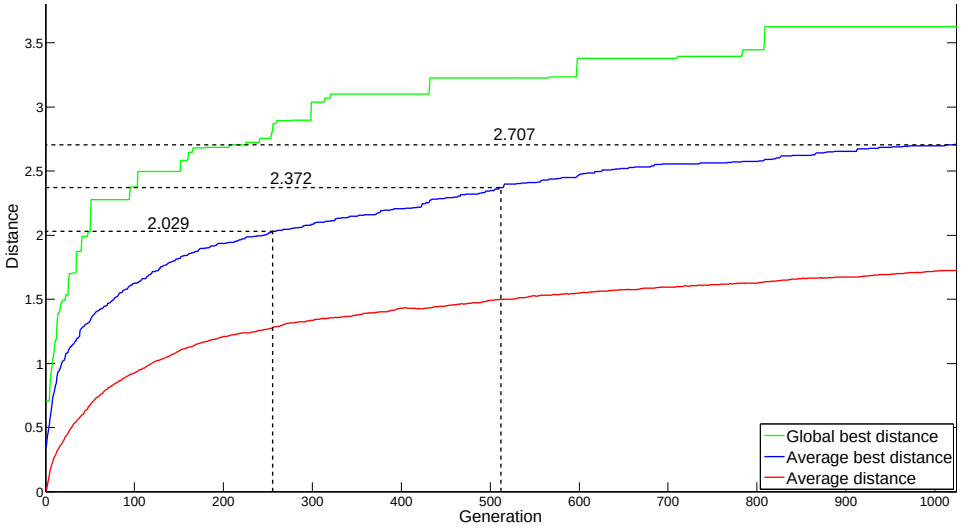


Figure 5.1: A graph showing the distance travelled over 1024 generations from runs 640-656, with markers for 200, 500, and 1024 generations.

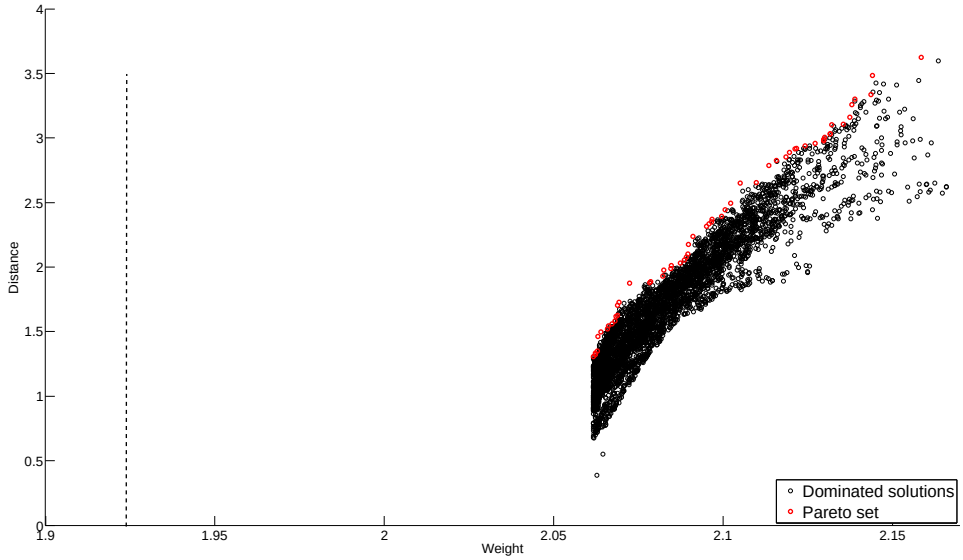
5.2.4 Verifying morphologies

Evolution of	**Parameter tuning of**
Control	
Morphology	
	Other parameters
Objectives	Random reset mutation probability (C): 0.02
Distance travelled	Non-uniform mutation probability: 1
Total weight	Non-uniform mutation step size (C): 0.025
	Non-uniform mutation step size (M): 0.01

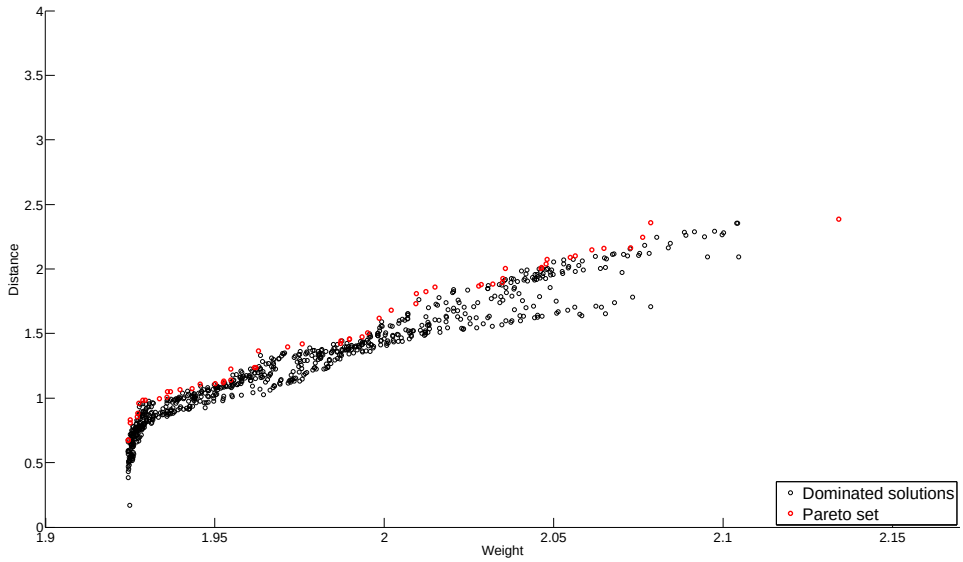
Runs 640-656 were done as a final test of morphology runs after the parameter tuning, and ran for 1024 generations. The top 10 individuals from each run were then inspected closely in the viewer, to ensure that everything worked correctly. Graphs for fitness development and the Pareto front were also inspected.

Results When inspecting the final Pareto front of runs 640-656, showing the distance travelled by solutions from all final generations, an error was found. As seen in figure 5.2a, all minimum sizes are along a vertical line, which is to be expected, but the line shows a weight of 2.06kg, which is well above the minimum weight of the robot. The inspected individuals along this line also showed different lengths of the femurs, even though they all indicated the same weight. The problem turned out to be an error in the simulation environment, where the weight was calculated, but not applied to the part. Weights in PhysX works by applying a separate weight or density to each shape in a joint, then applying the combined weight of all shapes to the link, thereby calculating weight distribution and converting to

the internal representation of weight. This had not been done to the femurs, and they therefore had the same constant weight regardless of physical size. Three evolutionary runs were done with only 512 generations to ensure that the change worked. The search now found individuals from the current lowest weight and upwards, as seen in figure 5.2b, and generated a number of feasible solutions.



(a) The Pareto front of runs 640-656, showing errors in the morphology code. The dashed line shows the minimum weight of the robot.



(b) The Pareto front of runs 657-659, showing a more plausible distribution of weights.

Analysis Since the weight has a minimum optimal value, a clear line of individuals can be seen in both Pareto fronts. The wrong placement of the first Pareto front line indicated the error found, but a weight difference of only about 100g from lightest to heaviest robot also seemed unlikely.

Run	$Dist_{med}$	$Dist_{arithmetic\ mean}$	$Dist_{mean(max)}$	$Dist_{max}$
640-656	1.6778m	1.7272m	2.7012m	3.6236m
657-659	1.2264m	1.2710m	2.2030m	2.3871m

Table 5.7: Table of results comparing evolutionary runs before and after fixing a bug in the femur parametric weight function.

The new Pareto front seemed more plausible, and showed a wide range of different weights and speeds. It also follows a fairly linear line, though the low number of runs and generations makes it impossible to get any statistically significant conclusions.

5.3 Evolving new morphologies

Evolution of	**Parameter tuning of**
Control	
Morphology	
	Other parameters
Objectives	Random reset mutation probability (C): 0.02
Distance travelled	Non-uniform mutation probability: 1
Total weight	Non-uniform mutation step size (C): 0.025
	Non-uniform mutation step size (M): 0.01

After getting good results in runs 657-659, the main morphology runs were started. 20 runs of 1024 generations of 256 individuals were conducted with settings chosen from earlier experiments.

Results As seen in figure 5.3, only small increases in the highest distance travelled were achieved after around generation 400. The average distance travelled does, however, still show a relatively high improvement. The Pareto front for the morphology runs is fairly linear, as seen in figure 5.4

Analysis The average distance is still improving at the end of the run, which shows that longer runs could potentially still be worth the increase in computational time. An even higher linearity than experienced in the Pareto front would likely be achieved if more runs had been done. The linear shape makes automatically choosing a solution hard, due to the high number of Pareto-optimal individuals. This does, however, show that the evolutionary algorithm used is able to produce a wide and varied subset of morphologies. Since the evolutionary run resulted in such a high number of Pareto optimal solutions, it was decided that manual selection would be best. An automatic choice of the best robot is be beneficial for a number of scenarios, but this type of selection is not the aim of this thesis.

Selection of robots It was decided to randomly select three robots smaller than the manually designed robot (referred to as robot1), and three heavier

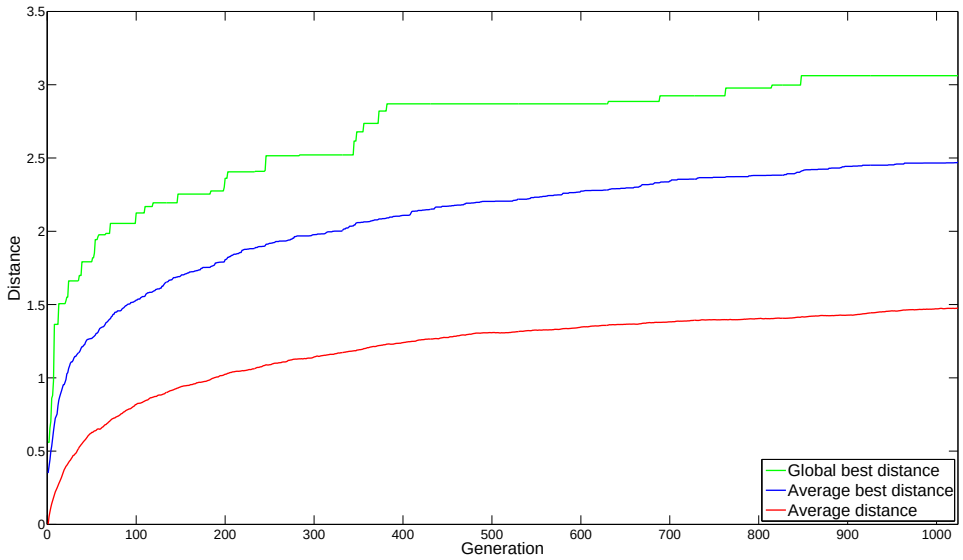


Figure 5.3: The distance travelled for final morphology runs 660-679. Highest fitness across all runs is shown in green, while the blue line shows the average of the maximum fitness for all runs. Average distance is seen in red.

Robot	Weight	Distance travelled
2	1.97647kg	1.623m
1	2.04037kg	-
3	2.06179kg	2.429m

Table 5.8: Table showing characteristics of the two chosen robots. Distance was not included for robot1, as it was not a result of a comparable evolutionary run.

robots, from the Pareto front. These were then inspected, and the one with the qualitatively highest chance of success was chosen for real life testing. The smaller robot (referred to as robot2) was chosen from the three randomly selected individuals, and seems to walk with a tripod-like gait, though this may change when new gaits are evolved. The weight is slightly lower than robot1, as seen from table 5.8. A heavier robot (referred to as robot2) was also selected. This robot intuitively seemed too large for the servos selected, but was still chosen for its interesting morphology. It featured long front legs, with small side legs being able to fit under the front legs when they intersected. This configuration overcomes the classical restrictions posed by the servo sector model (see section 4.1.1 for a description of the servo sector model). This configuration has not been seen in previously designed six legged robots or robotics related scientific research.

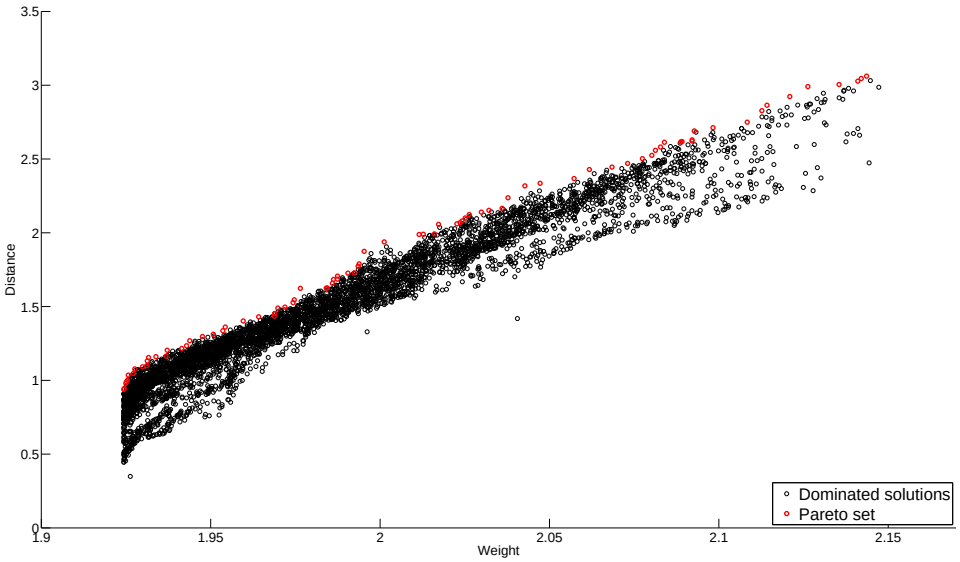
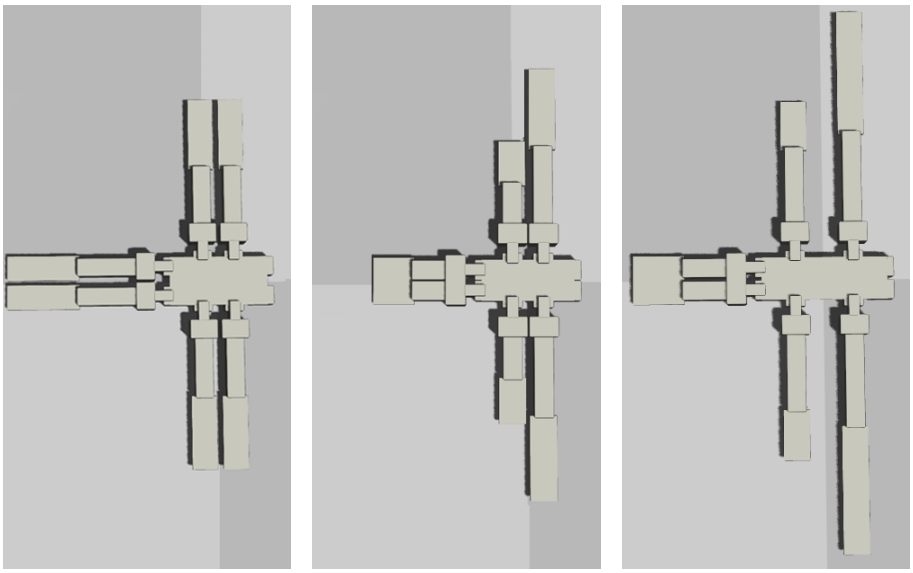


Figure 5.4: The Pareto front of final morphology runs 660-679, showing a nearly linear Pareto front.



(a) Robot 1 (reference) (b) Robot 2 (small) (c) Robot 3 (large)

Figure 5.5: Images of all three robots in the simulator. Robot 1 is the manually designed robot, while the other two are the selected evolved morphologies.

5.4 Evolving new control systems

Evolution of	**Other parameters**
Control	Random reset mutation probability (C): 0.02
	Non-uniform mutation probability: 1
Objectives	Non-uniform mutation step size (C): 0.025
Distance travelled	Non-uniform mutation step size (M): 0.01
Parameter tuning of	

Robot1, the manually designed robot seen in figure 5.7, was not part of the runs evolving morphology. New runs evolving control alone on this robot was done to give a base line for comparisons to the evolved robots. It was decided to also evolve control systems for the two evolved robots, to ensure a fair comparison between the robots. The morphology runs consist of a much larger solution space, and evolved gaits might improve when evolved alone, with a locked morphology.

5.4.1 Evolving control system for robot1

Past evolutionary runs were done with 200 generations, but it was chosen to run the search for 1024 generations, to evaluate as many solutions as the morphology evolving runs.

Results Maximum distance walked in simulation is:

- Calculated to a speed of 0.299m/s.
- An increase in 3.2% over the speed of a robot from the Pareto front of the morphology evolving runs with similar weight.

As seen in figure 5.6, only small improvements in maximum distance are achieved after generation 300. Table 5.9 shows the performance statistics of the runs.

Run	$\text{Dist}_{\text{median}}$	$\text{Dist}_{\text{arithmetic mean}}$	$\text{Dist}_{\text{mean(max)}}$	Dist_{max}
740-759	1.9804m	1.9563m	2.0597m	2.3931m

Table 5.9: Table showing the result of runs 740-759, where the control system for robot 1 was evolved.

Analysis Figure 5.6 shows a relatively small difference between average distance and average best distance throughout the runs. The close trailing between average best distance and average distance likely shows a search tactic relying on local search through small mutations, more than long jumps through recombination or high mutations. This is because improvements through local search typically comes gradually, while the sudden high jumps in maximum fitness typically happen when finding superior individuals in new areas of the search space. An increase in 3.2% over a comparable model with similar weight evolved in the morphology runs is a fairly low improvement, and can not be shown to be statistically significant with the relatively low number of simulations run.

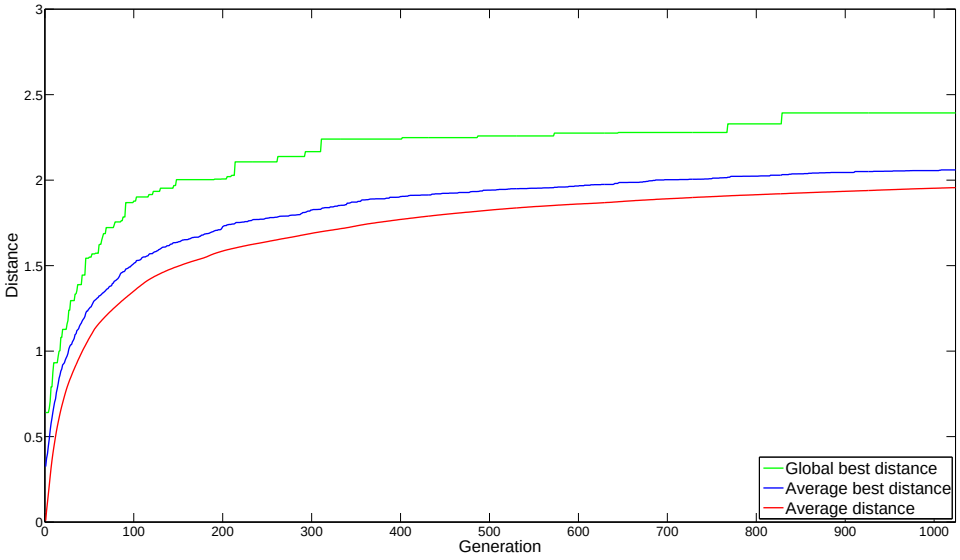


Figure 5.6: Graph showing the fitness progression of runs 740-759.

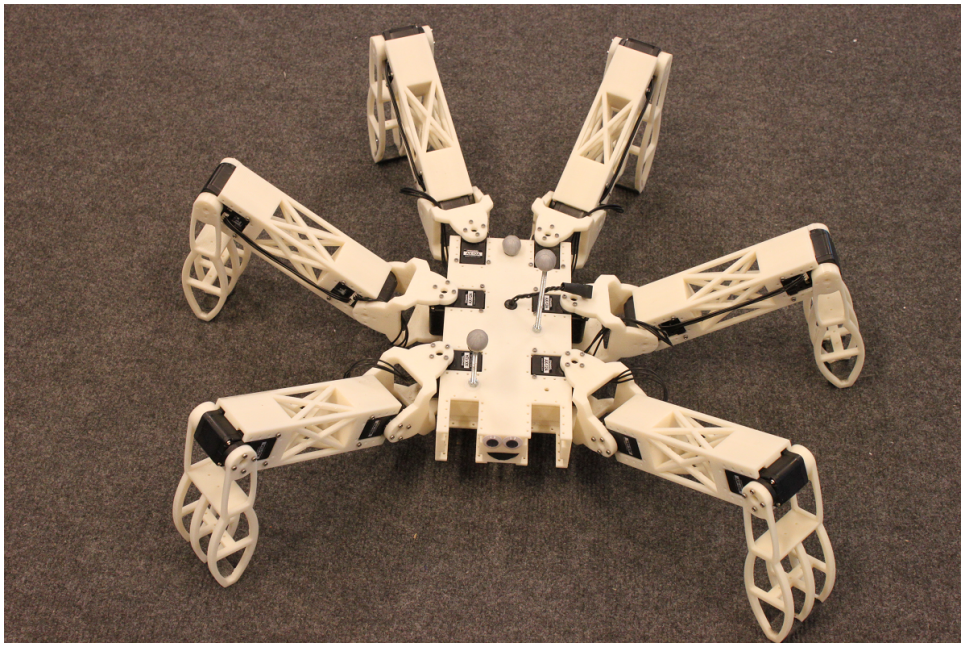


Figure 5.7: Image of the printed and assembled robot1. The face was added when it was presented at the Oslo Maker Faire, to make the robot seem less frightening to children.

5.4.2 Evolving control system for robot2

The small evolved robot (referred to as robot 2) feature long front legs, and shorter back legs, and can be seen in figure 5.8. A single gait was evolved simultaneously as the morphology, but a new group of evolutionary runs with locked morphology was performed to see if results would improve when the control system evolved alone.

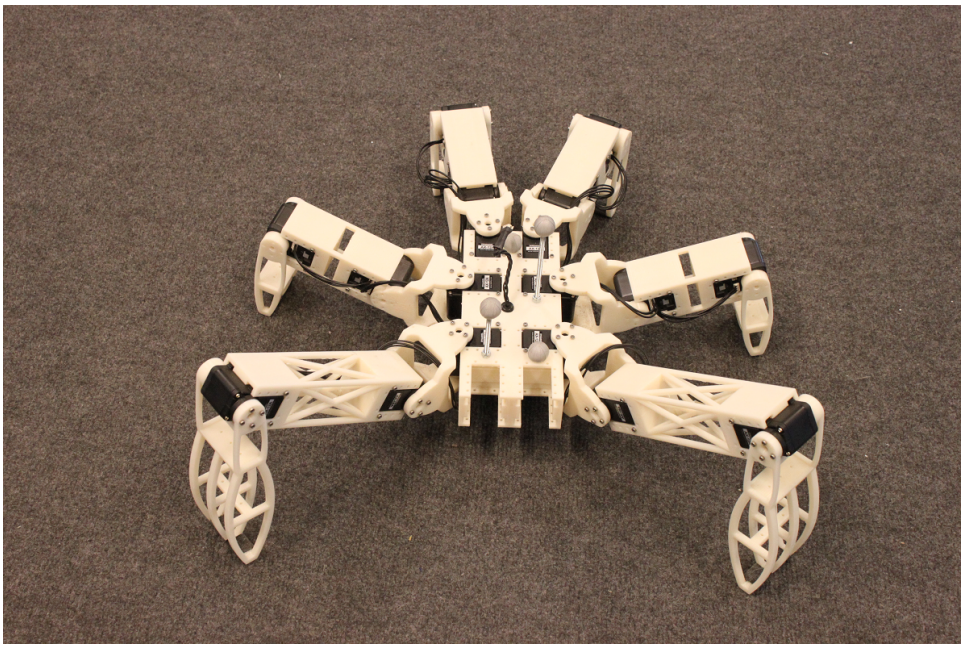


Figure 5.8: Image of the printed and assembled robot2.

Results Maximum distance walked in simulation is:

- Calculated to a speed of 0.234m/s.
- An increase in 15.5% over the speed found in the morphology evolving runs.
- 21.7% lower than the longest distance walked by robot1 in simulation.

Evolving only control system produced better distance scores for 3717 of the 5120 individuals from the final generations of the runs, 73.6%, when compared to the original fitness score received in the morphology evolving run. As seen from figure 5.9, the average fitness has nearly stopped improving, so the received scores are most likely close to the highest score achievable with the evolutionary parameters chosen.

Analysis As the morphology runs were still improving when they were stopped, they might have found a gait comparable to the new best gait, given more time. This is, however, impossible to say without doing new

morphology runs with a greatly increased number of generations. Doing evolutionary runs only including control on the evolved morphologies did, however, show a significant improvement over the past gait, and also produced a number of different gaits. This is also an important feature, as these might have different reality gaps and walking techniques, and could be manually or autonomously inspected and evaluated for different tasks or goals than they were evolved for.

Run	$Dist_{median}$	$Dist_{arithmetic\ mean}$	$Dist_{mean\ max}$	$Dist_{max}$
700-719	1.6858m	1.6629m	1.7329m	1.8747m

Table 5.10: Table showing the result of runs 700-719, where the control system for robot 2 was evolved.

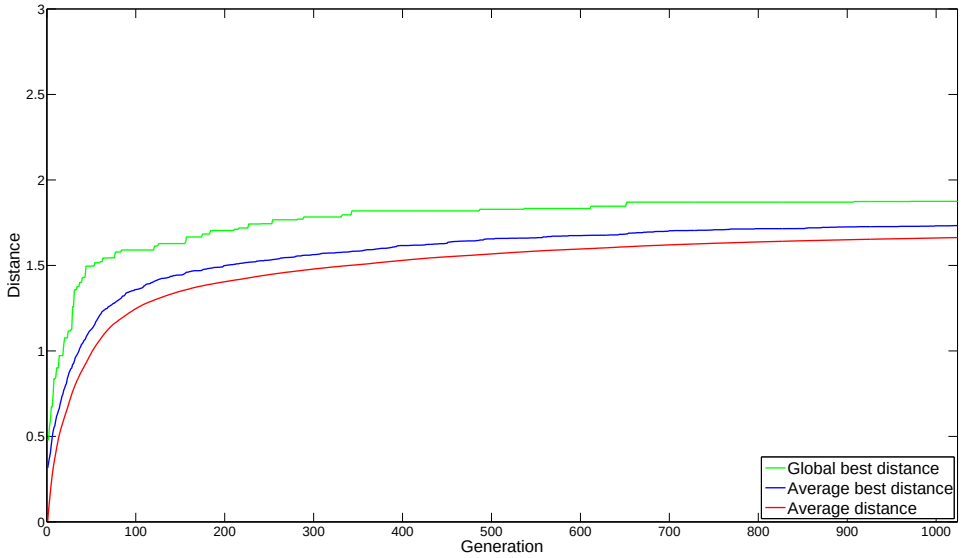


Figure 5.9: Graph showing fitness progression of runs 700-719.

5.4.3 Evolving control system for robot3

The large robot, referred to as robot3, can be seen in figure 5.10. As with robot2, an evolved gait was already made when morphology was evolved, but an additional run with locked morphology was done.

Results Maximum distance walked in simulation is:

- Calculated to a speed of 0.339m/s.
- An increase in 11.7% over the speed found in the morphology evolving runs.
- 13.4% further than the longest distance walked by robot1 in simulation.
- 44.9% further than the longest distance walked by robot2 in simulation.

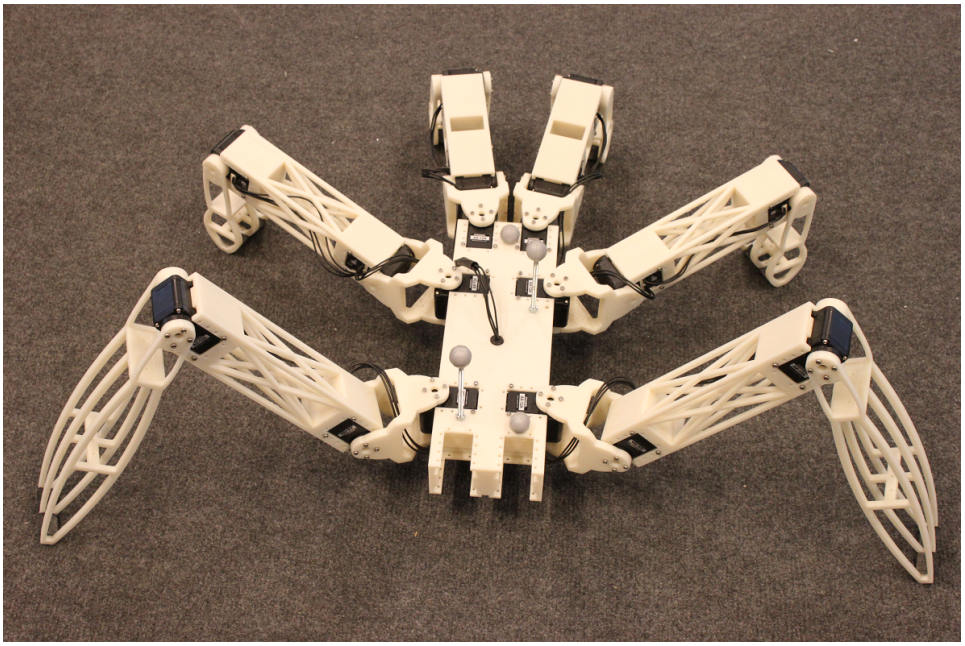


Figure 5.10: Image of the printed and assembled robot3.

Evolving only control system produced better distance scores for 2252 of the 5120 individuals from the final generations of the runs, 44.0%, when compared to the original fitness score received in the morphology evolving run. As seen from figure 5.11, the average fitness has close to stopped improving. This was also seen with the previous two robots, and shows that the received scores are most likely close to the highest score achievable with the evolutionary parameters chosen.

Run	$Dist_{median}$	$Dist_{arithmetic\ mean}$	$Dist_{mean(max)}$	$Dist_{max}$
720-739	2.3271m	2.2194m	2.3227m	2.7138m

Table 5.11: Table showing the result of runs 720-739, where the control system for robot 3 was evolved.

Analysis As with robot2, the higher performance when compared to the evolutionary runs including morphology might only be caused by an insufficient number of generations. However, an improvement was experienced yet again, so this is definitely an interesting development that should be checked by future research. Inspection of solutions showed a wide range of feasible different gaits, some qualitatively similar to the gait generated in the previous runs, some very different.

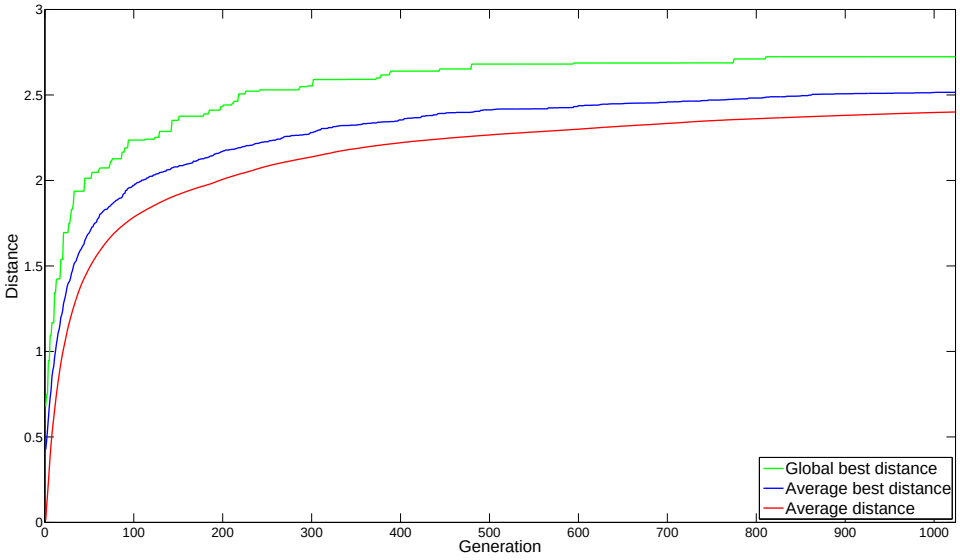


Figure 5.11: Graph showing fitness progression of runs 720-739.

5.4.4 Analysis of control system evolution

It can be hard to distinguish the performances of the three robots when analyzing each robot gait individually. It was therefore decided to plot the performance of each robot's group of control system evolutionary runs, into the Pareto front generated from the morphology evolutionary run. The green lines seen in the graph include the complete final generations from all control evolutionary runs of the three robots.

Results Figure 5.12 shows the Pareto front from the final morphology runs, with the results from the control system runs for each robot plotted in green. Robot1 (middle green line) shows a speed ranging from slightly above the morphology runs to well below the mass of the dominated solutions. Both evolved morphologies shows speeds well above the Pareto front, and even has their slowest individuals fairly close to center mass of the dominated solutions. This graph shows that the reference morphology performs significantly worse than the evolved morphologies when used for evolution of the gaits.

Analysis These results shows, as also seen in past results, that evolution of control after runs co-evolving control and morphology, has the ability to improve solutions significantly. The other, and perhaps more interesting feature seen from the results, is that robot1 had a much lower performance boost when compared to the morphology evolving runs than both evolved robots. This suggests that the morphology of the robot is inferior to the morphologies found in evolution, and gives support to the notion of evolution of morphology and control as an effective design tool in robotics.

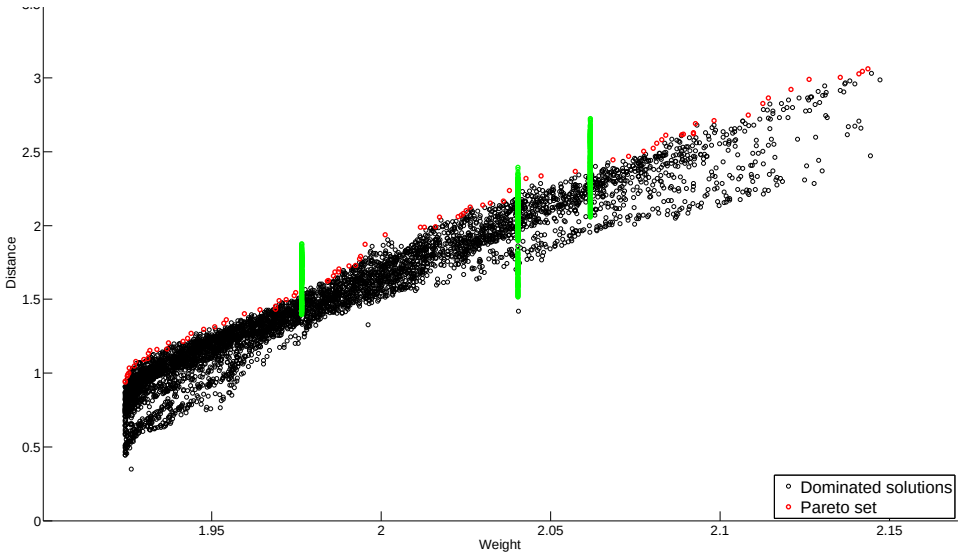


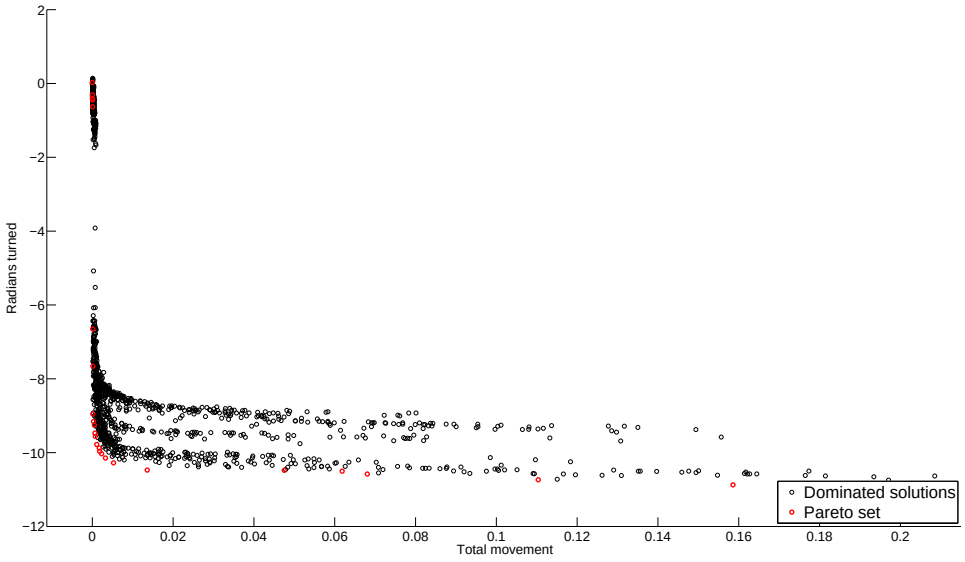
Figure 5.12: The Pareto front of the final morphology runs, run 660-679, compared to the three groups of evolutionary runs for the robot control systems.

5.5 Evolving control system for turning

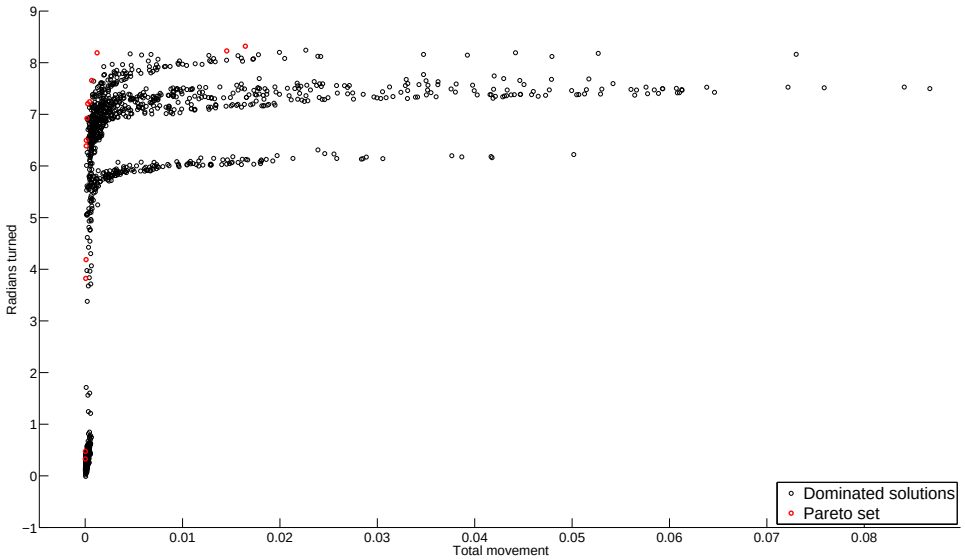
Evolution of	**Parameter tuning of**
Control	
Objectives	**Other parameters (control)**
Min/max turning	Random reset mutation probability (C): 0.02
Min total movement	Non-uniform mutation probability: 1
	Non-uniform mutation step size (K): 0.025
	Non-uniform mutation step size (M): 0.0001

The robots need to be able to turn around, and to keep walking within the perimeter of the tracking equipment, when using the motion capture equipment for testing the performance of gaits on the physical robots. Two new fitness functions were implemented, one that gave a score based on rotation, and one that gave a score based on total displacement in the ground plane (in contrast to the fitness goal used in other simulations, which only rewarded forward movement). More details can be seen in section 4.4.1. These functions should produce a gait that rotates the robot as much as possible, around a given point as close to the robot center as possible. Since this is only used to facilitate motion capture experiments, there were low requirements to the performance of the gaits. Therefore, only 10 runs were done for each robot, 5 in each direction. One direction could be evolved and mirrored for each robot type, but this was not done, as parameterizable mirroring was beyond the scope of this thesis.

Results As seen from figures 5.13a and 5.13b, very different Pareto fronts were achieved, when compared to runs evolving morphology and control. This is the Pareto fronts from robot1, but comparable results were found for the other two robots (Pareto fronts for the all robots can be found in section D.3 in the appendix). The figures show that accepting very small movements results in a much higher turning rate, up to a total movement of about 0.01m, or 1cm.



(a) The Pareto front of runs 760-764, evolving turning left for robot 1.



(b) The Pareto front of runs 765-769, evolving turning right for robot 1.

Figure 5.13: Two Pareto fronts when turning is evolved.

Analysis The number of Pareto optimal solutions was fairly high, but the number of feasible solutions that was decided to test in reality was low. Only a few individuals with high turning radius and movements around

0.01m was selected, and measures could be done to ensure a wider range of Pareto optimal individuals in this area. Since evolution of turning gaits was simply a means to do continuous and autonomous testing, time was not spent to improve performance or diversity of feasible solutions.

5.6 Evolving sideways walk

Evolution of	**Parameter tuning of**
Control	
Objectives	**Other parameters (control)**
Min/max turning	Random reset mutation probability (C): 0.02
Min total movement	Non-uniform mutation probability: 1
	Non-uniform mutation step size (K): 0.025
	Non-uniform mutation step size (M): 0.0001

As a test of the versatility of the evolutionary framework, a new group of runs were conducted to see if a varied group of feasible gaits and robots were still being found, even with new goals. Several people seeing the robots compared them to crabs, and sideways movement like many crabs do was therefore decided as a new objective to test. A new fitness function rating sideways movement speed was implemented, which should require different gaits and changed morphology.

Results Runs 680-699 ran with similar parameters as the final morphology run, and the result statistics were surprisingly very similar. Even so similar that the Wilcoxon rank sum test showed no significant difference in performance between forward and sideways movement, given a significance level of 0.01. The distinct objectives would intuitively lead to different results, but the quantitative data was very similar, even when the gaits were qualitatively very different. The Pareto fronts were not identical, as the new Pareto front followed a steeper rise of distance travelled to weight. Inspecting generated individuals from the Pareto front showed several promising candidates, though this was not pursued further in this thesis.

Analysis The steeper Pareto front indicates that an increase in weight is more rewarding for sideways gaits than for forward gaits, given this evolutionary setup. The generation of feasible candidates for this new goal shows some of the adaptability of both the evolutionary framework and the method of using evolution of morphology and control for robot design.

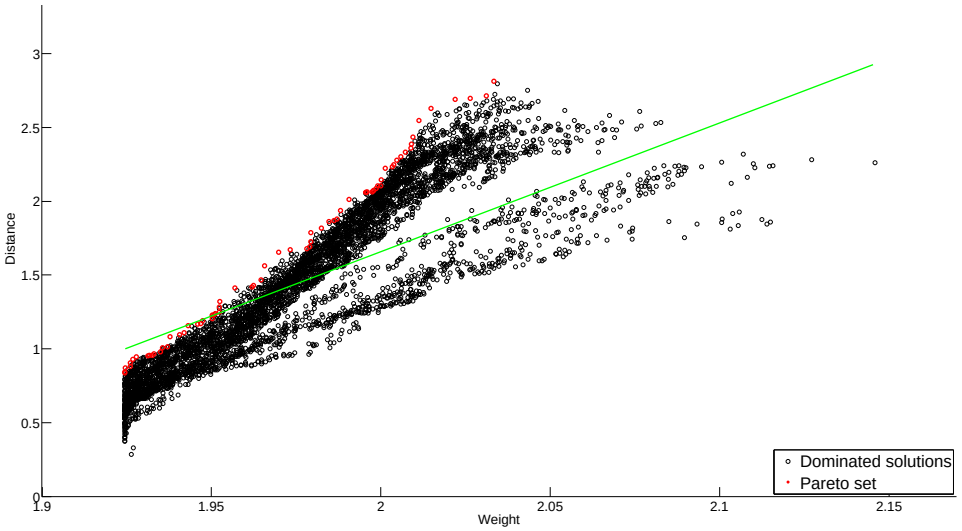


Figure 5.14: The Pareto front of runs 680-699, evolving a sideways movement. The green line is the Pareto front from the final morphology runs, run 660-679.

Chapter 6

Physical experiments and results

Experiments on the physical robots were done using the motion capture equipment to estimate the speed. The first section describes the setup of the motion capture equipment, and the work done to ensure accurate results. The second section describes the experiments done on the manually designed gait for robot1. The third section shows testing of the crude gaits evolved in simulation, on the physical robots. To correct for the reality gap present, two different learning algorithms were run on the robots, and the experiments and results from these algorithm runs are described in the two subsequent sections. Gait verification analysis has to be done to get comparable data on the results from the learning runs, and this is addressed in the last section of the chapter.

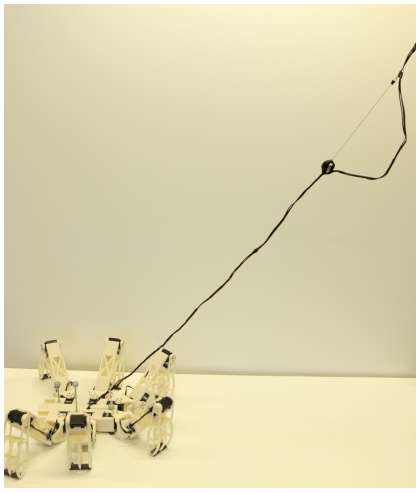
Experiment parameters Earlier tests showed that the servos used have a chance of shutting down with an overload error when torque applied gets near the limit of the servos. This, combined with the fact that they stall at about 75% of their rated torque [85], prompted the decision to reduce the max torque to 50%. Servo speed was set to 50% of max speed to reduce the strain on the motors, and reduce the chance of breaking the plastic parts of the robots.

Evaluation of learning runs Since two different learning runs are used, there is a need for measurements able to distinguish the performance of different runs. As the distribution of performance can not assumed to be either normal or lacking outliers, the median is preferred to mean. Two features have been chosen as comparable measurements, the highest fitness achieved throughout the run, and the fitness value at the end of the run. Standard deviation and max values are also calculated, although max value is more vulnerable to noise than the median, and is also less stable across different runs, which is why these are only used as guides.

Enabling autonomous tests The goal when using the motion capture setup is enabling autonomous testing without human intervention. All

tests in the thesis was done under observation, but there are several reasons why making sure the system runs by itself is important. The most important reason for excluding the engineer from the process is that humans make errors and affect the accuracy and precision of the measurements. Things like placement of robots, time taking, or even recoding or transcription of data heightens the probability of errors, when compared to a fully automatic system. It also frees up the time of the observer to observe the gait and development of the learning runs, rather than focus on a number of tasks that needs to be performed. The biggest challenge of running continuous and autonomous motion capture tests on a robot is making sure the robot stays inside the camera area, and that cable tangling or twisting, or legs catching on the cables, are prevented. This was solved by evolving turning gaits for both left and right turning, and alternating between these each time the robot reached the edge of the defined capture area. This ensured that the robot did not exit the area, and prevented cable twisting by alternating turning directions.

Motion capture harness To stop the wire from tangling or otherwise interfering with the robot, a wired testing setup needed to be made. There was no need to add complexity to the system by sending commands wirelessly, since power would have to be supplied by cable either way. Since the robot needs to move, a wiring harness needs to be made that affects the robot in the least possible way. This was done by using two different cable management techniques. A pulley system was constructed, as seen in figure 6.1b. Weight was added to both sides of the system by putting a number of metal nuts into plastic bags. A fairly high weight was added to make sure the weight of the varying length of cable on each side of the pulley was as insignificant as possible. A ball bearing was added with a 3d printed cable guide attached. This provided a fairly frictionless setup, but reacted slowly to changes to the cable. To enable a quicker and more fluid reaction to the robot movement, a self retracting reel system was added to the lower part of the cable, as seen in figure 6.1a. The reel was a modified self retracting key reel, which is cheap and easily available. The system put a relatively low force on the cable to the robot, with the reel allowing fast movements, acting as a spring intermediary for the pulley system, which allows retraction for the full length of the cable.



(a) An image of the robot connected to the reel system in the wiring harness.

(b) An image of the pulley system used in the wiring harness.

Figure 6.1: Images of the wiring harness used for motion capture experiments.

6.1 Testing and tweaking of the motion capture setup

There are various options available for tweaking the motion capture setup, including settings and locations of the cameras, point cloud solver, and streaming or visualization of the data. The goal is to reduce position vibration, mainly caused by small inaccuracies in the cameras affecting the point cloud solution, and reduce the number of wrongly reported positions or orientations, mainly caused by misidentification of markers. These goals are somewhat conflicting, as settings making markers visible on a smaller number of cameras lower noise that cause position vibrations, caused by minor differences in point positions from different cameras. They do, however, also increase misidentification problems, as a smaller number of point sources are available to distinguish different orientations and positions of the robot.

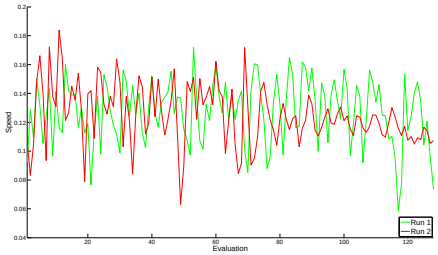
Decreasing wrong point cloud calculations The biggest problem in the initial motion capture runs, were measurements of wrong position or orientation. The position would jump a couple of cm along the ground, or the software would report wrong orientation, and therefore also position. Effective placement of the reflective markers is challenging, as they all need to occupy the small space available on the robot, but still be as far apart as possible with unique distances between all markers. Initial placement of three markers on the base of the robot proved ineffective, so the total number of markers were increased to four, while two makers were mounted on threaded rods of different heights above the robot, to feature markers in the full three dimensional space, rather than in a plane. The marker placement for all robots can be seen in figures 5.7, 5.8, and 5.10. In addition

to this, new software was installed that featured a new point cloud solver, and a new strobe mode for the IR lights in the cameras. This resulted in stronger lights and clearer reflections, thereby increasing the effective range of each camera, making the camera overlap larger. This resulted in stable and correct measurements in the middle of the capture area, but the system still struggled near the edges of the capture area, due to the lower number of cameras in view. To overcome the negative effects of low camera coverage, the minimum number of lines required to estimate the position was lowered from four to two, while the minimum distance between calculated points for each camera was lowered from 10mm to 3mm. This change led to good performance for all three robots across the whole capture area. Table 6.1 shows the settings used for all motion capture experiments. Other settings not mentioned in the table are settings dependant on individual camera setup.

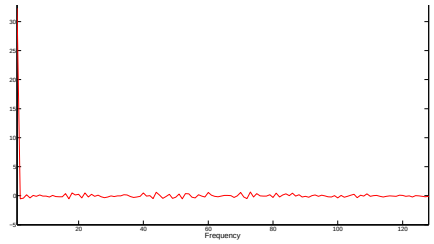
Software version	OptiTrack Arena 1.7.3
Point cloud solver	V2.0
Camera processing mode	Precision grayscale
IR setting	High power, Strobed
Roundness filter	60
Min line	2
Residual	4mm

Table 6.1: Settings used for all motion capture experiments.

Measurement of noise When plotting the graph of the first runs featuring identical gaits, it is apparent that there is a great deal of noise in the measurements. Noise is defined here as any data point not representing a correct absolute measurement of the gaits speed. As seen in figure 6.2a, the results vary between a speed of about 0.06m/s to 0.18m/s while executing the same gait. The noise also seems to increase and decrease in a somewhat repetitive way. To check for any fixed frequency noise oscillations, a Fourier analysis was done on the measurements, but as seen from figure 6.2b, there does not seem to be any apparent frequencies of noise in the system. To see whether the noise was too high for successful evaluation of gaits, the measurements were sorted and graphed, to more easily see the distribution of measurements. Figure 6.3 shows the sorted data from the two runs. The lines are fairly linear, but feature a couple of outliers on each side of the distribution. To remove these, a simple filter was applied by removing 15% of the data from each side of the distribution. This should remove any outliers caused by negative factors like slippage or harness exaggerations, and any potentially positive outliers like measurement errors or manual movement of the robot by human intervention. In addition to this passive filtering, a Tukey filter was applied to the data (see section 2.3.11 for more info on Tukey filtering).



(a) Speed measurements over time.



(b) A fast fourier transform of the measurements to the left.

Figure 6.2: Results from the first two runs of identical gaits using robot1.

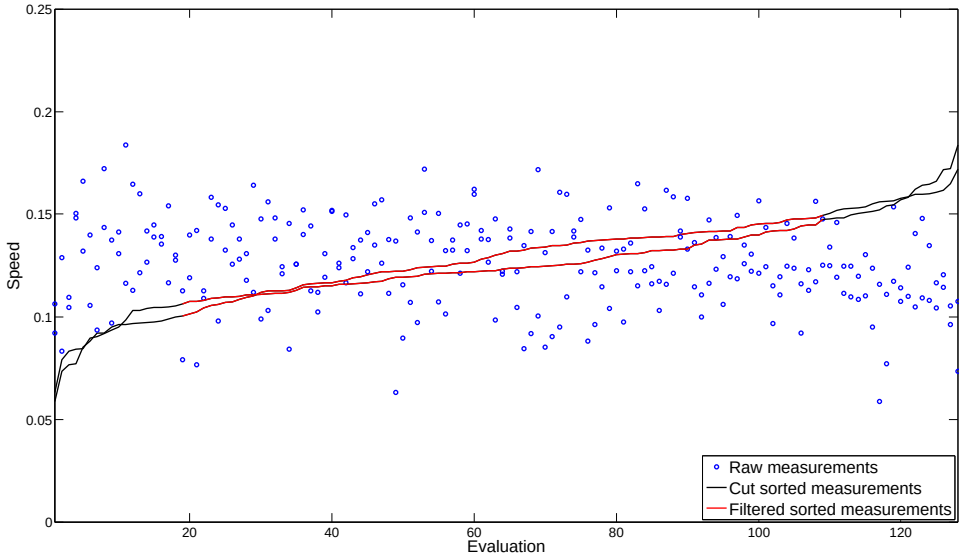


Figure 6.3: Graph showing motion capture performance of two identical gaits from simulation, running on robot 1.

6.1.1 Motion capture analysis

More runs would need to be made to draw a statistically valid conclusion that the 128 evaluations are enough for an absolute measurement, but the small difference between the two lines seen in figure 6.3 suggests consistent measurements. It was therefore decided that single runs of 128 evaluations were enough for measurement of absolute gait performance. There are still a number of sources of measurement noise, even with a perfectly working motion capture setup. Among the highest sources of inaccuracies, is the physical behavior of the servos. The coreless DC motors within, all have slightly different speed and torque characteristics, and they all change over time due to heat, humidity, wear, and other factors. Another common source of speed variations is the inaccuracy in the servo movement. The robot sometimes works right on the edge legs of crashing into each other, and due to the low accuracy, sometimes does. The problem with this is that some of the parts have a tendency to get stuck to each other, and this may ruin a whole evaluation of four seconds, or even several evolutions in a row.

Another problem with continuous testing of gaits is that gaits are started in the middle of another gait. If these two gaits involve poses which is hard or impossible to transition between, the last gait might end up with a lower score than it might have received, if all gaits started from a common initial position. This, however, would require substantially longer test times, and does not affect similar gaits or gaits with compatible poses. It was therefore acknowledged and accepted as a source of noise, but not dealt with due to already long evaluation times. The test harness of the robot also contributes to the variations and inaccuracies in the measured speeds. It adds a non-linear friction from both the pulley system and the reel system. The Fourier transform in figure 6.2b shows no dominating frequencies which could indicate an oscillation present in the harness, but there is still a speed and orientation specific friction constant added to the robot. There is also a multitude of smaller sources of noise, like uneven carpets, control bus overload or communication errors, small differences in printed parts, timing issues of computer or communication. These are, however, negligible to the above mentioned sources, along with the fairly large inaccuracies in the motion capture equipment.

6.2 Testing manually designed gait on robot1

A manually designed gait was made early in the project as a way of testing the mechanical and electrical setup of the first robot. The gait was represented as a set of six poses with individual durations, and was generated by a very simple path planning script in Matlab using the inverse kinematics of the robot. This gait was controlled using the Arbotix micro controller board. To enable a similar evaluation as evolved gaits, the controller was also implemented in the simulation environment to run the manually designed gait on the physical robot. This removes any execution or evaluation difference present between the simulation environment and the Arbotix, and ensures measurements with the same accuracy and precision.

Experiment To test the performance of the manually designed gait, a run was performed without any learning algorithm modifying the gait. As seen in figure 6.4, the measurements are fairly close, except for a clear dip in speed towards the end of the run. Table 6.2 shows a significant change of minimum value as filtering is applied, with a smaller cut happening to the maximum value. This does, however, not affect the mean to a large degree, and the median is left unchanged as the same amount is removed from top and bottom.

Analysis The temporary dip in speed measurements towards the end of the run was most likely caused by the imperfect pulley system, twisting or tangling of the cables, or other external causes. This should not affect the evaluation of the gait, and shows the use for a filtering technique. The red line seen in figure 6.4 shows the speed measurements after sorting and removal of 15% of the data points on each side, and appears close to linear,

	Min	Mean	Median	Max	SD
Unfiltered	0.059	0.122	0.125	0.143	0.014
.15 filtered	0.106	0.123	0.125	0.134	0.008
Tukey filtered	0.096	0.123	0.125	0.143	0.011

Table 6.2: Table showing the results of the manually designed gait for robot1 from the motion capture equipment. All numbers are given in m/s.

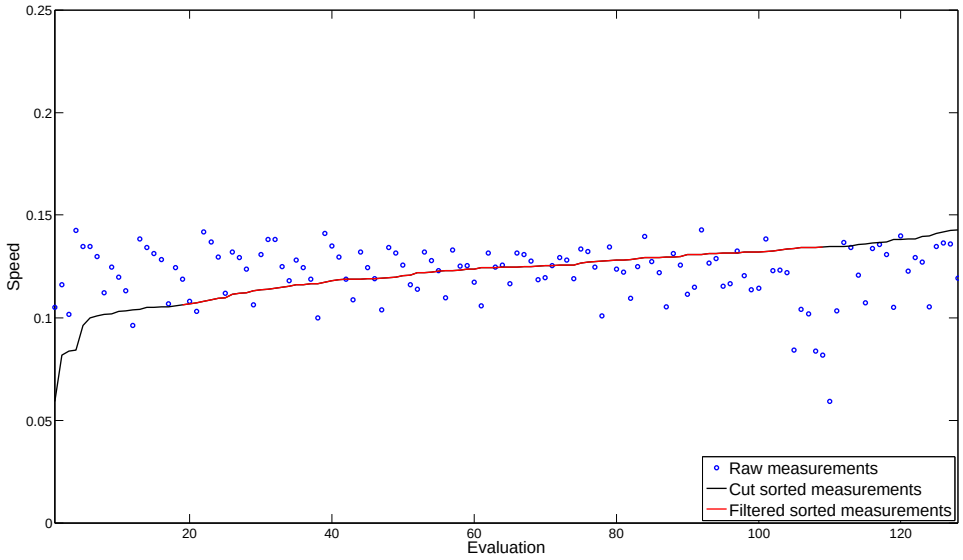


Figure 6.4: Graph showing motion capture performance of the manually designed gait on robot 1.

suggesting a fairly accurate evaluation of the gait speed. As seen in table 6.2, the mean remains fairly constant before and after the filtering, and together with the non-skewed distribution of the filtered results, should make for a good measure of the performance of the gait. The median is also used in comparisons with future gaits as a control, as it should give a good indication of performance, also if the distribution is skewed in future experiments.

6.3 Crude gait experiments

Gaits evolved in simulation, without any change or modification, is referred to as crude gaits. Crude gaits are typically expected to perform worse when tested on the physical robots, as the reality gap tends to favor the simulator, and the lack of external disturbances there. A single gait was selected for each robot from the control system evolutionary runs, and was tested twice for each robot to ensure a good baseline measurement for comparison against the manual gait, simulation, and learning runs.

Robot	Filter	Min	Mean	Median	Max	SD	Rank
1(M)	Unfiltered	0.059	0.122	0.125	0.143	0.014	2
1(M)	.15 filtered	0.106	0.123	0.125	0.134	0.008	
1(M)	Tukey filtered	0.09	0.123	0.125	0.143	0.011	
1	Simulation			0.257			
1	Unfiltered	0.059	0.126	0.125	0.184	0.021	2
1	.15 filtered	0.100	0.126	0.125	0.149	0.012	
1	Tukey filtered	0.073	0.127	0.125	0.184	0.021	
2	Simulation			0.224			
2	Unfiltered	0.032	0.119	0.122	0.189	0.024	3
2	.15 filtered	0.088	0.120	0.122	0.143	0.014	
2	Tukey filtered	0.062	0.120	0.122	0.171	0.022	
3	Simulation			0.336			
3	Unfiltered	0.115	0.164	0.161	0.241	0.022	1
3	.15 filtered	0.141	0.162	0.161	0.191	0.013	
3	Tukey filtered	0.115	0.163	0.161	0.220	0.021	

Table 6.3: Table showing motion capture performance of the crude gaits on all robots. All speed measures are given in m/s. Performance of the manual gait is included for convenience. All numbers other than rank are given as m/s.

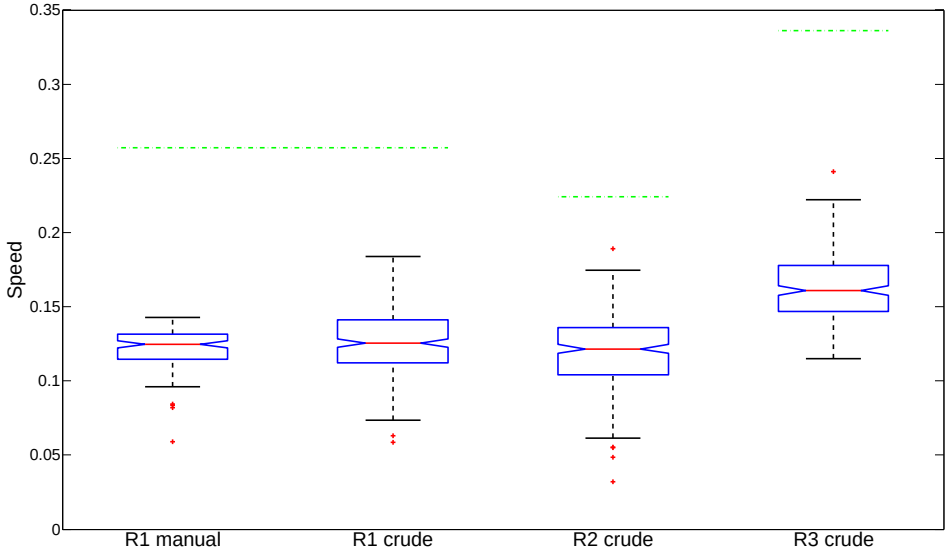


Figure 6.5: Box plot of the speeds from the motion capture of crude gaits on all robots. The green dashed line indicates the speed of the robot in simulation.

6.3.1 Robot1 results

Median speed is:

- 51.4% lower than the crude gait in simulation.
- The same as the median speed of the physical manual gait[1].

The top performing gait from run 750 was chosen for physical tests on robot1. The gait was among the fastest individuals from control evolving runs 740-759, and featured a gait similar to the tripod gait, with alternating legs in the air and touching the ground. Figure 6.5 shows two low outliers being cut by the Tukey filter, and a significant spread in measurement data.

6.3.2 Robot2 results

Median speed is:

- Median speed is 45.5% lower than the crude gait in simulation
- Significantly better than the physical crude gait on robot1[2].

The top performing gait from run 700 was chosen for the physical tests on robot2. As the gait selected for robot1, this also moves similar to the tripod gait. Figure 6.5 shows four low outliers and one high outlier being cut by the Tukey filter, and as with robot1, significant spread in measurement data.

6.3.3 Robot3 results

Median speed is:

- Median speed is 52.1% lower than the crude gait in simulation
- Significantly better than the physical crude gait on both robot1 and robot2[3].

The top performing gait from run 727 was chosen on robot3. Contrary to the past two robots, this gait did not resemble the classic tripod gait. The two front legs move in sync, and in opposite phase to the two side legs. The back legs each move in sync with one of the two other leg pairs. Since three legs are still resting on the ground at all times, a three dimensional base of support is still being maintained. The difference in base of support is very high during this gait, being about half the length of the robot. This can therefore not be considered a static gait since, opposed to the other two robots, the line of gravity is outside the base of support during the gait. Figure 6.5 shows a single high outlier being cut by the Tukey filter, and comparable spread in measurement data to the two other robots.

[1] According to a Wilcoxon rank sum test with significance level 0.01.
[2] According to a Wilcoxon rank sum test with significance level 0.01.
[3] According to a Wilcoxon rank sum test with significance level 0.01.

6.3.4 Crude gait analysis

As seen in table 6.3, median speed for all robots remains unchanged for all filtering methods. This implies that filtering is not needed when only comparing the medians, but it is kept for convenience since not all future experiments may behave the same way. The evolved gait for robot1 and the manually designed gait have received the same rank, as there is no statistically significant difference between the populations, according to a Wilcoxon rank sum test with significance level 0.01. This still bodes well for the evolvable controller, as the reality gap of the gait high, and should be reduced by the use of machine learning runs. Figure 6.5 shows a fairly large reality gap of about 50% on all evolved gaits. There are many differences between the simulator and reality that causes this, but the largest differences is in force calculation for the motors in simulation, the friction calculation in simulation, and the different environment in reality, including the cable pulley system and imperfect walking surface.

Manual vs evolved gait The perhaps most apparent difference between the physical crude gait and the learned gaits, is the difference in spread of measurements. As seen in table 6.3, the manual gait has a spread of 0.047m/s between max and min tukey filtered speed, while the other robots have differences of 0.111m/s, 0.109m/s and 0.105m/s respectively, interestingly very close to each other. This can also be seen in the difference in standard deviations, from 0.011 in the manual gait to 0.121-0.122 in the evolved gaits. Differences in morphology can not be said to be the cause, since the evolved gait on robot1 has the same spread as the other two, while the manual gait for robot1 had a different spread than the evolved gait running on the same hardware. One major difference is that the manual gait includes several margins of error on both angles, durations and torque, either by design or introduced unintentionally. The evolved gaits having been designed to fully utilize all available torque, and is therefore much closer to the maximum torque limits of the servos. By operating on the edge of available torque, small changes due to factors like model inaccuracies, leg slippage, motor temperature, or higher friction on the cable pulley system can affect the speed to a much higher degree than if there was a small factor of safety between the torque used in the simulation, and what was available in real life. Another reason which often affects the spread of measurements is whether a gait is static or dynamic. The manually designed gait was designed to be completely static, which in its nature is less affected by external factors. Robot3 has a gait which is out of balance for a longer duration than the two others. The dynamic gait can therefore not be said to be the only reason for the difference in spreads, since the gait of robot3 does not show a higher difference than the other two evolved gaits, although it could be a contributing factor.

6.4 One plus lambda experiments

One plus lambda was chosen for its use of reevaluations, and the hope that this would help mitigate the problem of noisy measurements. The algorithm was run with lambda set to four, and reevaluations happening if no improvement has been seen for five evaluations. Reevaluations results in the score of the individual being set to the arithmetic mean of the new measurements and all previous scores. This should in theory lessen the degree of noise for every reevaluation of an individual. Eight runs of 128 four second evaluations were done on each of the selected crude gaits, which in practice resulted in about 8 hours of continuous testing.

Robot	SD(max)	Median(max)	Max	SD(last)	Median(last)	Max(last)
1	0.015	0.205	0.213	0.015	0.170	0.185
2	0.015	0.187	0.207	0.026	0.147	0.201
3	0.013	0.236	0.248	0.020	0.156	0.191

Table 6.4: Table showing the results of running all one plus lambda learning algorithms on all robots. All numbers are given as m/s.

6.4.1 Robot1 results

Median last value is:

- 36.0% higher than the median speed of the crude gait
- 33.8% lower than the crude gait in simulation

In addition to this, a significant loss of performance is experienced during the runs, as seen in the difference in median max and median last of table 6.4. This can also be seen in figure 6.6, where runs have a fairly stable max value from evaluation 10, although gaits are reevaluated and replaced by better performing gaits throughout the runs.

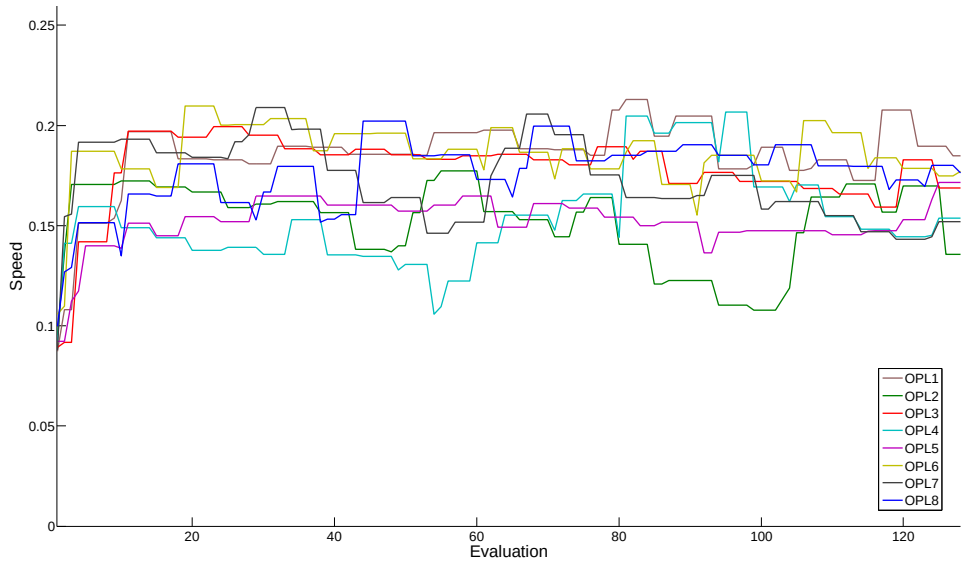


Figure 6.6: Graph showing the current best individual of the one plus lambda learning algorithms for robot1.

6.4.2 Robot2 results

Median last value is:

- 20.5% higher than the median speed of the crude gait
- 34.4% lower than the crude gait in simulation

As seen in figure 6.7, the runs of this robot also achieve close to the maximum speed after only 10 evaluations. As with robot1, reevaluations and improvements are done throughout the runs.

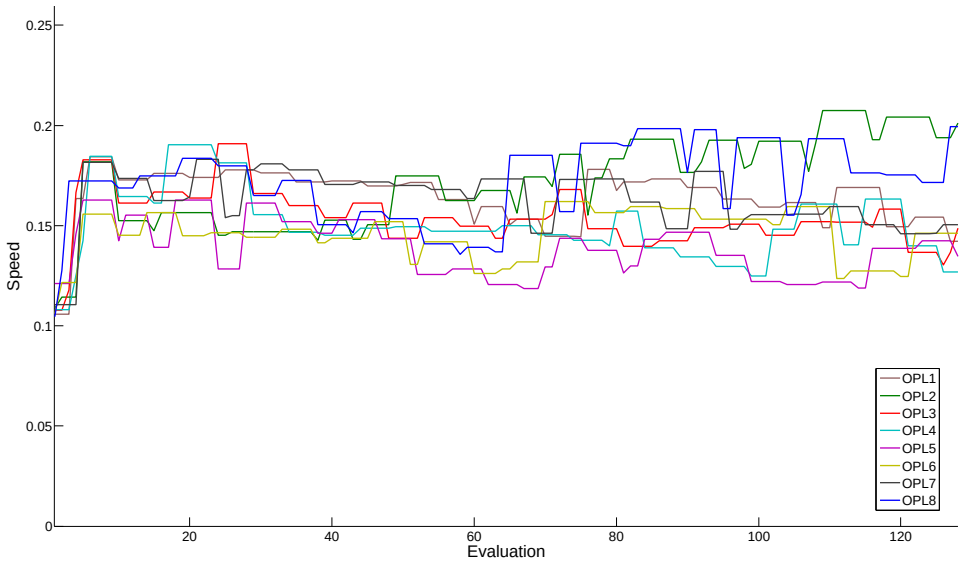


Figure 6.7: Graph showing the current best individual of the one plus lambda learning algorithms for robot2.

6.4.3 Robot3 results

Median last value is:

- 3.1% lower than the median speed of the crude gait
- 53.6% lower than the crude gait in simulation

As seen from figure 6.8, these runs behave slightly differently than the runs of the previous two robots. As before, max values are achieved after about ten evaluations, but there is a distinguishable, fairly constant, decline in speed experienced throughout the run. The median fitness of the last individuals ends up below the original fitness of the crude gait which it uses as a starting point for learning. This can also be seen in table 6.4, as robot3 has the highest decrease in max to max last of all the robots by 33.9%, to 21.4% for robot2 and 17.1% for robot1.

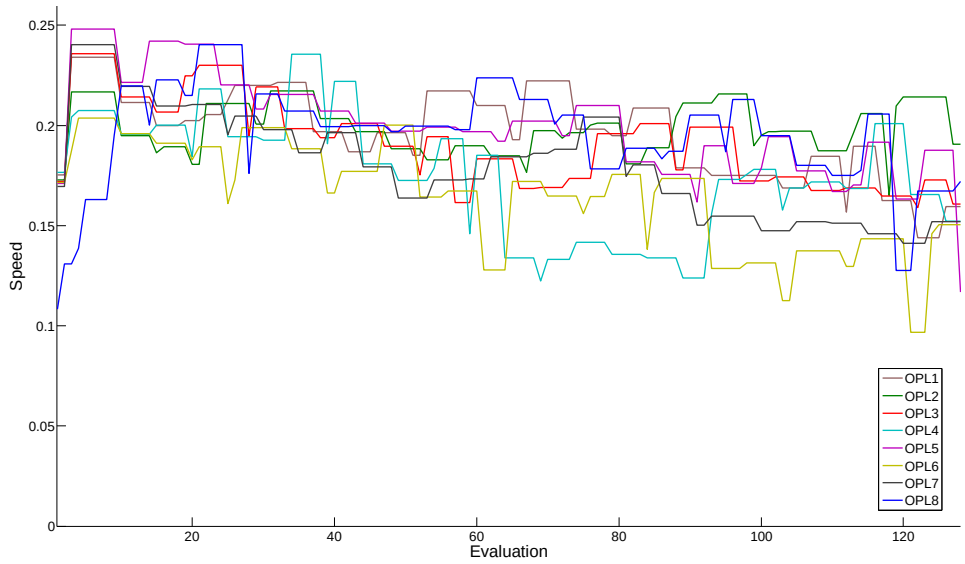


Figure 6.8: Graph showing the current best individual of the one plus lambda learning algorithms for robot3.

6.4.4 OPL analysis

One plus lambda is an algorithm that in theory never loses the best individual, and as such should not lose performance during the run. This algorithm does, however, do a reevaluation of the best individual if no improvement is seen for 5 new individuals. This is to discourage noisy readings from forcing premature convergence to a false local optima by giving individuals with overly optimistic measurements new evaluations. The mean of all measurements are then set as the new fitness value. Figures 6.6 and 6.7 show runs of robot1 and robot2 having fairly constant fitness from evaluation 10, most likely caused by quite frequent overestimations and following reevaluations. Figure 6.8 does, however, show a different progression of the search. The graph shows a steady decline of fitness throughout the run. The most probable explanation is that the size of robot3 affects the speed by requiring a higher amount of torque, which in turn causes the servos to heat up more quickly. DC motors have a fairly high change of characteristics as temperature is changed, as a hot motor has a higher idle speed, but lower stall torque than a cold motor. The AX18 servos even has coreless DC motors, which from the lack of metal mass in the core heat up even quicker than DC motors with the normal solid metal core. The steady decrease in speed makes the median last fitness artificially low, but should not affect the results of the learning algorithm considerably, since reevaluations ensure a constant readjustment to the lower performance.

Improvement over crude gaits As seen when testing the crude gait, noise was not a factor after doing two complete runs on the same run without learning applied. Noise is, however, a much bigger problem for the gaits generated in the learning run, some of which might only have received a

single evaluation. This implies that the improvement of the new individuals over the crude gaits can be considered to be directly comparable. The improvements seen for all three robots can therefore be said to be promising, although not conclusive, without verification from experiments done in the same way as the crude gaits.

6.5 Simulated annealing experiments

Simulated annealing was chosen as the second learning algorithm due to the fact that the start of the algorithm features very low selection pressure and a high degree of exploration, which hopefully handles the noise in the measurements, even though no reevaluations are used. As the run progresses, the probability of accepting inferior solutions lower, leading to a higher degree of exploitation. As with OPL, eight runs of 128 four second evaluations were done on each robot, using its crude gait as a starting point.

Robot	SD(max)	Median(max)	Max	SD(last)	Median(last)	Max(last)
1	0.020	0.166	0.198	0.017	0.144	0.173
2	0.029	0.173	0.188	0.032	0.173	0.180
3	0.016	0.242	0.254	0.012	0.200	0.222

Table 6.5: Table showing the results of running all simulated annealing learning algorithms on all robots. All numbers are given as m/s.

6.5.1 Robot1 results

Median last value is:

- 15.2% higher than the median speed of the crude gait.
- 44.0% lower than the crude gait in simulation.
- 15.3% lower than the median last value when using OPL.

As seen from figure 6.9, the speeds vary greatly in the beginning of each run, while stabilizing towards the last quarter of the evaluations. Some runs stabilize earlier, like SA3 and SA1, which stabilize between evaluation 50 and 70, effectively cutting the number of used evaluations from the run in half. This is in contrast to the OPL runs on the same robot, which progress throughout all evaluations. Table 6.5 shows a smaller difference in max and max last, possibly showing a smaller degree of rejection of noisy measurements than OPL.

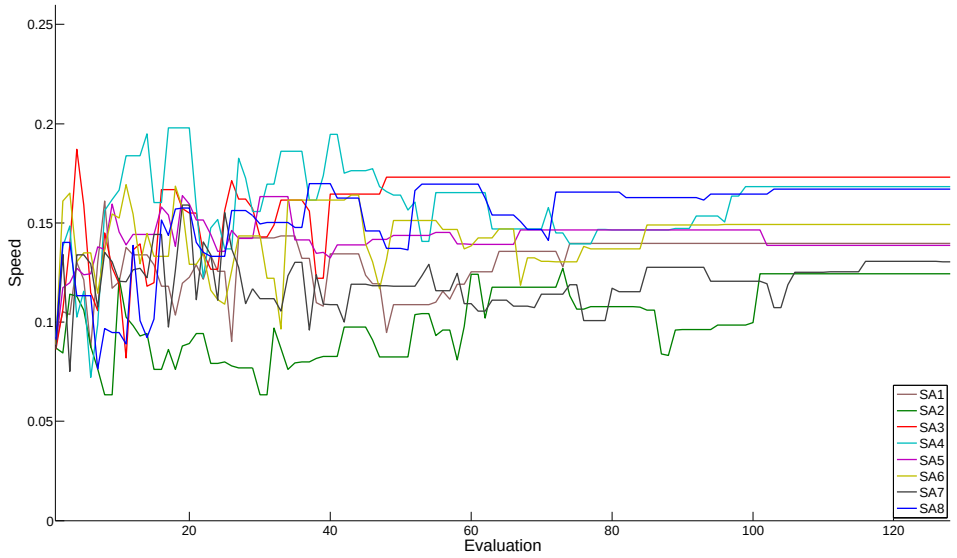


Figure 6.9: Graph showing the current best individual of the simulated annealing learning algorithms for robot1.

6.5.2 Robot2 results

Median last value is:

- 41.8% higher than the median speed of the crude gait.
- 22.8% lower than the crude gait in simulation.
- 17.7% higher than the median last value when using OPL.

As seen from figure 6.10, several of the runs reach stable values far earlier than using OPL, which was also seen when using SA on robot1. It also shows a higher increase in fitness during the run than for robot1. When comparing to the OPL learning on robot2, five of the best performing SA runs end up with similar speed, close to the maximum speed, while the OPL runs result in a grouping of six runs closer to the minimum speed. This affects the median last fitness value, perhaps making SA seem better than it is.

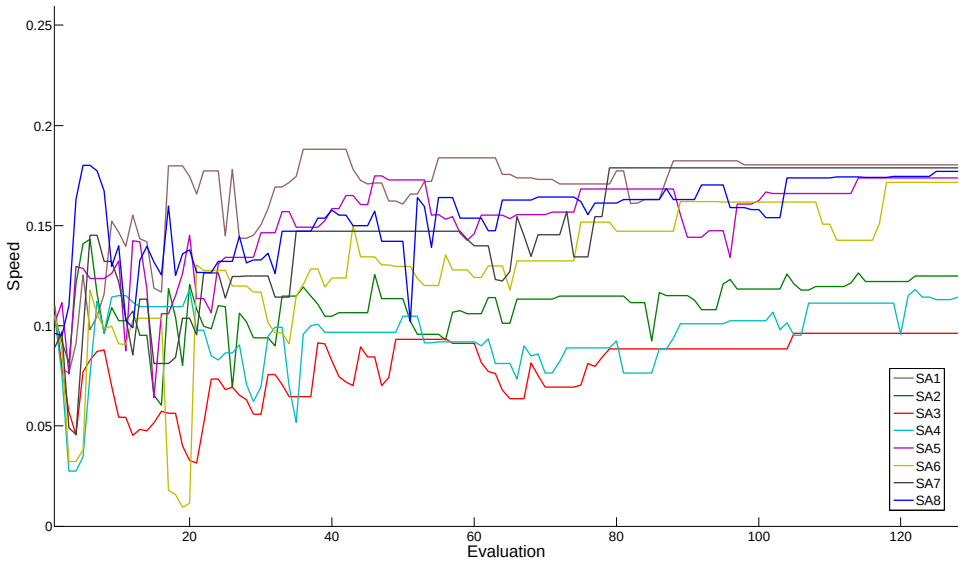


Figure 6.10: Graph showing the current best individual of the simulated annealing learning algorithms for robot2.

6.5.3 Robot3 results

Median last value is:

- 24.2% higher than the median speed of the crude gait.
- 40.5% lower than the crude gait in simulation.
- 28.2% higher than the median last value when using OPL.

As seen in figure 6.11, most of the runs prematurely converge on a stable value. This is happening quicker than for the previous robots, and the run does not suffer the steady decline as seen in the OPL learning runs for the same robot.

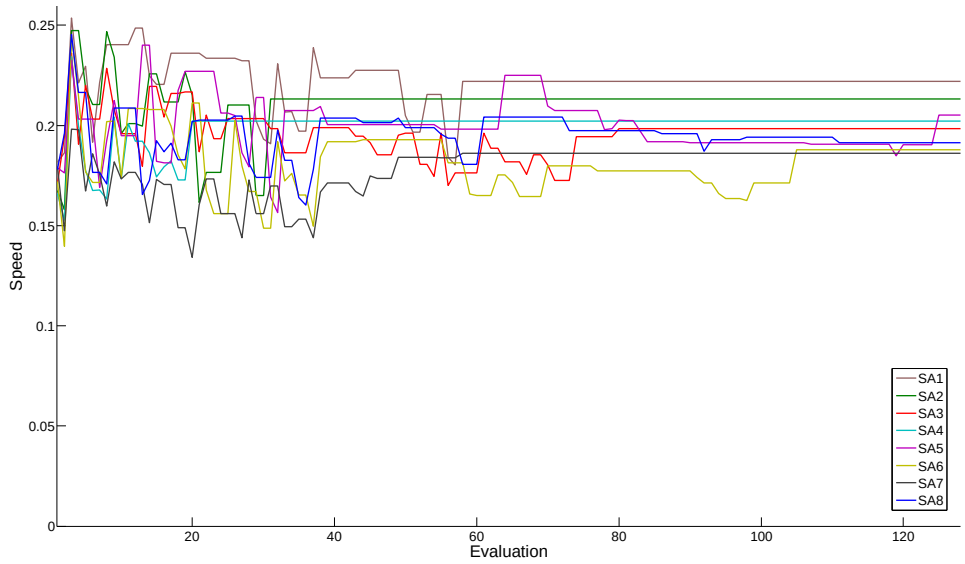


Figure 6.11: Graph showing the current best individual of the simulated annealing learning algorithms for robot3.

6.5.4 SA analysis

As seen in figures 6.9 and 6.10, both runs on robot1 and robot had a high exchange of individuals early in the run, showing the low selection pressure during the start of SA runs, which typically results in higher genetic diversity. Runs on robot1 tended to stabilize around evaluation 80, while robot2 seemed to reach the stable values later, for some runs even improving until the end. This might indicate less noisy data from the simulated annealing runs on robot2, or even that the morphology is better suited for the exploitation being done in the last phase of the simulated annealing run.

Lack of reevaluations In contrast to one plus lambda, simulated annealing never reevaluates individuals during the runs. An apparent problem with this is clearly seen in figure 6.10, where improvements are, for most runs, only done in the first half of the evaluations. The cause of this can be seen when comparing to the graph of the OPL runs for the same robot in figure 6.8. The steady decline in performance is caused by the motors heating up, but is handled by the reevaluations of the algorithm. Since the probability of accepting inferior individuals decreases continuously when using SA, no new solutions will be accepted after about half the evaluations, as all new individuals are performing poorer than the individuals tested earlier because of the lower torque caused by the temperature change.

Noise Another suspected problem of simulated annealing is that overly optimistic measurements due to noise would be dominant, due to combined effect of noisy measurements and lack of reevaluations. SA runs on robot3 resulted in higher median last speed than the OPL runs, but this seems

unlikely to be correct, as only about half the evaluations were effectively used by the search algorithm. This can not be said for certain, without verifying the gaits and comparing the speeds given during learning to the median speed through a whole run without a running learning algorithm.

6.6 Gait verification experiments

Both learning algorithms are susceptible to noise. OPL does reevaluations if no improvement is found for a number of evaluations, but there is no guarantee of several measurements on the same individual if a new and improved individual is found close to the end of the run. Simulated annealing never reevaluates any solutions, so should, in theory, be more sensitive to the noisy environment. Because of this, a large uncertainty exists for the speed of the individuals measured during the learning algorithms. For this reason, two individuals from the sixteen individuals generated of each robot are tested without learning, in the same way as the crude gait was tested. This ensures measurements without substantial noise, and makes it possible to compare the performance of the new gaits to the original crude gait, and calculate whether a difference is statistically significant or not. One individual from each learning algorithm was chosen from each robot for gait verification, and the tests were done identical to the tests run in the crude gait experiments.

Robot	Gait		Min	Median	Max	Rank
1	Manual	Tukey filtered	0.096	0.125	0.143	4
1	Crude	Tukey filtered	0.073	0.125	0.184	4
1	SA3	Tukey filtered	0.080	0.119	0.164	5
1	OPL8	Tukey filtered	0.128	0.173	0.213	2
2	Crude	Tukey filtered	0.061	0.122	0.175	5
2	SA1	Tukey filtered	0.131	0.173	0.209	2
2	OPL8	Tukey filtered	0.141	0.186	0.217	1
3	Crude	Tukey filtered	0.115	0.161	0.222	3
3	OPL2	Tukey filtered	0.125	0.181	0.240	1
3	SA1	Tukey filtered	0.127	0.187	0.246	1

Table 6.6: Table showing the results of the motion capture performance of all learned gaits selected for verification. Crude gait performance is given in gray to ease comparison. All numbers other than rank are given as m/s.

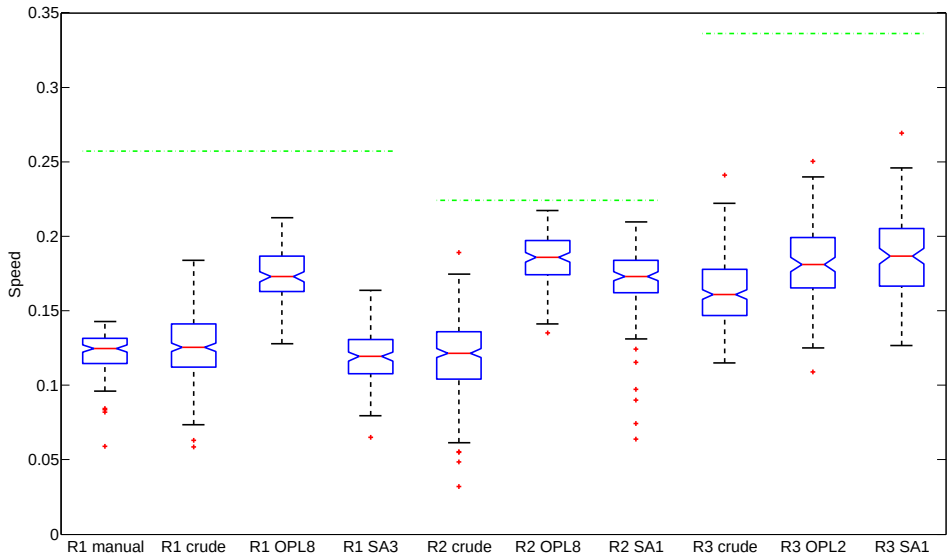


Figure 6.12: Box plot showing all verification runs on all robots. The performance in simulation is shown in green.

Robot1		Robot2		Robot3	
OPL8	SA3	OPL8	SA1	OPL2	SA1
-3.9%	-31.2%	-6.5%	-3.9%	-5.2%	-15.8%

Table 6.7: Table showing the median of the resulting individuals from all learning runs compared to their measured speed during the respective learning runs.

Robot1		Robot2		Robot3	
OPL8	SA3	OPL8	SA1	OPL2	SA1
+38.4%	-4.8%	+52.5%	+41.8%	+12.4%	+16.1%

Table 6.8: Table showing the median of the verified individuals from all learning runs compared to the median speed of the respective robots' crude gait.

6.6.1 Robot1 results

The individual SA3 was selected for having the highest score of all SA individuals. The best performing gait from OPL walked with a very curved trajectory, which means tests take considerably longer, and are more sensitive to noise in the measurements. The individual was therefore rejected, and replaced by the second best individual from the OPL runs, OPL8. Table 6.6 shows OPL8 outperforming the manual gait, the physical crude gait, and SA3. SA3, however, is significantly worse than all other verified gaits on robot1. Table 6.7 shows a high difference in loss of

Robot1		Robot2		Robot3	
OPL8	SA3	OPL8	SA1	OPL2	SA1
-32.7%	-53.7%	-17.0%	-22.8%	-46.1%	-44.3%

Table 6.9: Table showing the median of the resulting individuals from all learning runs compared to the speed of the crude gait in simulation.

performance when compared to the results during the learning run of 27.3 percentage points. This greatly affects the performance of the two gaits, as seen in table 6.8, where OPL8 outperforms the crude gait, and SA3 actually loses speed when compared to the physical crude gait. This change can also be seen when comparing the median speed to the original gait in simulation, as seen in table 6.9, where OPL performs significantly better.

6.6.2 Robot2 results

Individuals SA1 and OPL8 were both selected for having the highest speed recorded during their learning runs. As seen in table 6.7, both performed slightly below their original speed measurements by about 5%. Table 6.8 shows a fairly large improvement in both individuals of above 40%, but as seen in table 6.9, they are still both under the speed of the crude gait in simulation by over 15%, although they are the two gaits closest of the six selected gaits.

6.6.3 Robot3 results

As with robot2, the highest performing individual from each learning algorithm run was selected for verification, OPL2 and SA1. As seen in table 6.7, the OPL individual only had a 5% lower speed than measured during the run, while the SA individual had a nearly 16% loss, about thrice the loss of the OPL individual. Even with this higher loss, the median of SA1 is still higher than OPL2, and has the highest rise of speed of the two, when compared to the crude gait, as seen in table 6.8. As shown in table 6.9, both gaits performed about 45% below the speed experienced in simulation, better than the 50% in robot1, but far below the 20% experienced in robot2.

6.6.4 Gait verification analysis

It is hard to draw general conclusions from the physical experiments, since only one gait from each learning run was selected and verified. There are, however, some interesting indications. OPL8 on robot2 performed as well as both learned gaits on robot3, According to a Wilcoxon rank sum test with significance level 0.01, which is an excellent result considering the difference in weight and size of the two robots. Robot2 is also lighter and smaller than robot1, yet outperformed the manually designed gait, the crude gait from simulation, and the two learned gaits, according to a Wilcoxon rank sum test with significance level 0.01. Since the torque had to be lowered by 50%, it is impossible to say whether robot3 would have had a higher benefit of

the added torque than the two other robots, and outperformed them like it did in simulation. Figure 6.12 shows that robot2 almost reached the speed experienced in simulation, and might actually have achieved it given full torque. This shows that machine learning is able to at least reduce the reality gap considerably.

Chapter 7

Discussion

This chapter concludes the thesis with a discussion of the results and related issues, conclusive summary, and finally describes possibilities for future work.

7.1 General discussion

Re-evolution of control The results from chapter five, and specifically figure 5.12, shows that the evolution of control systems done on the morphologies from the main runs increases the quality of individuals considerably, which goes against the expected performance increase from co-evolving morphology and control. All three robots outperformed their co-evolved control systems. The increase in performance could, however, be caused by the smaller solution spaces when only evolving the control system, and full runs with a more sophisticated end criteria could result in equal, or even better performance, from the co-evolutionary runs.

Manually designed robot and gait The manually chosen parameters for robot1 were the result of looking at comparable robots, engineering intuition, and very simple force calculations. A professionally designed robot platform might perform considerably better, which might skew the results of this thesis. The manually designed gait could also be improved further, though the tripod gait is by many considered among the top performing gaits for six legged robots. The evolutionary system was, however, designed on the same premises as the manually designed robot and gait, and might improve to a comparable degree, if designed or improved by professional engineers or researchers.

Robot platform Figure 2.7 from chapter 5 shows that the evolutionary platform is able to produce a wide range of morphologies. The tests of the evolved individuals also showed a diverse group of qualitatively different gaits. Being able to successfully adapt to objectives of both turning and walking sideways, as evident from the results seen towards the end of chapter five, also shows the platform being capable of adjusting to different goals.

Improvement by learning algorithms Chapter six shows the high reality gap experienced when transferring to reality. Much of the gap is closed by learning, and tuning of learning parameters or use of other algorithms might improve the results even further. Both evolved morphologies outperformed the manually designed robot, and the evolved gait outperformed the manually designed gait on robot1 as well. This shows that evolution of morphology and control is a feasible design technique, and outperformed the traditional design approach in this single case. The successful use of the algorithm would allow a robot designer to either get the same speed from a lighter robot, or get higher speed from a robot with the same weight.

Addition of front legs or tools The robot initially had front legs, but it was decided that these would be removed to lessen the weight and reduce the solution space for the evolutionary search. Front legs could again be added, but would not affect speed on a normal surface, since eight legs over four would still result in two walking phases with continuous forward movement, using a tripod style gait. It would, however, increase balance and load capacity, and might increase speed if using a closed loop control and walking on a demanding surface. Other tools or sensors could be added instead if needed.

Fitness functions Speed was the main measure of quality used for the robots in this thesis. Weight was also added to ensure a greater variation of individuals, and to have two partially conflicting objectives, as this often enhances the quality of available solutions [50]. A positive side effect of lower robot weight is lower costs and shorter production time. A problem seen on some of the evolved solutions, were that they moved in a long arc, rather than in a straight line. Since the fitness function only rewarded for maximum forward movement, this was not penalized in simulation, but made the gait infeasible in reality. A new fitness goal that recorded the absolute sideways displacement could be made, and a minimization of this new fitness function should help eliminate the problem of robots walking in circles. Another fitness function that might aid the search is to reward gaits that lift the whole body of the robot. Friction was heightened for the base alone to encourage this behavior, but a goal of lifting the base to a certain height would most likely give smaller ground friction and higher speeds, both in simulation and reality. Another problem with some of the gaits, is an uneven movement along the direction of travel. A constant velocity is traditionally seen as the quickest way to move, and is observed in the classic tripod gait. A fitness goal of constant velocity could be added to encourage even, fluid movements.

7.2 Conclusion

This thesis presented the development of a robotic platform for evolutionary experiments. The robot has six legs, with the possibility of adding two more servos to the front for tools or additional legs. It was

integrated into a physics simulation environment used at the University of Oslo, and a parameter search for both control evolution, and evolution of control and morphology, was conducted to give a starting point for other researchers using parts of the proposed framework. The analysis of the evolved robots showed a wide range of different morphologies and gaits, which is promising for solving more demanding fitness goals than speed. This shows that the designed platform works for different goals, and can serve as a valuable tool for evolutionary experiments.

A manually designed instance of the robotic platform was made, and was thoroughly tested against the two evolved models in both simulation and reality. Analysis of the evolved gaits for all three robots showed a higher quality than the evolved controllers for the manually designed robot in simulation. Evolving morphologies resulted in a wide range of both robots and controllers to choose from, which in itself is advantageous over the few resulting robots of conventional robot design. This improvement leaves the robot designer free to conclude on all trade-offs in the system, which may very well change during the development phase. One of the evolved morphologies also showed a novel approach to legged walking, bypassing the traditional limitation of servo sectors.

A single gait was selected from each robot for testing in real life, and was run through two learning algorithms to lessen the reality gap. Each robot gait went through eight runs of one plus lambda learning and eight runs of simulated annealing. One resulting gait was selected from each algorithm type and verification of speed showed the manually designed robot being outperformed by both evolved robots. The smaller evolved robot outperformed the manually designed robot in speed and weight, while the larger evolved robot outperformed it in speed, with a slight increase in weight. This shows that evolution of morphology was able to make a robot both smaller and faster than the manually designed robot, and has therefore served as a successful tool for robot design in this thesis.

7.3 Future work

The first goal of this thesis was to design a framework for evolutionary experiments, and this opens up for a fairly large amount of future work.

Parameter tuning and control Manual parameter tuning was done in this thesis, but using statistical methods for optimization of parameters before the run has the potential of improving the speed of the algorithm, and the quality of evolved solutions[40]. Parameter control, i.e. changing key parameters during the run[41], would also be an interesting addition to the framework.

Conventional robotics The manually designed robot used as a baseline in this thesis was mainly designed without many of the tools used in conventional robotics. Usage of techniques and engineering know-how from other areas of robotics could be used to design a new robot using the parameterizable model, and new comparisons could be done to the evolved solutions found in this thesis, or new evolved solutions found after optimization of the evolutionary system as well.

Novel legged design The morphology of the large evolved robot showed a novel design of legged robots, by enabling intersection of legs in a gait. This has not been seen in any previous work, and should be investigated further by conventional robotics techniques and tools.

Reality gap reduction Several techniques exist for the reduction of the reality gap, and these could be used in conjunction with the framework, as the reality gap was shown to be as high as a performance loss of 50%. Techniques like noise generation [81], adaption of simulator parameters [82, 83], or transferability approach [84] could al be used on the evolutionary framework developed in this thesis.

Other control systems The control system used was a modified version of the controller used by Koos et al. [63], made symmetric and easier to evolve and limit. This controller was strictly open-loop, getting no feedback from the robot itself. Using more advanced controllers, and especially closed-loop controllers, would be an interesting study. This would enable sensors to affect the gait (and even the evolution of morphology), and would enable more complex behaviors.

Fitness functions Two fitness functions were used during the evolution in this thesis; speed, and weight. Speed was the main goal, while total weight was used to control the runs and ensure diversity in the solutions through innovization[50]. One of the goals when creating the robotic platform was to make an adaptable platform capable of more complex tasks. It would therefore be interesting to test the robot on more complicated fitness functions. This could be tied to sensors or tools the robot could carry, or to the environment around the robot, given a closed-loop controller.

Co-evolution and incremental evolution Results showed a significant improvement when doing evolution of control systems alone on the morphologies resulting from the co-evolutionary runs. This should be verified or invalidated by further experiments. This would open up new exciting research possibilities of doing incremental evolution on the designed platform, and could potentially lead to even higher performance increases than seen in the simple incremental evolution used in this thesis. The work of this thesis included an initial co-evolutionary run of morphology and control, and then a control only evolutionary run. Being able to cut the computationally expensive co-evolutionary run and instead

do several runs of control or morphology individually, might improve the quality of solutions and the time used for evolution, but raises several questions about the loss of co-evolution and it's perceived benefits.

Evolutionary aided design The work in this thesis involved first manually designing a model, then doing parameter optimization on that model using evolutionary algorithms. Evolutionary aided design takes the process one step further, and analyses the output of the parameter optimization, before manually designing a new model based on the lessons learned by evolution. This involves analyzing the Pareto front in several dimensions, in contrast to the two fitness dimensions used in this thesis. An analysis would then show the significance of the different lengths and sizes of the robot, and ease the many decisions faced when manually designing a legged robot.

Online evolution Machine learning was used to lessen the reality gap, but this was only intended to get back some of the lost fitness from moving from the simulation. Online evolutionary adaption is an evolutionary process done during operation of the robot, and makes the robot able to adjust to new tasks or environments by changing its behavior. Many researchers argue that this is a necessary part of an evolutionary design process, and that the high quality of evolutionary processes seen in nature requires a period of adjustment and habituation [5]. This could also be combined with the principles of simulated annealing, to do a higher degree of adjustment out of the box, and then adapt in smaller increments as time progresses.

Self modifying hardware in ER When the Golem project [17] brought evolved robots into the real world in 2000, the field of evolutionary robotics was changed forever. With the progression of technology, and the advancements in robot construction and evolution of morphology, it might be time to take one step further. By designing robots that can modify their hardware as done in simulations, one would not be limited to an adaption of the control system for new tasks or environments, but could also change the morphology of the robot in a process similar to online evolutionary adaption. This would open up a new dimension to evolutionary robotics, and might give the push the field needs to be fully accepted in the robotics community. The platform proposed in this thesis could be expanded to include self-modifying hardware, like variable length femurs and tibias, and be subjected to an online optimization of morphology as well as control.

Bibliography

[1] T. Kamegawa et al. “Development of the snake-like rescue robot "kohga".” In: *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*. Vol. 5. Apr. 2004, pp. 5081–5086.

[2] “Biological Inspiration: From Carangiform Fish to Multi-Joint Robotic Fish.” In: *Journal of Bionic Engineering* 7.1 (2010), pp. 35–48.

[3] “Abigaille-III: A Versatile, Bioinspired Hexapod for Scaling Smooth Vertical Surfaces.” In: *Journal of Bionic Engineering* 11.1 (2014), pp. 1–17.

[4] M.A. Diftler et al. “Robonaut 2 - The first humanoid robot in space.” In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. May 2011, pp. 2178–2183.

[5] KennethO. Stanley. “Why Evolutionary Robotics Will Matter.” In: *New Horizons in Evolutionary Robotics*. Ed. by Stéphane Doncieux, Nicolas Bredèche, and Jean-Baptiste Mouret. Vol. 341. Studies in Computational Intelligence. Springer Berlin Heidelberg, 2011, pp. 37–41. ISBN: 978-3-642-18271-6.

[6] Jean-Arcady Meyer, Phil Husbands, and Inman Harvey. “Evolutionary robotics: A survey of applications and problems.” In: *Evolutionary Robotics*. Ed. by Philip Husbands and Jean-Arcady Meyer. Vol. 1468. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1998, pp. 1–21.

[7] P.J Fleming and R.C Purshouse. “Evolutionary algorithms in control systems engineering: a survey.” In: *Control Engineering Practice* 10.11 (2002), pp. 1223–1241.

[8] Andrew L. Nelson, Gregory J. Barlow, and Lefteris Doitsidis. “Fitness functions in evolutionary robotics: A survey and analysis.” In: *Robotics and Autonomous Systems* 57.4 (2009), pp. 345–370.

[9] Josh Bongard, Victor Zykov, and Hod Lipson. “Resilient machines through continuous self-modeling.” In: *Science* 314.5802 (2006), pp. 1118–1121.

[10] Ho-sik Seok et al. “An on-line learning method for object-locating robots using genetic programming on evolvable hardware.” In: *in: Proceedings of the Fifth International Symposium on Artificial Life and Robotics, AROB'00*. 2000, pp. 321–324.

[11] Frank Hoffmann and Gerd Pfister. *Evolutionary Learning of a Fuzzy Control Rule Base for an Autonomous Vehicle*. 1996.

[12] G.S. Hornby et al. “Evolving robust gaits with AIBO.” In: *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*. Vol. 3. 2000, 3040–3045 vol.3.

[13] Francesco Mondada, Edoardo Franzi, and Andre Guignard. “The development of khepera.” In: *Proceedings of the 1st international Khepera workshop*. Vol. 64. 1999, pp. 7–14.

[14] Sara Lohmann et al. “Aracna: An open-source quadruped platform for evolutionary robotics.” In: *Artificial Life*. Vol. 13. 2012, pp. 387–392.

[15] John M Galeotti et al. “EvBots-The Design and Construction of a Mobile Robot Colony for Conducting Evolutionary Robotic Experiments.” In: *CAINE*. 2002, pp. 86–91.

[16] Karl Sims. “Evolving virtual creatures.” In: *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. ACM. 1994, pp. 15–22.

[17] Jordan B Pollack and Hod Lipson. “The GOLEM project: Evolving hardware bodies and brains.” In: *Evolvable Hardware, 2000. Proceedings. The Second NASA/DoD Workshop on*. IEEE. 2000, pp. 37–42.

[18] Chris Leger. “Automated synthesis and optimization of robot configurations: an evolutionary approach.” PhD thesis. Carnegie Mellon University, 1999.

[19] Gregory S Hornby, Jordan B Pollack, et al. “Body-brain co-evolution using L-systems as a generative encoding.” In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*. 2001, pp. 868–875.

[20] Eivind Samuelsen, Kyrre Glette, and Jim Torresen. “A Hox Gene Inspired Generative Approach to Evolving Robot Morphology.” In: *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*. GECCO '13. Amsterdam, The Netherlands: ACM, 2013, pp. 751–758.

[21] Hod Lipson and Jordan B Pollack. “Automatic design and manufacture of robotic lifeforms.” In: *Nature* 406.6799 (2000), pp. 974–978.

[22] G.S. Hornby, H. Lipson, and J.B. Pollack. “Evolution of generative design systems for modular physical robots.” In: *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*. Vol. 4. 2001, 4146–4151 vol.4.

[23] Ian Macinnes and Ezequiel Di Paolo. “Crawling out of the simulation: Evolving real robot morphologies using cheap reusable modules.” In: *In Artificial Life IX: Proc. Ninth Intl. Conf. on the Simulation and Synthesis of Life*. MIT Press, 2004, pp. 94–99.

[24] A. Crespi et al. “Salamandra Robotica II: An Amphibious Robot to Study Salamander-Like Swimming and Walking Gaits.” In: *Robotics, IEEE Transactions on* 29.2 (Apr. 2013), pp. 308–320.

[25] K.A. Morgansen, B.I. Triplett, and D.J. Klein. "Geometric Methods for Modeling and Control of Free-Swimming Fin-Actuated Underwater Vehicles." In: *Robotics, IEEE Transactions on* 23.6 (Dec. 2007), pp. 1184–1199.

[26] QiDi Wu et al. "Survey of locomotion control of legged robots inspired by biological concept." English. In: *Science in China Series F: Information Sciences* 52.10 (2009), pp. 1715–1729.

[27] Kan Yoneda and Shigeo Hirose. "Dynamic and static fusion gait of a quadruped walking vehicle on a winding path." In: *Advanced Robotics* 9.2 (1994), pp. 125–136.

[28] M.W. Spong and S. Hutchinson. *Robot Modeling and Control*. Wiley, 2005.

[29] Nate Kohl and Peter Stone. "Machine learning for fast quadrupedal locomotion." In: *AAAI*. Vol. 4. 2004, pp. 611–616.

[30] Haocheng Shen et al. "Learning fast quadruped robot gaits with the RL power spline parameterization." In: *Cybernetics and Information Technologies* 12.3 (2012), pp. 66–75.

[31] Bill Horne, M. Jamshidi, and Nader Vadiee. "Neural networks in robotics: A survey." English. In: *Journal of Intelligent and Robotic Systems* 3.1 (1990), pp. 51–66. ISSN: 0921-0296.

[32] Auke Jan Ijspeert. "Central pattern generators for locomotion control in animals and robots: A review." In: *Neural Networks* 21.4 (2008), pp. 642–653.

[33] Auke Jan Ijspeert. "A connectionist central pattern generator for the aquatic and terrestrial gaits of a simulated salamander." In: *Biological cybernetics* 84.5 (2001), pp. 331–348.

[34] Suchan Lee et al. "Evolving gaits for physical robots with the HyperNEAT generative encoding: the benefits of simulation." In: *Applications of Evolutionary Computation*. Springer, 2013, pp. 540–549.

[35] Kyrre Glette et al. "Evolution of locomotion in a simulated quadruped robot and transferral to reality." In: *Proceedings of the Seventeenth International Symposium on Artificial Life and Robotics*. 2012.

[36] John J Grefenstette, David E Moriarty, and Alan C Schultz. "Evolutionary algorithms for reinforcement learning." In: *Journal Of Artificial Intelligence Research* 11 (1999), pp. 241–276.

[37] Antoine Cully and Jean-Baptiste Mouret. "Learning to Walk in Every Direction." In: *The Computing Research Repository* (2013).

[38] David B Fogel. *Evolutionary computation: the fossil record*. Wiley-IEEE Press, 1998.

[39] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. "Comparison of Multiobjective Evolutionary Algorithms: Empirical Results." In: *Evol. Comput.* 8.2 (June 2000), pp. 173–195. ISSN: 1063-6560.

[40] S. K. Smit and A.E. Eiben. "Comparing parameter tuning methods for evolutionary algorithms." In: *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*. May 2009, pp. 399–406.

[41] A.E. Eiben, R. Hinterding, and Z. Michalewicz. "Parameter control in evolutionary algorithms." In: *Evolutionary Computation, IEEE Transactions on* (July 1999).

[42] D.H. Wolpert and W.G. Macready. "No free lunch theorems for optimization." In: *Evolutionary Computation, IEEE Transactions on* 1.1 (Apr. 1997), pp. 67–82.

[43] Olivier Francois and C. Lavergne. "Design of evolutionary algorithms-A statistical perspective." In: *Evolutionary Computation, IEEE Transactions on* 5.2 (Apr. 2001), pp. 129–148.

[44] Y. Jin. "A comprehensive survey of fitness approximation in evolutionary computation." English. In: *Soft Computing* 9.1 (2005), pp. 3–12.

[45] Andrea Toffolo and Ernesto Benini. "Genetic diversity as an objective in multi-objective evolutionary algorithms." In: *Evolutionary Computation* 11.2 (2003), pp. 151–167.

[46] Nicholas Freitag McPhee and Nicholas J Hopper. "Analysis of genetic diversity through population history." In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Vol. 2. Citeseer. 1999, pp. 1112–1120.

[47] Aniko Ekart and Sandor Z Nemeth. "Maintaining the diversity of genetic programs." In: *Genetic Programming*. Springer, 2002, pp. 162–171.

[48] Raymond Burke, Steven Gustafson, and Graham Kendall. "A Survey And Analysis Of Diversity Measures In Genetic Programming." In: *GECCO*. Vol. 2. 2002, pp. 716–723.

[49] E.K. Burke, S. Gustafson, and G. Kendall. "Diversity in genetic programming: an analysis of measures and correlation with fitness." In: *Evolutionary Computation, IEEE Transactions on* 8.1 (Feb. 2004), pp. 47–62.

[50] Kalyanmoy Deb and Aravind Srinivasan. "Innovization: Discovery of Innovative Design Principles Through Multiobjective Evolutionary Optimization." In: *Multiobjective Problem Solving from Nature*. Ed. by Joshua Knowles et al. Natural Computing Series. Springer Berlin Heidelberg, 2008, pp. 243–262. ISBN: 978-3-540-72963-1.

[51] K. Deb et al. "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II." In: *Trans. Evol. Comp* 6.2 (Apr. 2002), pp. 182–197.

[52] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. *SPEA2: Improving the Strength Pareto Evolutionary Algorithm*. Tech. rep. 2001.

[53] Christian Igel, Nikolaus Hansen, and Stefan Roth. "Covariance matrix adaptation for multi-objective optimization." In: *Evolutionary computation* 15.1 (2007), pp. 1–28.

[54] Hans-Georg Beyer and Hans-Paul Schwefel. "Evolution strategies–A comprehensive introduction." In: *Natural computing* 1.1 (2002), pp. 3–52.

[55] Hans-Paul Schwefel. *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie: mit einer vergleichenden Einfuhrung in die Hill-Climbing-und Zufallsstrategie*. Birkhauser, 1977.

[56] Frank Hoffmeister and Thomas Back. "Genetic self-learning." In: *Proceedings, 1st Euro. Conf. on Artificial Life. The MIT Press, Cambridge, MA*. 1992, pp. 227–235.

[57] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. "Optimization by Simulated Annealing." In: *Science* 220.4598 (1983), pp. 671–680.

[58] Robert McGill, John W. Tukey, and Wayne A. Larsen. "Variations of Box Plots." In: *The American Statistician* 32.1 (1978), pages.

[59] Salvador García et al. "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." In: *Journal of Heuristics* 15.6 (Dec. 2009), pp. 617–644. ISSN: 1381-1231.

[60] Joaquín Derrac et al. "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms." In: *Swarm and Evolutionary Computation* 1.1 (2011), pp. 3–18.

[61] David Shilane et al. "A general framework for statistical performance comparison of evolutionary computation algorithms." In: *Information Sciences* 178.14 (2008), pp. 2870–2879.

[62] Stéphane Doncieux, Jean-Baptiste Mouret, Nicolas Bredeche, et al. "Exploring new horizons in evolutionary design of robots." In: *IROS Workshop on Exploring New Horizons in Evolutionary Design of Robots*. 2009, pp. 5–12.

[63] Sylvain Koos, Antoine Cully, and Jean-Baptiste Mouret. "Fast Damage Recovery in Robotics with the T-resilience Algorithm." In: *International Journal of Robotics Research* 32.14 (Dec. 2013), pp. 1700–1723.

[64] Jason Yosinski et al. "Evolving robot gaits in hardware: the hyperneat generative encoding vs. parameter optimization." In: *Proceedings of the 20th European Conference on Artificial Life*. 2011, pp. 11–18.

[65] Vinod K. Valsalam and Risto Miikkulainen. "Modular Neuroevolution for Multilegged Locomotion." In: *Proceedings of the Genetic and Evolutionary Computation Conference GECCO 2008*. New York, NY, USA: ACM, 2008, pp. 265–272.

[66] Peter Nordin and Wolfgang Banzhaf. "An on-line method to evolve behavior and to control a miniature robot in real time with genetic programming." In: *Adaptive Behavior* 5.2 (1997), pp. 107–140.

[67] Jean-Marc Montanier and Nicolas Bredeche. "Embedded Evolutionary Robotics: The (1+1)-Restart-Online Adaptation Algorithm." In: *New Horizons in Evolutionary Robotics*. Ed. by Stéphane Doncieux, Nicolas Bredèche, and Jean-Baptiste Mouret. Vol. 341. Studies in Computational Intelligence. Springer Berlin Heidelberg, 2011, pp. 155–169.

[68] Jason Yosinski et al. "Generating gaits for physical quadruped robots: evolved neural networks vs. local parameterized search." In: *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*. ACM. 2011, pp. 31–32.

[69] Jean-Christophe Zufferey et al. "Evolving Vision-Based Flying Robots." In: *Biologically Motivated Computer Vision*. Ed. by HeinrichH. Bülthoff et al. Vol. 2525. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2002, pp. 592–600.

[70] M Okura et al. "Artificial evolution of FPGA that controls a miniature mobile robot Khepera." In: *SICE 2003 Annual Conference*. Vol. 3. IEEE. 2003, pp. 2858–2863.

[71] Adrian Thompson. *Evolving electronic robot controllers that exploit hardware resources*. Springer, 1995.

[72] Dongbing Gu et al. "GA-based learning in behaviour based robotics." In: *Computational Intelligence in Robotics and Automation, 2003. Proceedings. 2003 IEEE International Symposium on*. Vol. 3. IEEE. 2003, pp. 1521–1526.

[73] Adrian Boeing, Stephen Hanham, and Thomas Braunl. "Evolving autonomous biped control from simulation to reality." In: *Proceedings of the 2nd International Conference on Autonomous Robots and Agents, Palmerston North, New Zealand*. 2004, pp. 13–15.

[74] S. Kernbach et al. "Evolutionary robotics: The next-generation-platform for on-line and on-board artificial evolution." In: *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*. May 2009, pp. 1079–1086.

[75] Joshua E. Auerbach and Josh C. Bongard. "Evolving CPPNs to Grow Three-dimensional Physical Structures." In: *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*. GECCO '10. Portland, Oregon, USA: ACM, 2010, pp. 627–634. ISBN: 978-1-4503-0072-8.

[76] Joshua E Auerbach and Joshua C Bongard. "On the relationship between environmental and morphological complexity in evolved robots." In: *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*. ACM. 2012, pp. 521–528.

[77] Joel Lehman and Kenneth O. Stanley. "Abandoning Objectives: Evolution Through the Search for Novelty Alone." In: *Evol. Comput.* 19.2 (June 2011), pp. 189–223.

[78] Faustino Gomez and Risto Miikkulainen. "Incremental Evolution of Complex General Behavior." In: *Adaptive Behavior* 5 (1997), pp. 5–317.

[79] Hod Lipson. "Principles of modularity, regularity, and hierarchy for scalable systems." In: *In GECCO Workshop on Modularity, Regularity, and Hierarchy in Evolutionary Computation*. 2004.

[80] Nick Jakobi, Phil Husbands, and Inman Harvey. "Noise and the reality gap: The use of simulation in evolutionary robotics." In: *Advances in artificial life*. Springer, 1995, pp. 704–720.

[81] Nick Jakobi. "Evolutionary Robotics and the Radical Envelope-of-Noise Hypothesis." In: *Adaptive Behavior* 6.2 (1997), pp. 325–368.

[82] Gordon Klaus, Kyrre Glette, and Jim Tørresen. "A Comparison of Sampling Strategies for Parameter Estimation of a Robot Simulator." In: *Simulation, Modeling, and Programming for Autonomous Robots*. Ed. by Itsuki Noda et al. Vol. 7628. Lecture Notes in Computer Science. 2012, pp. 173–184.

[83] JuanCristóbal Zagal and Javier Ruiz-del-Solar. "Combining Simulation and Reality in Evolutionary Robotics." English. In: *Journal of Intelligent and Robotic Systems* 50.1 (2007), pp. 19–39.

[84] S. Koos, J.-B. Mouret, and S. Doncieux. "The Transferability Approach: Crossing the Reality Gap in Evolutionary Robotics." In: *Evolutionary Computation, IEEE Transactions on* 17.1 (Feb. 2013), pp. 122–145.

[85] Ethan Tira-Thompson. *Digital Servo Calibration and Modeling*. Tech. rep. Pittsburgh, PA: Robotics Institute, Mar. 2009.

[86] Youngbum Jun, Robert Ellenberg, and Paul Oh. "Realization of miniature humanoid for obstacle avoidance with real-time zmp preview control used for full-sized humanoid." In: *Humanoid Robots (Humanoids), 2010 10th IEEE-RAS International Conference on*. IEEE. 2010, pp. 46–51.

[87] Felix Faber et al. "The humanoid museum tour guide Robotinho." In: *Robot and Human Interactive Communication, 2009. RO-MAN 2009. The 18th IEEE International Symposium on*. IEEE. 2009, pp. 891–896.

[88] Jirg Stuckler, Johannes Schwenk, and Sven Behnke. "Getting Back on Two Feet: Reliable Standing-up Routines for a Humanoid Robot." In: *IAS*. 2006, pp. 676–685.

[89] T.-T. Lee, C.-M. Liao, and T.K. Chen. "On the stability properties of hexapod tripod gait." In: *Robotics and Automation, IEEE Journal of* 4.4 (Aug. 1988), pp. 427–434.

[90] Tom Smith et al. "Fitness landscapes and evolvability." In: *Evolutionary computation* 10.1 (2002), pp. 1–34.

[91] Josh C. Bongard and Chandana Paul. "Investigating morphological symmetry and locomotive efficiency using virtual embodied evolution." In: *From Animals to Animats: The Sixth International Conference on the Simulation of Adaptive Behaviour*. MIT Press, 2000, pp. 420–429.

[92] V.K. Valsalam and R. Miikkulainen. "Evolving Symmetry for Modular System Design." In: *Evolutionary Computation, IEEE Transactions on* 15.3 (June 2011), pp. 368–386.

Appendix A

Calculations

A.1 Force calculations

Stride length is constant between different lengths, given a weight of the robot of 3kg and 3 legs in the ground at all times. This is constant since the max length between coxa servo center and end of tibia is the same.

$$\begin{aligned}
\text{Length}_{\text{max}} &= \text{Length}_{\text{base}\rightarrow\text{coxa}} + \text{Length}_{\text{coxa}\rightarrow\text{tibia}} \\
\text{Length}_{\text{max}} &= 59mm + 183mm \\
\text{Length}_{\text{max}} &= 242mm
\end{aligned}$$

Using the cosine rule and the maximum angle of rotation by the coxa, this gives a maximum stride length of:

$$\begin{aligned}
c^2 &= (242mm)^2 + (242mm)^2 - 2 \times (242mm)(242mm) \times cos(0.81) \\
c^2 &= 58564mm^2 + 58564mm^2 - 117128mm^2 \times cos(0.81) \\
c^2 &= 36368mm^2 \\
c &= 190.7mm
\end{aligned}$$

Femur: 230mm, tibia: 230mm

Minimum length Minimum length between femur servo center and tibia end for femur and tibia lengths of 230mm is given in equation A.1. The law of cosines is used for calculating the distance, and π-2.490 comes from the angle limit of the femur/tibia servo.

$$
\begin{aligned}
c^2 &= a^2 + b^2 - 2ab \times cos(C) \\
c^2 &= (230mm)^2 + (230mm)^2 - 2(230mm)(230mm) \times cos(\pi - 2,490) \\
c^2 &= 52900mm^2 + 52900mm^2 - 105800mm^2 \times (0.795) \\
c^2 &= 21676.416mm^2 \\
c &= \sqrt{21676.416mm^2} \\
c &= 147mm
\end{aligned}
$$

Maximum angle Maximum angle for AX18 servo in femur, given femur and tibia lengths of 230mm, a robot weight of 3kg, and 3 supporting legs:

$$
\begin{aligned}
cos(C) &= \frac{a^2 + b^2 - c^2}{2ab} \\
cos(C) &= \frac{(230mm)^2 + (230mm)^2 - (183mm)^2}{2(230mm)(230mm)} \\
cos(C) &= \frac{52900mm^2 + 52900mm^2 - 33489mm^2}{105800mm^2} \\
cos(C) &= \frac{72311mm^2}{105800mm^2} \\
C &= cos^{-1}(\frac{72311}{105800}) \\
C &\approx 0.818rad \approx 46.9deg
\end{aligned}
$$

This angle is converted into the coordinate system of the servo by the following formulas:

$$
\begin{aligned}
\text{Angle}_{calculations} &= \pi - \text{Angle}_{servo} \\
\text{Angle}_{calculations} &= \pi - 0.818 \\
\text{Angle}_{calculations} &\approx 2.323rad \approx 133deg
\end{aligned}
$$

Femur: 150mm, tibia: 150mm

Length Minimum length between femur servo center and tibia end for femur and tibia lengths of 150mm is given in equation A.1. The law of cosines is used for calculating the distance, and π-2.490 comes from the angle limit of the femur/tibia servo.

$$
\begin{aligned}
c^2 &= a^2 + b^2 - 2ab \times cos(C) \\
c^2 &= (150mm)^2 + (150mm)^2 - 2(150mm)(150mm) \times cos(\pi - 2,490) \\
c^2 &= 22500mm^2 + 22500mm^2 - 45600mm^2 \times (0.795) \\
c^2 &= 8748.00mm^2 \\
c &= \sqrt{8748.00mm^2} \\
c &\approx 94mm
\end{aligned}
$$

Maximum angle Maximum angle for AX18 servo in femur for a femur and tibia lengths of 150mm is calculated in equation A.1.

$$
\begin{aligned}
cos(C) &= \frac{a^2 + b^2 - c^2}{2ab} \\
cos(C) &= \frac{(150mm)^2 + (150mm)^2 - (183mm)^2}{2(150mm)(150mm)} \\
cos(C) &= \frac{22500mm^2 + 22500mm^2 - 33489mm^2}{45000mm^2} \\
cos(C) &= \frac{11511mm^2}{45000mm^2} \\
C &= cos^{-1}(\frac{11511}{45000}) \\
C &\approx 1.31rad \approx 75.2deg
\end{aligned}
$$

This angle is converted into the coordinate system of the servo by the following formulas:

$$
\begin{aligned}
\text{Angle}_{calculations} &= \pi - \text{Angle}_{servo} \\
\text{Angle}_{calculations} &= \pi - 1.31 \\
\text{Angle}_{calculations} &\approx 1.83rad \approx 105.0deg
\end{aligned}
$$

A.2 Forward kinematics

Link	a_i/r_i	α_i	d_i	θ_i
1	59mm	$\pi/2$	-26.33mm	θ_1^*
2	$L_{1\to2}$	0	0	θ_2^*
3	$L_{2\to3}$	0	0	θ_3^*

Table A.1: DH parameter table for the back legs

First link (Coxa):

$$A_1 = \begin{matrix} cos(\theta_1) & -sin(\theta_1)cos(\alpha_1) & sin(\theta_1)sin(\alpha_1) & r_1cos(\theta_1) \\ sin(\theta_1) & cos(\theta_1)cos(\alpha_1) & -cos(\theta_1)sin(\alpha_1) & r_1sin(\theta_1) \\ 0 & sin(\alpha_1) & cos(\alpha_1) & d_1 \\ 0 & 0 & 0 & 1 \end{matrix}$$

$$A_1 = \begin{matrix} cos(\theta_1^*) & -sin(\theta_1^*)cos(\pi/2) & sin(\theta_1^*)sin(\pi/2) & 59mm * cos(\theta_1^*) \\ sin(\theta_1^*) & cos(\theta_1^*)cos(\pi/2) & -cos(\theta_1^*)sin(\pi/2) & 59mm * sin(\theta_1^*) \\ 0 & sin(\pi/2) & cos(\pi/2) & -26.33mm \\ 0 & 0 & 0 & 1 \end{matrix}$$

$$A_1 = \begin{matrix} cos(\theta_1^*) & 0 & sin(\theta_1^*) & 59mm * cos(\theta_1^*) \\ sin(\theta_1^*) & 0 & -cos(\theta_1^*) & 59mm * sin(\theta_1^*) \\ 0 & 1 & 0 & -26.33mm \\ 0 & 0 & 0 & 1 \end{matrix}$$

Second link (femur):

$$A_2 = \begin{matrix} cos(\theta_2) & -sin(\theta_2)cos(\alpha_2) & sin(\theta_2)sin(\alpha_2) & r_2cos(\theta_2) \\ sin(\theta_2) & cos(\theta_2)sin(\alpha_2) & -cos(\theta_2)sin(\alpha_2) & r_2sin(\theta_2) \\ 0 & sin(\alpha_2) & cos(\alpha_2) & r_2 \\ 0 & 0 & 0 & 1 \end{matrix}$$

$$A_2 = \begin{matrix} cos(\theta_2^*) & -sin(\theta_2^*)cos(0) & sin(\theta_2^*)sin(0) & L_{1\to2} * cos(\theta_2^*) \\ sin(\theta_2^*) & cos(\theta_2^*)sin(0) & -cos(\theta_2^*)sin(0) & L_{1\to2} * sin(\theta_2^*) \\ 0 & sin(0) & cos(0) & L_{1\to2} \\ 0 & 0 & 0 & 1 \end{matrix}$$

$$A_2 = \begin{matrix} cos(\theta_2^*) & -sin(\theta_2^*) & 0 & L_{1\to2} * cos(\theta_2^*) \\ sin(\theta_2^*) & 0 & 0 & L_{1\to2} * sin(\theta_2^*) \\ 0 & 0 & 1 & L_{1\to2} \\ 0 & 0 & 0 & 1 \end{matrix}$$

Third link (tibia):

$$A_3 = \begin{matrix} cos(\theta_3) & -sin(\theta_3)cos(\alpha_3) & sin(\theta_3)sin(\alpha_3) & r_3cos(\theta_3) \\ sin(\theta_3) & cos(\theta_3)sin(\alpha_3) & -cos(\theta_3)sin(\alpha_3) & r_3sin(\theta_3) \\ 0 & sin(\alpha_3) & cos(\alpha_3) & r_3 \\ 0 & 0 & 0 & 1 \end{matrix}$$

$$A_3 = \begin{matrix} cos(\theta_3^*) & -sin(\theta_3^*)cos(0) & sin(\theta_3^*)sin(0) & L_{2\to3} * cos(\theta_3^*) \\ sin(\theta_3^*) & cos(\theta_3^*)sin(0) & -cos(\theta_3^*)sin(0) & L_{2\to3} * sin(\theta_3^*) \\ 0 & sin(0) & cos(0) & L_{2\to3} \\ 0 & 0 & 0 & 1 \end{matrix}$$

$$A_3 = \begin{matrix} cos(\theta_3^*) & -sin(\theta_3^*) & 0 & L_{2\to3} * cos(\theta_3^*) \\ sin(\theta_3^*) & 0 & 0 & L_{2\to3} * sin(\theta_3^*) \\ 0 & 0 & 1 & L_{2\to3} \\ 0 & 0 & 0 & 1 \end{matrix}$$

All three links:

$$A_1 * A_2 * A_3$$

Appendix B

Code

B.1 Fitness functions

```
bool MoveEval::onSimulationStep(sim::Simulator*, float next_sim_t)
{
        _score = scenario()->waypointDistance();
        return next_sim_t < 8;
}

bool TotalMoveEval::onSimulationStep(sim::Simulator*, float next_sim_t)
{
        _score = sqrt(pow(abs(scenario()->xDistance()),2)+pow(abs(scenario()->zDistance()),2));
        return next_sim_t < 8;
}

bool SidewaysMoveEval::onSimulationStep(sim::Simulator*, float next_sim_t)
{
        _score = scenario()->xDistance();
        return next_sim_t < 8;
}

bool LeftTurnEval::onInclusion(sim::Simulator* sim)
{
        auto qVector = scenario()->machine->pose().R[2];
        lastBasePlaneVector = (qVector * float3(1,0,1)).normalized();
        return true;
}

bool LeftTurnEval::onSimulationStep(sim::Simulator* givenSim, float next_sim_t)
{
        auto qVector = scenario()->machine->pose().R[2];

        auto basePlaneVector = (qVector * float3(1,0,1)).normalized();

        auto currentRotationMagnitude = cross(basePlaneVector, lastBasePlaneVector).y;
        auto currentRotation = -asin(currentRotationMagnitude);

        _score += currentRotation;
        lastBasePlaneVector = basePlaneVector;

        return next_sim_t < 8;
}

bool TotalWeightEval::onInclusion(sim::Simulator* s)
{
        _score = scenario()->machine->getTotalWeight();
        return true;
}

tonnesfn_machine::getTotalWeight() const{
        float totalWeightToReturn = 0.0f;

        for (int i = 0; i < _body->linkCount(); i++){ // Traverse all links
                for (int j = 0; j < _body->link(i).shapeCount(); j++){ // Traverse all shapes
                        totalWeightToReturn += _body->link(i).shape(j).mass();
                }
        }
        return totalWeightToReturn;
}
```

B.2 Crossover operators

Uniform crossover

```
void uniformCrossover(Tonnesfn_genes a, Tonnesfn_genes b){
        auto testa = a.control->param;
        auto testb = b.control->param;

        for (int i = 0; i < 36; i++){
                if (rng.uniform(0.0f,1.0f) <= evo_parameters().a_m_nonuniform)
                        a.control->param[i] = b.control->param[i];
                if (rng.uniform(0.0f,1.0f) <= evo_parameters().a_m_nonuniform)
                        b.control->param[i] = a.control->param[i];
        }
}
```

B.3 Mutation operators

Non-uniform mutation of non-discrete floats

```
float nonUniformMutation(float numberToMutate) {
        if (rng.uniform(0.0f,1.0f) <= evo_parameters().p_m_nonuniform) {
                numberToMutate += float(rng.normal(0,evo_parameters().a_m_nonuniform));
        }
        return numberToMutate;
}
```

Random reset mutation of non-discrete floats

```
float randomResetMutation(float numberToMutate, float lowerLimit, float upperLimit) {
        if (rng.uniform(0.0f,1.0f) <= evo_parameters().p_m_reset){
                numberToMutate = (float) rng.uniform(lowerLimit,upperLimit);
        }
        return numberToMutate;
}
```

Non-uniform mutation of discrete values

```
float nonUniformMutationOfDiscreteValues(float numberToMutate, float divisionSize){
        float numberToReturn = numberToMutate;
        while (true){
                if (rng.uniform(0.0f,1.0f) > evo_parameters().p_m_nonuniform_p){
                        return numberToReturn;
                }

                if (rng.random(2) == 0){
                        numberToReturn += divisionSize;
                }else{
                        numberToReturn -= divisionSize;
                }
        }
        return numberToReturn;
}
```

Random reset mutation of discrete values

```
float randomResetMutationWithDiscreteValues(float numberToMutate,
                float lowerLimit, float upperLimit, float stepSize){
        if (rng.uniform(0.0f,1.0f) <= evo_parameters().p_m_reset){
                numberToMutate = roundToNearest(
                                (float) rng.uniform(lowerLimit,upperLimit), stepSize);
        }
        return numberToMutate;
}
```

B.4 Controller

MaxMin symmetric controller

```
void MaxMinSymmetryController::update(Controllable* s, float time)
{
        for (auto i : count_to(_pairs.size()))
        {
                const int joint[2] = { _pairs[i].first, _pairs[i].second };
                const auto j = i*4;
                const float a = param[j];
```

```
                const float b = param[j+1];

                const float amp = (a-b)/2;
                const float offset = (a+b)/2;
                float phase = 0;

                for (int k = 0; k < 2; ++k)
                {
                        phase += param[j+k+2];
                        float target = amp * tanh(4*sin(2*FPI*(time+phase))) + offset;
                        s->output[joint[k]] = isPositional() ? target
                                        : p_ctrl(target - s->state[joint[k]], K_P, 0.005f);
                }
        }
}
```

B.5 Servo code

```
void TonnesfnMachine::velocityTarget(int id, float target)
{
        static float sta_f = 0.15f;
        static float noload_vel;
        static float stall_trq;

        if (id == 1 || id == 4 || id == 7 || id == 10 || id == 13 || id == 16){
                // AX18
                noload_vel = 10.15f; //97rpm = 10.15rad/s
                stall_trq  = 1.8f;   //Nm
        }else{
                // AX12
                noload_vel = 6.18f; //59rpm = 6.18/s
                stall_trq  = 1.5f;  //Nm
        }

        static const float vel_cap = 0.3f;
        static const float dyn_f = (stall_trq-sta_f)/noload_vel;

        auto link = _body->link(id+1);
        const float current_pos = link.jointPosition();
        const float current_vel = link.jointVelocity();

        link.jointDrive(0,600*std::max(0.f,noload_vel*vel_cap-fabs(current_vel)));
        link.velocityTarget(clamp(target,-1,1)*noload_vel*vel_cap,0,0);
}
```

B.6 Learning algorithms

One plus lambda

(This code was not written as part of this thesis, but is included for documentation)

```
void OnePlusLambda::evaluate(float score)
{
        if (auto current = param())
                {
                textlog << format("score:␣%.3f␣", score);
                textlog << format("(%.3f,␣%.3f,␣%d)␣", _best_score, calc_gms(_best_sigma)
                                , _agenda.size()-1);
                if (_reign == 0)
                {
                        _best_score = !_finite(_best_score) ? score
                                : (_best_score*(_reevalc) + score)/(_reevalc+1);
                        ++_reevalc;
                        if (_reevalc > 1)
                                textlog << format("mean␣score␣%.3f␣(%d)␣", _best_score, _reevalc);
                        _best_param = _agenda.back().param;
                        _best_sigma = _agenda.back().sigma;
                }
                else if (_best_score < score)
                {
                        textlog << "new␣best!␣";
                        _best_score = score;
                        _best_param = _agenda.back().param;
                        _best_sigma = _agenda.back().sigma;
                        _reign = 0;
                        _reevalc = 1;
                }
                _agenda.pop_back();
                ++_reign;
                if (_agenda.empty()) // new generation
                {
                        if (_reign >= _max_reign)
                        {
```

```
                                textlog << "reeval ";
                                for (auto& s : _best_sigma)
                                        s *= 0.95f;
                                _reset();
                        }
                        else
                        {
                                _agenda.resize(_lambda);
                                for (auto& e : _agenda)
                                {
                                        e.param = _best_param;
                                        e.sigma = _best_sigma;
                                        float gmean = 1;
                                        for (auto& sp : both(e.sigma,e.param))
                                        {
                                                gmean *= sp.first *= (float)exp(rng.normal(0.5f));
                                                sp.second += (float)rng.normal(sp.first);
                                        }
                                        gmean = pow(gmean,1.f/e.sigma.size());
                                }
                        }
                }
                textlog << std::endl;
                param(_agenda.back().param);
        }
}
```

Simulated annealing

(This code was not written as part of this thesis, but is included for documentation)

```
void SimulatedAnnealing::evaluate(float score)
{
        if (auto current = param())
        {
                textlog << format("score: %.3f (%.3f) ", score, _best_score);
                ++_iteration;
                const float temp = 1.0f/_iteration;
                const float sigma = SCALE*sqrt(temp);
                textlog << format("#%d - sigma: %.5f ", _iteration, sigma);
                if (_best_score <= score)
                {
                        _best_score =  score;
                        _best_param = *current;
                        textlog << "better ";
                }
                else
                {
                        const float pA = exp( (score-_best_score)/(temp*0.5f) );
                        textlog << format("pA: %.3f ", pA);
                        if (pA > rng.uniform())
                        {
                                _best_score =  score;
                                _best_param = *current;
                                textlog << "exchange ";
                        }
                        else
                                textlog << "keep ";
                }
                param_vector next = _best_param;
                for (auto& v : next)
                        v += (float)rng.normal(sigma);
                param(next);
                textlog << "\n";
        }
}
```

B.7 Gait generation script

B.7.1 Matlab main script

```
z_l = -130;
z_h = -30;

stride_length = 50.0;
stride_half_distance_safety_margin = 10;

back_start = 150.0;
side_start = 150.0;
```

```
forward_skew = 50.0;

% --------------- 1: FN BO FN BO FN BO

inverse(side_start,stride_length+stride_half_distance_safety_margin + forward_skew,
                z_l, false, true); %B3, FN
inverse(side_start,-stride_length-stride_half_distance_safety_margin + forward_skew,
                z_h, false, false); %B4, BO

inverse(back_start,stride_half_distance_safety_margin,
                z_l,false,false); % B5, FN
inverse(back_start+stride_length,-stride_half_distance_safety_margin,
                z_h,false,true); % B6, BO

inverse(side_start,stride_half_distance_safety_margin - forward_skew,
                z_l, false, true); %B7, FN
inverse(side_start,-stride_half_distance_safety_margin - forward_skew,
                z_h, false, false); %B8, BO

% --------------- 2: BN FO BN FO BN FO

inverse(side_start,stride_half_distance_safety_margin + forward_skew,
                z_l, false, true); %B3, BN
inverse(side_start,-stride_half_distance_safety_margin + forward_skew,
                z_h, false, false); %B4, FO

inverse(back_start+stride_length,stride_half_distance_safety_margin,
                z_l,false,false); % B5, BN
inverse(back_start,-stride_half_distance_safety_margin,
                z_h,false,true); % B6, FO

inverse(side_start,stride_length+stride_half_distance_safety_margin - forward_skew,
                z_l, false, true); %B7, BN
inverse(side_start,-stride_length-stride_half_distance_safety_margin - forward_skew,
                z_h, false, false); %B8, FO

% --------------- 3: BN FN BN FN BN FN

inverse(side_start,stride_half_distance_safety_margin + forward_skew,
                z_l, false, true); %B3, BN
inverse(side_start,-stride_half_distance_safety_margin + forward_skew,
                z_l, false, false); %B4, FO

inverse(back_start+stride_length,stride_half_distance_safety_margin,
                z_l,false,false); % B5, BN
inverse(back_start,-stride_half_distance_safety_margin,
                z_l,false,true); % B6, FO

inverse(side_start,stride_length+stride_half_distance_safety_margin - forward_skew,
                z_l, false, true); %B7, BN
inverse(side_start,-stride_length-stride_half_distance_safety_margin - forward_skew,
                z_l, false, false); %B8, FO

% --------------- 4: BO FN BO FN BO FN

inverse(side_start,stride_half_distance_safety_margin + forward_skew,
                z_h, false, true); %B3, BO
inverse(side_start,-stride_half_distance_safety_margin + forward_skew,
                z_l, false, false); %B4, FN

inverse(back_start+stride_length,stride_half_distance_safety_margin,
                z_h,false,false); % B5, BO
inverse(back_start,-stride_half_distance_safety_margin,
                z_l,false,true); % B6, FN

inverse(side_start,stride_length+stride_half_distance_safety_margin - forward_skew,
                z_h, false, true); %B7, BO
inverse(side_start,-stride_length-stride_half_distance_safety_margin - forward_skew,
                z_l, false, false); %B8, FN

% --------------- 5: FO BN FO BN FO BN

inverse(side_start,stride_length+stride_half_distance_safety_margin + forward_skew,
                z_h, false, true); %B3, FO
inverse(side_start,-stride_length-stride_half_distance_safety_margin + forward_skew,
                z_l, false, false); %B4, BN

inverse(back_start,stride_half_distance_safety_margin,
                z_h,false,false); % B5, FO
inverse(back_start+stride_length,-stride_half_distance_safety_margin,
                z_l,false,true); % B6, BN

inverse(side_start,stride_half_distance_safety_margin - forward_skew,
                z_h, false, true); %B7, FO
inverse(side_start,-stride_half_distance_safety_margin - forward_skew,
                z_l, false, false); %B8, BN

% --------------- 6: FN BN FN BN FN BN

inverse(side_start,stride_length+stride_half_distance_safety_margin + forward_skew,
                z_l, false, true); %B3, FO
```

```
inverse(side_start,-stride_length-stride_half_distance_safety_margin + forward_skew,
                z_l, false, false); %B4, BN

inverse(back_start,stride_half_distance_safety_margin,
                z_l,false,false); % B5, FO
inverse(back_start+stride_length,-stride_half_distance_safety_margin,
                z_l,false,true); % B6, BN

inverse(side_start,stride_half_distance_safety_margin - forward_skew,
                z_l, false, true); %B7, FO
inverse(side_start,-stride_half_distance_safety_margin - forward_skew,
                z_l, false, false); %B8, BN
```

B.7.2 Matlab Inverse kinematics function

```
function [array] = inverse(L12, L23, x, y, z, ~, left)

        t1 = atan(y/x);

        p1_x = cos(t1)*59;
        p1_y = sin(t1)*59;
        p1_z = -26.33;

        L13 = sqrt((x-p1_x)^2+(y-p1_y)^2+(z-p1_z)^2);

        t3 = pi-acos((L12^2+L23^2-L13^2)/(2*L12*L23));

        L13xy = sqrt((x-p1_x)^2+(y-p1_y)^2);
        L13z  = p1_z-z;

        t2_m_a = atan(L13z/L13xy);

        t2 = t2_m_a - acos((L12^2+L13^2-L23^2)/(2*L12*L13));

        if left
            t2 = -t2;
            t3 = -t3;
        end

        t1_servo = rad2dyn(t1);
        t2_servo = rad2dyn(t2);
        t3_servo = 1024-rad2dyn(t3);

        if left
            t2_servo = 1024 - t2_servo;
            t3_servo = 1024 - t3_servo;
        end

        array = [t1_servo t2_servo t3_servo];

end
```

B.7.3 Matlab radian to dynamixel function

```
function angle = rad2dyn(givenRadValue)
        angle = round( (512 * givenRadValue / (3.14*(150/180))) + 512);
end
```

Appendix C

Part dimensions and weight

C.1 Femur

Robot	$\text{Length}_{\text{middle}}$	$\text{Length}_{\text{total}}$	$\text{Weight}_{\text{SW}}$	$\text{Weight}_{\text{printed}}$
2		80mm		
2		97mm		
3		103mm		
1	70mm	150mm	29.00g	
3		168mm		
2		170mm		
3	120mm	200mm	38.32g	
-	150mm	230mm	43.97g	35.9g
-	174mm	254mm	48.5g	

Table C.1: Weights of simulated or printed femurs.

C.2 Tibia

Iteration	$\text{Length}_{\text{end}}$	$\text{Length}_{\text{total}}$	$\text{Weight}_{\text{SW}}$	$\text{Weight}_{\text{printed}}$
2		80mm		
2		89mm		
3		100mm		
3		103mm		
1	84mm	150mm	33.96g	
2		163mm		
-	134mm	200mm	40.81g	
-	174mm	240mm	46.22g	40.8g
3	188mm	254mm	48.10g	

Table C.2: Weights of simulated or printed tibias.

Appendix D

Experiment parameters and results

This chapter includes parameters and results for all evolutionary runs, including graphs for the runs evolving turning gaits. It also features graphs of motion capture results, and the results from the learning runs.

D.1 Simulation parameters

Run	Type	Gen	Ind	pr_{ph}	pr_{pos}	p_m_{nu}p	p_m_{nu}f	p_m_{nu}	a_m_{nu}control	a_m_{nu}morph	p_r_u	a_r_u	p_m_{reset}
9-41	C	999	256	0	0	0	0	1	0.3	0	0	0	0
42-65	C	200	512	0	0	0	0	1	0.3	0	0	0	0
66-115	C	200	128	0	0	0	0	1	0.3	0	0	0	0
116-128	C	200	256	0	0	0	0	1	0.3	0	0	0	0
129-154	C	200	256	0	0	0	0	1	0.3	0	0	0	0
155-178	C	200	256	0	0	0	0	1	0.3	0	0	0	0
179	C	200	256	0	0	0	0	1	0.3	0	0	0	0
180-216	C	200	256	0	0	0	0	1	0.3	0	0	0	0
217-252	C	200	256	0	0	0	0	1	0.3	0	1	**0.5**	0
253-274	C	200	256	0	0	0	0	1	0.3	0	1	**0.25**	0
275-291	C	200	256	0	0	0	0	1	0.3	0	1	**0.1**	0
292-295	C	200	256	0	0	0	0	1	0.3	0	0	**0**	0
296-304	C	200	256	0	0	0	0	1	0.3	0	1	**0.05**	0
305-318	C	200	256	0	0	0	0	1	0.3	0	1	**0.02**	0
319-330	C	200	256	0	0	0	0	1	**0.15**	0	0	0	0
331-337	C	200	256	0	0	0	0	1	**0.45**	0	0	0	0
338-357	C	200	256	0	0	0	0	1	**0.6**	0	0	0	0
358-377	C	200	256	0	0	0	0	1	**0.05**	0	0	0	0
378-397	C	200	256	0	0	0	0	1	**0.1**	0	0	0	0
398-417	C	200	256	0	0	0	0	1	**0.025**	0	0	0	0
418-437	C	200	256	0	0	0	0	1	**0.038**	0	0	0	0
438-457	C	200	256	0	0	0	0	1	0.025	0	0	0	**0.02**

Run	Type	Gen	Ind	pr_{ph}	pr_{pos}	p_m_{nu}p	p_m_{nu}f	p_m_{nu}	a_m_{nu}control	a_m_{nu}morph	p_r_u	a_r_u	p_m_{reset}
458-477	C	200	256	0	0	0	0	1	0.025	0	0	0	**0.04**
478-497	C	200	256	0	0	0	1	0.025	0	0	0	0	**0.01**
498-517	C	200	256	0	0	0	1	0.025	0	0	0	0	**0.03**
518-524	C	**1024**	256	0	0	0	1	0.025	0	0	0	0	0.02
525	C	200	256	**0.001**	**0.001**	**0.15**	**3**	0	0	0	0	0	**0.02**
526	C	200	256	**0.001**	**0.001**	**0.15**	**5**	0	0	0	0	0	**0.05**
527	C	200	256	**0.001**	**0.001**	**0.25**	**5**	0	0	0	0	0	**0.05**
528	C	200	256	**0.0001**	**0.0001**	**0.5**	**5**	0	0	0	0	0	**0.05**
529	C	200	256	**0.01**	**0.01**	**0.1**	**5**	0	0	0	0	0	**0.05**
530-549	M+C	200	256	0	0	0	0	1	0.025	0.001	0	0	0.02
550	M+C	200	256	0	0	0	0	1	0.025	0.001	0	0	0.02
551	M+C	200	256	0	0	0	0	1	0.025	0.001	0	0	0.02
552	M+C	200	256	0	0	0	0	1	0.025	0.001	0	0	0.02
553	M+C	200	256	0	0	0	0	1	0.025	0.001	0	0	0.02
554-559	M+C	200	256	0	0	0	0	1	0.025	0.001	0	0	0.02
560-579	M+C	200	256	0	0	0	0	1	0.025	0.001	0	0	0.02
580-599	M+C	200	256	0	0	0	0	1	0.025	0.001	0	0	0.02
600-619	M+C	256	256	0	0	0	0	1	0.025	0.01	0	0	0.02
620-239	M+C	512	256	0	0	0	0	1	0.025	0.01	0	0	0.02
640-656	M+C	1024	256	0	0	0	0	1	0.025	0.01	0	0	0.02
657-659	M+C	512	256	0	0	0	0	1	0.025	0.01	0	0	0.02
660-679	M+C	1024	256	0	0	0	0	1	0.025	0.01	0	0	0.02
680-699	M+C(X)	1024	256	0	0	0	0	1	0.025	0.01	0	0	0.02
700-719	C_2	1024	256	0	0	0	0	1	0.025	0.01	0	0	0.02

Run	Type	Gen	Ind	pr_{ph}	pr_{pos}	p_m_{nu}p	p_m_{nu}f	p_m_{nu}	a_m_{nu}control	a_m_{nu}morph	p_r_u	a_r_u	p_m_{reset}
720-739	C_3	1024	256	0	0	0	0	1	0.025	0.01	0	0	0.02
740-759	C_1	1024	256	0	0	0	0	1	0.025	0.01	0	0	0.02
760-764	C_1(R)	1024	256	0	0	0	0	1	0.025	0.01	0	0	0.02
765-769	C_1(L)	1024	256	0	0	0	0	1	0.025	0.01	0	0	0.02
770-774	C_2(R)	1024	256	0	0	0	0	1	0.025	0.01	0	0	0.02
775-779	C_2(L)	1024	256	0	0	0	0	1	0.025	0.01	0	0	0.02
780-784	C_3(R)	1024	256	0	0	0	0	1	0.025	0.01	0	0	0.02
785-789	C_3(L)	1024	256	0	0	0	0	1	0.025	0.01	0	0	0.02

Table D.1: Parameters used in all simulation runs.

D.2 Simulation results

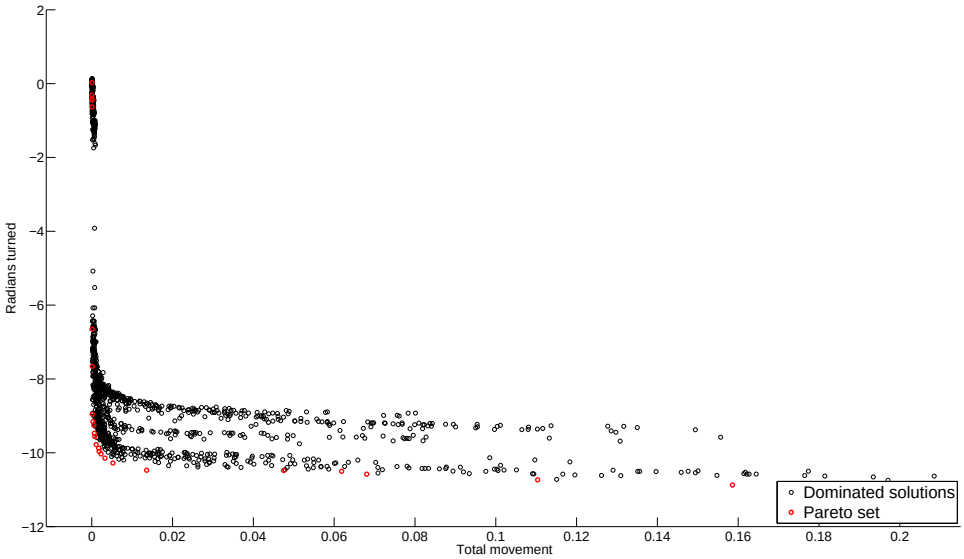
Run	#	$Time_{mean}$	$Time_{total}$	Rank	$Fit._{worst}$	$Fit._{midrange}$	$Fit._{median}$	$Fit._{SD(\sigma)}$	$Fit._{mean}$	$Fit._{mean\ best}$	$Fit._{best}$
9-41	33	350m	8.015d	2	0.8795	1.5250	1.3607	0.2134	1.3093	1.7498	2.1705
42-65	24	177.5m	2.957d	4	0.7409	1.3417	1.1076	0.1740	1.0927	1.5988	1.9426
66-115	50	50.5m	1.736d	6	0.3803	1.1101	0.9168	0.2004	0.9320	1.3099	1.8399
116-128	13	86.5m	0.776d	5	0.6962	1.2163	1.0136	0.1661	0.9992	1.4131	1.7364
129-154	26	49.5m	0.903d	1	1.1424	2.0113	1.4122	0.2073	1.4612	2.2617	2.8802
155-178	24	40.5m	0.667d	3	0.8069	1.4117	1.1591	0.1936	1.1628	1.7371	2.0165
180-216	37	38m	0.976d		0.7414	1.3388	0.9031	0.1322	0.9336	1.5275	1.9363
217-252	36	15m	0.875d		0.6720	1.2150	0.7946	0.1225	0.8281	1.4104	1.7580
253-274	22	33m	0.504d		0.6765	1.4420	0.7918	0.1224	0.8265	1.4618	2.2075
275-291	17	36m	0.425d		0.6710	1.0670	0.7919	0.1149	0.8239	1.3416	1.4631

Run	#	$\text{Time}_{\text{mean}}$	$\text{Time}_{\text{total}}$	Rank	$\text{Fit.}_{\text{worst}}$	$\text{Fit.}_{\text{midrange}}$	$\text{Fit.}_{\text{median}}$	$\text{Fit.}_{\text{SD}(\sigma)}$	$\text{Fit.}_{\text{mean}}$	$\text{Fit.}_{\text{mean best}}$	$\text{Fit.}_{\text{best}}$
292-295	4	37m	0.103d		0.7768	1.2198	0.9146	0.1255	0.9445	1.5343	1.6628
296-304	9	37m	0.231d		0.6710	1.1407	0.7870	0.1190	0.8201	1.3586	1.6104
305-318	14	37.5m	0.364d		0.6740	1.0745	0.7981	0.1220	0.8306	1.3606	1.4751
319-330	12	32.5m	0.270d	4	1.0521	1.5549	1.2785	0.1487	1.2942	1.8727	2.0576
331-337	7	28m	0.137d	5	0.6946	1.1486	0.8134	0.1241	0.8486	1.3966	1.6027
338-357	20	30.5m	0.426d	6	0.6547	1.1103	0.7569	0.1130	0.7908	1.3123	1.5659
358-377	20	30.5m	0.424d	2	1.5301	2.2200	2.0297	0.3294	2.0755	2.3122	2.9099
378-397	20	32m	0.443d	3	1.2464	1.7730	1.6463	0.2008	1.6125	2.0286	2.2995
398-417	20	33.5m	0.468d	1	1.6925	2.2961	2.0204	0.2610	2.0943	2.2343	2.8969
418-437	20	33m	0.458d	2	1.4530	2.2089	1.9661	0.3650	2.0758	2.2707	2.9648
438-457	20	37m	0.514d	2	1.8240	2.4740	2.5711	0.2617	2.4695	2.6454	3.1241
458-477	20	35.5m	0.486d	4	1.8415	2.3345	2.3266	0.2134	2.3048	2.5192	2.8275
478-497	20	34m	0.472d	5	1.7008	2.3312	2.3032	0.2990	2.2506	2.4258	2.9616
498-517	20	35.5m	0.486d	3	1.7605	2.4676	2.3246	0.2938	2.3027	2.5043	3.1748
518-524	7	187.5m	0.909d	1	2.0580	2.5772	2.8052	0.3308	2.6587	2.7813	3.0964
525	1	29m	0.020d	4	0.6932	0.7625	0.7253	0.0324	0.7336	0.8319	0.8319
526	1	40.5m	0.028d	5	0.6886	0.7028	0.6933	0.0054	0.6950	0.7170	0.7170
527	1	33.5m	0.023d	3	1.0330	1.1050	1.0519	0.0227	1.0569	1.1770	1.1700
528	1	32.5m	0.022d	2	1.1729	1.2344	1.2001	0.0277	1.2079	1.2959	1.2959
529	1	39m	0.027d	1	1.5864	1.5876	1.5864	0.0003	1.5865	1.5887	1.5887
530-549	20	39.5m	0.556d	-	0	1.5314	1.5225	0.4707	1.5183	2.2680	3.0629
560-579	20	35m	0.486d	2	1.1536	1.6561	1.6016	0.2259	1.5885	1.7192	2.1586
580-599	20	39m	0.542d	6	0	1.3633	1.0014	0.4261	1.0305	1.7968	2.7665
600-619	20	38.5m	0.528d	5	0.2817	1.5037	1.1540	0.3923	1.1887	1.9338	2.7257

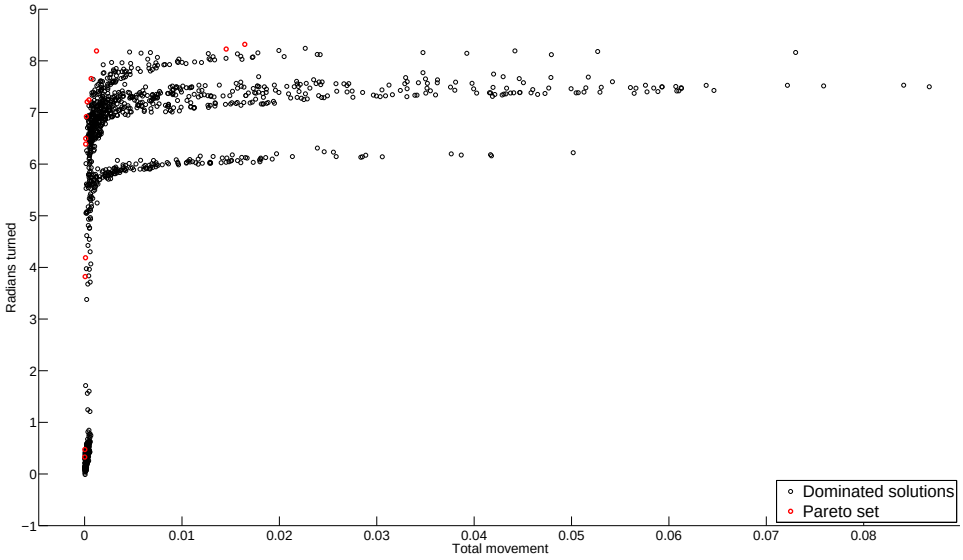
Run	#	$\text{Time}_{\text{mean}}$	$\text{Time}_{\text{total}}$	Rank	$\text{Fit.}_{\text{worst}}$	$\text{Fit.}_{\text{midrange}}$	$\text{Fit.}_{\text{median}}$	$\text{Fit.}_{\text{SD}(\sigma)}$	$\text{Fit.}_{\text{mean}}$	$\text{Fit.}_{\text{mean best}}$	$\text{Fit.}_{\text{best}}$
620-639	20	94m	1.306d	3	0.3834	1.7497	1.5023	0.4581	1.5621	2.4016	3.1160
640-656	17	216m	2.550d	1	0.3868	2.0052	1.6778	0.5223	1.7272	2.7071	3.6236
657-659	3	99m	0.206d	4	0.1695	1.2783	1.2264	0.4339	1.2710	2.2030	2.3871
660-679	20	263m	3.653d	1	0.3488	1.7052	1.3905	0.5066	1.4754	2.4720	3.0616
680-699	20	190m	2.639d	1	0.2845	1.5493	1.4547	0.5598	1.4849	2.4629	2.8141
700-719	20	335.5m	4.653d	3	1.3966	1.6357	1.6858	0.1112	1.6629	1.7329	1.8747
720-739	20	186m	2.583d	1	2.0575	2.3904	2.4012	0.1559	2.4004	2.5152	2.7233
740-759	20	180m	2.500d	2	1.5132	1.9532	1.9804	0.2066	1.9563	2.0597	2.3931
760-764	5	190m	0.660d	-	0.1402	-5.3671	-8.6579	3.8553	-6.7842	-9.9108	-10.8744
765-769	5	202m	0.701d	-	-0.0109	4.1541	6.5750	3.0333	4.9847	7.4967	8.3192
770-774	5	195m	0.677d	-	0.3373	-5.2834	-8.6644	3.4019	-7.2891	-9.9995	-10.9041
775-779	5	186m	0.646d	-	-0.1233	4.4523	7.5380	2.9252	6.2349	8.4297	9.0279
780-784	5	177m	0.615d	-	-0.2474	-6.395	-9.1564	4.1836	-7.2432	-11.0499	-12.5419
785-789	5	196m	0.618d	-	0.0996	5.9784	7.6429	4.1659	6.4927	11.1556	11.8572

Table D.2: Results from all simulation runs.

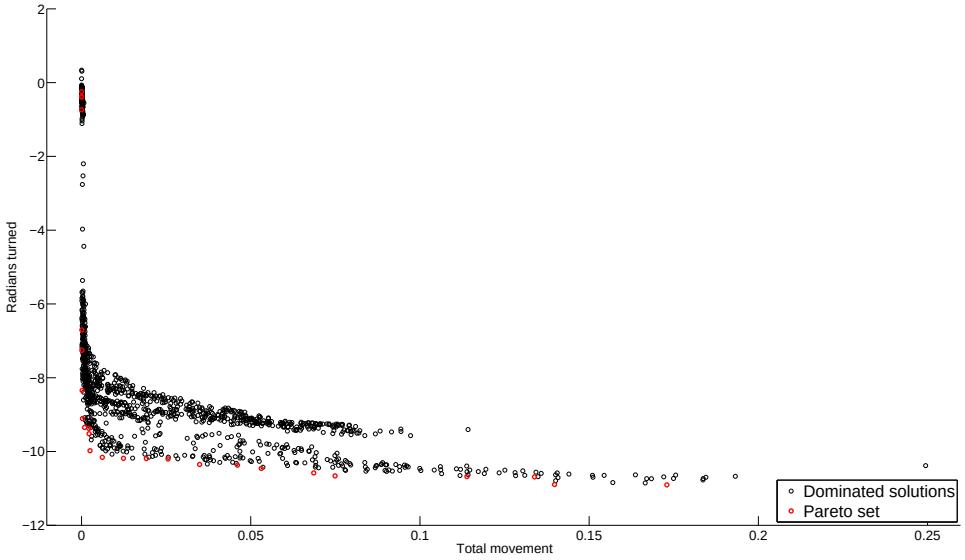
D.3 Turning evolution results



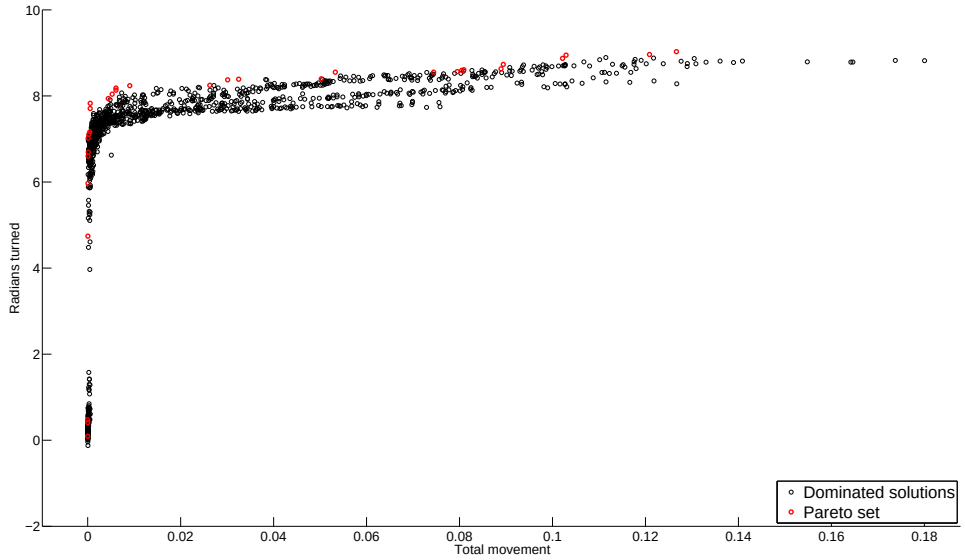
(a) The Pareto front of runs 760-764, evolving turning right for robot 1.



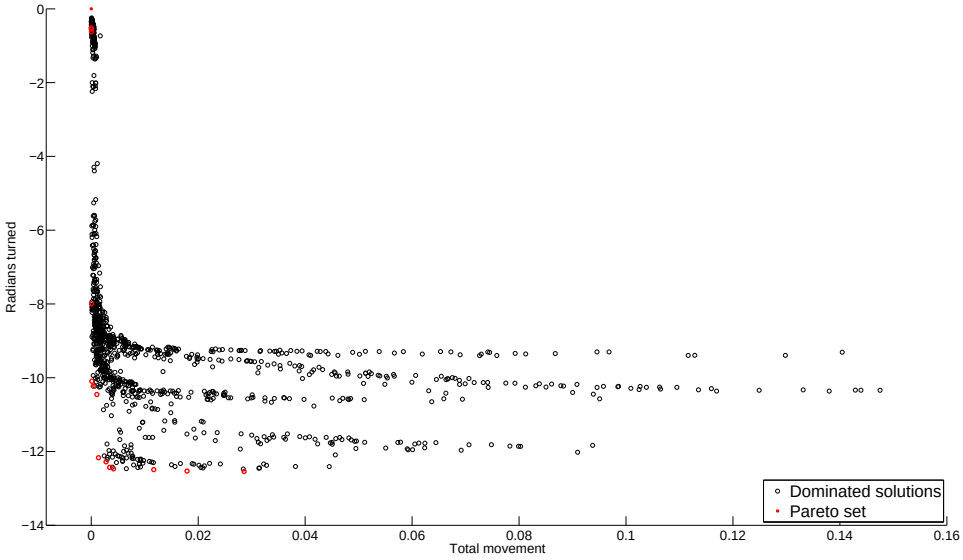
(b) The Pareto front of runs 765-769, evolving turning left for robot 1.



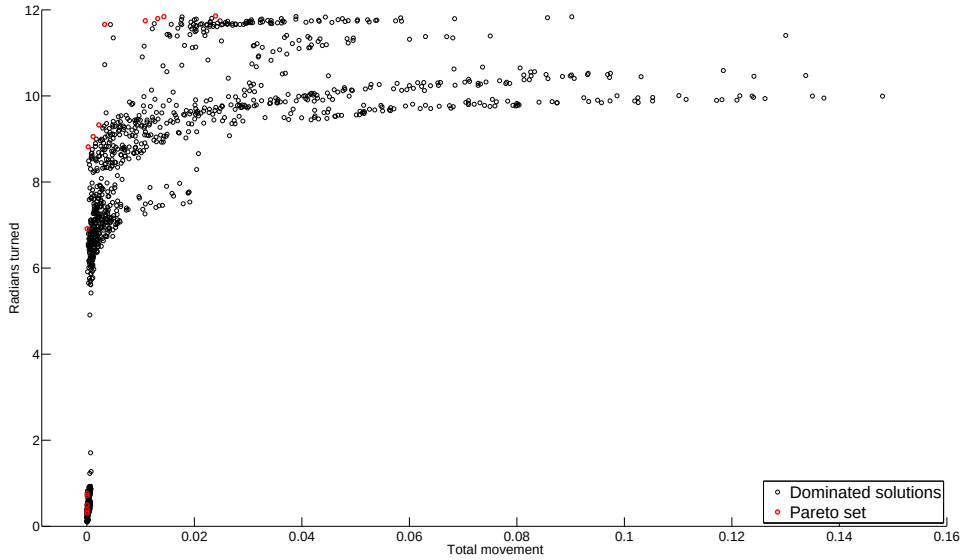
(a) The Pareto front of runs 770-774, evolving turning right for robot 2.



(b) The Pareto front of runs 775-779, evolving turning left for robot 2.



(a) The Pareto front of runs 780-784, evolving turning right for robot 3.



(b) The Pareto front of runs 785-789, evolving turning left for robot 3.

D.4 Motion capture graphs

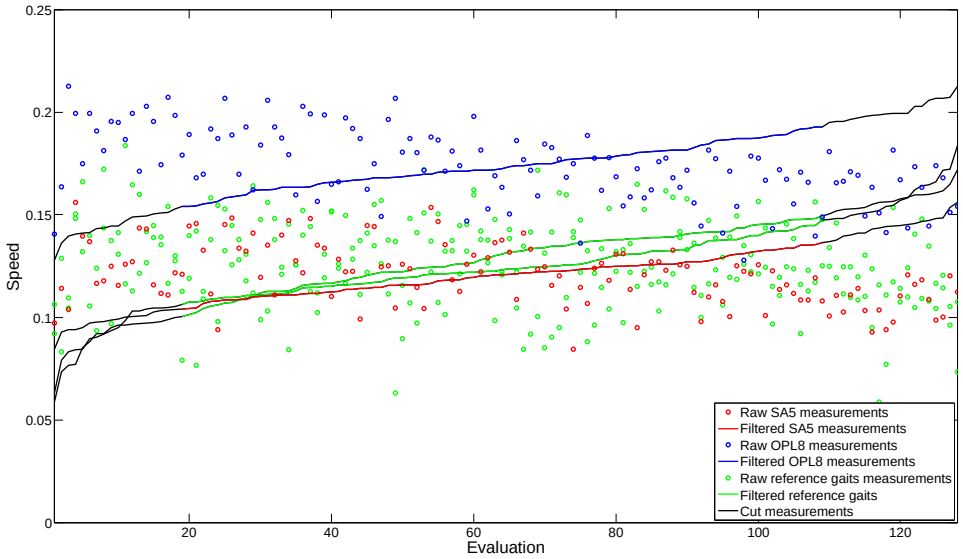


Figure D.4: Graph showing the motion capture recordings of all gaits run on robot1.

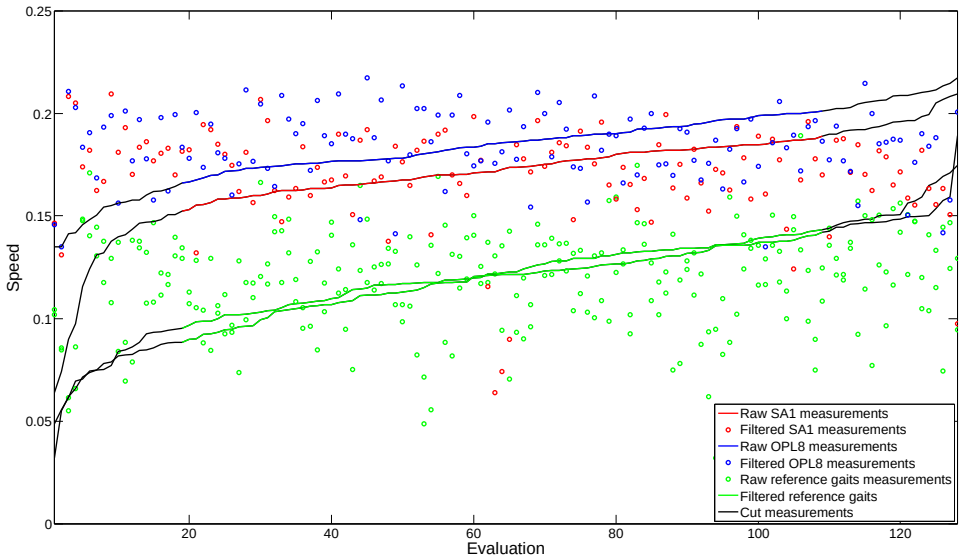


Figure D.5: Graph showing the motion capture recordings of all gaits run on robot2.

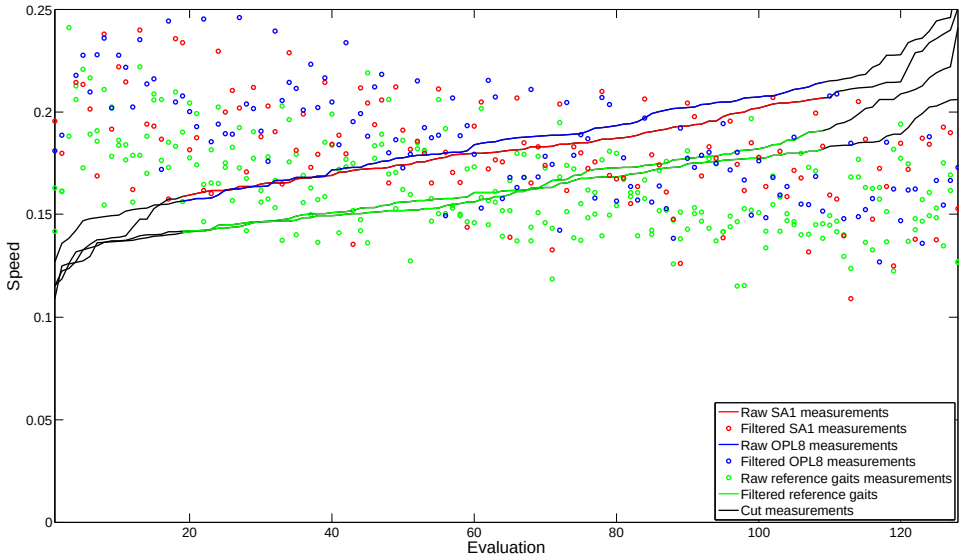


Figure D.6: Graph showing the motion capture recordings of all gaits run on robot3.

D.5 Learning run result tables

	1	2	3	4	5	6	7	8
Max	0.213	0.177	0.199	0.207	0.171	0.210	0.209	0.202
Last	0.185	0.136	0.169	0.154	0.171	0.177	0.152	0.176

Table D.3: Table showing the results of each run of the one plus lambda learning algorithm on robot1.

	1	2	3	4	5	6	7	8
Max	0.161	0.127	0.187	0.198	0.164	0.169	0.159	0.170
Last	0.140	0.124	0.173	0.168	0.139	0.149	0.130	0.167

Table D.4: Table showing the results of each run of the simulated annealing learning algorithm on robot1.

	1	2	3	4	5	6	7	8
Max	0.184	0.207	0.191	0.190	0.163	0.162	0.183	0.199
Last	0.142	0.201	0.146	0.127	0.135	0.146	0.150	0.199

Table D.5: Table showing the results of each run of the one plus lambda learning algorithm on robot2.

	1	2	3	4	5	6	7	8
Max	0.188	0.143	0.106	0.118	0.175	0.172	0.179	0.180
Last	0.180	0.124	0.096	0.114	0.174	0.172	0.179	0.177

Table D.6: Table showing the results of each run of the simulated annealing learning algorithm on robot2.

	1	2	3	4	5	6	7	8
Max	0.234	0.217	0.236	0.235	0.248	0.204	0.240	0.240
Last	0.160	0.191	0.161	0.152	0.117	0.151	0.152	0.172

Table D.7: Table showing the results of each run of the one plus lambda learning algorithm on robot3.

	1	2	3	4	5	6	7	8
Max	0.254	0.247	0.236	0.234	0.240	0.245	0.198	0.245
Last	0.222	0.213	0.198	0.202	0.205	0.188	0.186	0.191

Table D.8: Table showing the results of each run of the simulated annealing learning algorithm on robot3.