UiO : Department of Informatics
University of Oslo

# Filling The Reality Gap

## Using Obstacles to Promote Robust Locomotion for a Quadruped Robot

Andreas Leret Johnsen

Master's Thesis Spring 2014

# Filling The Reality Gap

Andreas Leret Johnsen

12th May 2014

# Abstract

One of the biggest challenges when developing a robot using legs for locomotion is the design of an effective gait. Initially the gaits were developed manually by people, but this is a very time consuming process, and for unconventional robots, the task may be extremely difficult for human engineers to complete.

Evolutionary Algorithms have been successfully utilized to evolve gait that work effectively, and have made the process significantly more autonomous. However, the process is still very time-consuming when testing every gait on a physical robot. This greatly limits the amount of gaits tested, thus limiting the possibility of finding an effective gait.

With the processing power available today, simulators are being used more and more. A simulator allows a significant speed up in the testing of different gaits. However, there is always a difference between the simulator and the real world, and the biggest problem with using simulators when evolving gaits is the so called Reality Gap.

In this thesis, the inclusion of obstacles in the simulator is suggested, to introduce noise in the simulator. The obstacles create a more complex environment for the robot to traverse, in an attempt to force the evolution of more robust gaits.

In the following experiments, several simulator runs have been executed. These have been executed in environments both with, and without, obstacles. A selection of the best gaits has been selected for testing on the physical robot for comparison. The gaits evolved in the environment with obstacles have been compared with the gaits evolved in the flat environment.

In the simulator, the flat environment produces significantly faster gaits than the environment with obstacles. However, in the first physical experiment on the real robot, the gaits evolved in the environment with obstacles produced significantly better results. The average gait produced in the environment with obstacles, was about 25% better than the average gait from the flat environment. In the second physical experiment, the relative reality gap was still smaller for the gaits evolved using obstacles, but the difference was not statistically significant.

# Contents

# List of Figures

# List of Tables

# Acknowledgements

I would like to sincerely thank my supervisor, Associate professor Kyrre Harald Glette, for his support during the work on this thesis. Also, I would like to thank Ph.D. candidate, Eivind Samuelsen, for his support and invaluable help with the simulator.

I also want to thank my fellow students at the lab and the other people at the Robin group for creating a very good study and social environment, especially the lunchtime quizzes.

Last, but not least, I would like to thank my family and friends, especially my significant other, Yliana Sandvold Syversen, and my mother, for continued support throughout the whole process.

Andreas Leret Johnsen,
12th May 2014

# Chapter 1

# Introduction

This chapter will give an overview of the motivation and goals for this thesis, together with an overview of the thesis.

## 1.1 Motivation

For a robot that uses legs for locomotion, it is essential for the robot to have a good gait. Compared to a robot using wheels or tracks for locomotion, it is a significantly more complex process to obtain a satisfying way to move. A wheeled robot can often be effective with only one motor, whereas a legged robot often have to use one motor per joint. This is because each joint need to be controlled individually, making the process of controlling the robot much more complex than controlling a wheeled robot.

So why use legged robots, when a significantly less complex alternative exists? The legged robots have at least one big advantage compared to wheeled robots; the ability to handle significantly rougher terrain [32]. Legged robots can walk through terrain where wheeled robots would be forced to stop, or get stuck [34]. Legged robots can e.g. climb up vertical walls, if they are not too tall, and clear various obstacles. Naturally, the ability to handle different terrain depends on the robot.

However, to handle different terrain, or to even move at all, the robot need an effective gait that utilizes the robot's strengths. Gaits may be developed manually, but this is a very time consuming task [32], and does not guarantee that a good gait is obtained. This is one of the reasons why, in research, the design of gaits is often done by using computers. Computers have the possibility to develop different gaits significantly faster than any human can, and since this process can be done automatically, it is possible to work on other tasks simultaneously.

However, even when using a computer to develop the gaits, the possibilities might be endless for developing a gait. And while a person may be able to easily remove a significant number of the gaits, the computer will only do as it is programmed to do. If the computer is programmed to use exhaustive search, it will check every possible solution, and that will be too time consuming.

This is where Evolutionary Robotics enters the picture. Evolutionary Robotics is the use of Evolutionary Computing in robotics for developing the controller or the morphology of a robot [27]. The use of evolutionary computing has the potential to obtain good gaits significantly faster than an exhaustive search algorithm. It also has the potential to find a solution not likely found by a human [10]. Another reason to use evolution, is that evolved gaits are often better than gaits designed by human engineers [4]. A more detailed description of Evolutionary Computing and Evolutionary Robotics is given in chapter 2.

The use of a simulator together with Evolutionary Robotics is becoming more common [36], and this creates a new challenge. A simulator will never be able to perfectly simulate the real world, creating the so called 'Reality-gap' between the simulator and the real world [16].

## 1.2 Goals

The goal of this thesis is to explore one possibility of reducing the reality gap. The possibility explored is adding noise to the simulator, in the shape of obstacles. The hypothesis is that the obstacles will give the robot a more complex environment to traverse, forcing the algorithm to evolve a more robust gait for the robot. In [14], a similar hypothesis was tested by using the AIBO robot. Gaits were evolved on a surface with plastic poles used as obstacles, but this experiment did not use a simulator for development of the gaits.

Gaits were evolved both in a flat environment, and in several environments including obstacles. The results were then compared, in the simulator and on the physical robot, to examine if including obstacles in the environment can help create more robust gaits.

## 1.3 Overview

The thesis consists of 6 chapters. The chapters are Introduction, Background, Tools & Equipment, Implementation, Experiments & Results and Discussion.

Chapter 2 gives an overview of the field of Evolutionary Computing, focusing on the Genetic Algorithm. It also gives an overview of the field of Evolutionary Robotics, and research done in Evolutionary Robotics. Chapter 3 gives an overview of the tools and equipment used to conduct the experiments and to process the results from the experiments.

Chapter 4 explains the process of setting up the experiments that have been conducted. It gives an overview of the simulator framework used in the simulations, and how the robot has been modelled in the simulator framework. It also explains the kind of Genetic Algorithm used in the evolution, and how this has been set up for the experiments.

Chapter 5 gives a description of each of the experiments conducted, and the results from these experiments, before these results are discussed. Chapter 6 contains an overall discussion of the results found in Chapter 5, together with a conclusion and suggestions for future work.

# Chapter 2

# Background

This chapter will give an overview over different fields related to the later experiments. This includes Genetic Algorithms, Simulation and previous and current work on Evolutionary Robotics.

## 2.1 Evolutionary Computing - Genetic Algorithms

Evolutionary Computing has many uses, one of them is the use in Evolutionary Robotics. Evolutionary Computing is inspired from the evolution found in nature (called biological evolution), where living creatures evolves to adapt to its environment [9]. Operators found in biological evolution, that are also used in Evolutionary Computing, include Selection, Mutation and Recombination [9]. Evolutionary Computing is also used to improve upon an earlier solution, evolving the solution, making it better than before.

The algorithms used in Evolutionary Computing are called Evolutionary Algorithms. From here on Evolutionary Algorithms will be shortened to EAs. An EA consists of a 'population' of solutions that is altered to improve the population, and to find the best possible solution in a given time. The population goes through several cycles to improve the solutions in the population. The operators mentioned before are used to achieve this improvement in the solutions. These cycles consist of 3 main components [9]:

- Parent Selection
- Variation (Mutation and Recombination)
- Survivor Selection

The 3 main components resemble the 3 main parts of biological Evolution, namely inheritance, variation and selection. Other important components of an EA, are the initial population, representation (how the different individuals/solutions are defined), evaluation function and termination condition. Figure 2.1 shows a generic Evolutionary Algorithm.

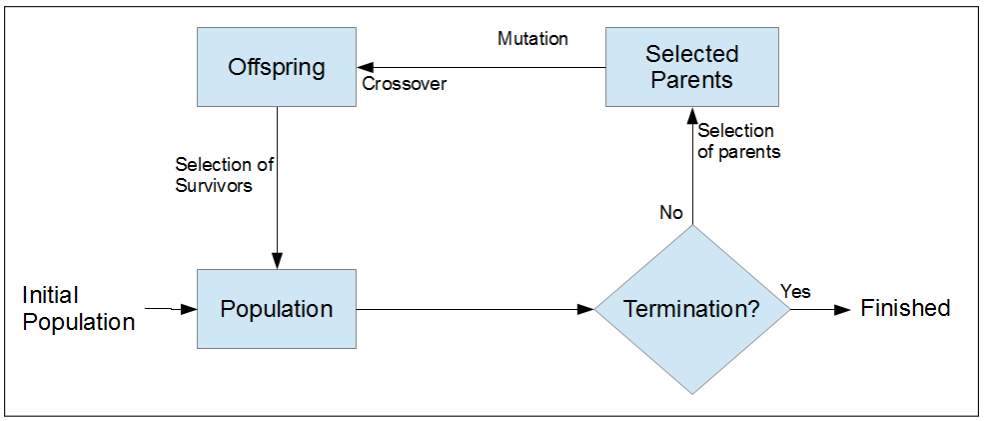



Figure 2.1: Flow chart of a generic Evolutionary Algorithm

EAs have 4 subareas [9], or variants, called Evolutionary Programming(EP), Evolution Strategies(ES), Genetic Algorithms(GA) and Genetic Programming(GP). The main differences between these 4 variants are how the population is represented, and how the variation operators are used. Later sections will focus on one of these variants; the Genetic Algorithm. In the following subsections the different steps and components of the GA will be briefly explained.

### 2.1.1 Representation of individuals in the population

The representation of the population is an important factor in an EA. It is also the first step in creating the GA. The representation creates a connection between the 'real world', and the 'digital world'. The solution in the real world is called the Phenotype, and the representation of the solution within the GA is called the Genotype. This means that to design the representation of the GA, is the same as creating a set of Genotypes to represent the Phenotypes.

The 4 different variations of EAs use different data structures to represent the Genotype. These are shown in Table 2.1.

| **Type of EA** | **Type of Representation** |
|---|---|
| `Genetic Algorithm` | Bit-strings, integer, real-valued vectors, permutation representation |
| `Evolution Strategies` | Real-valued vectors |
| `Evolutionary Programming` | Real-valued vectors |
| `Genetic Programming` | Tree structures |

Table 2.1: The different types of EAs, and the data structures used for representing the genotypes

When selecting what type of EA to use, it is important to choose a type that fits the problem at hand. A simple example is the travelling salesman problem. The travelling salesman problem consists in finding the shortest route between several locations, visiting each location only once, and returning to the starting location when finished travelling. This type of problem is very suitable for the permutation representation, thus making the GA a very suitable variation of the EA. The type of representation also affects how the recombination and mutation operator can be used. This will be discussed more in section 2.1.5.

### 2.1.2 Population

The population contains all the solutions, or individuals, in the GA. The solutions in the population are ranked using the fitness evaluation function. When the algorithm is running, the population is altered by replacing some of the solutions with new, improved, solutions. The variation operators(recombination and mutation) alter the solutions directly. The selection operators(parent and survivor selection) alter the population. This is done by selecting which solutions make it into the next generation.

The initial population in a GA is often randomly generated to simplify the initialization. In addition to simplicity, random generation often produce diversity in the solutions, even if this is not guaranteed. When the initialization is random, it may affect the performance of the algorithm. For example, if the initial population has many good solutions that are not optimal, this can lead to the GA getting stuck in a local optimum, thus stopping the evolution from getting any better results. A local optimum is a solution that is better than all nearby solutions in the solution space, but still not the best solution in total.

When initializing the population, it is also possible to use problem-specific heuristics to initialize the population. This is to create an initial population with higher fitness than when creating the population randomly. This type of initialization may create better initial solutions, but takes extra time to implement. This also means using extra computational power when initializing. This type of initialization may favor some part of the search space, creating a bias, thus limiting the potential for exploration of the search space.

### 2.1.3 Fitness Function

The fitness function is used to evaluate, or rank, the different solutions in the population according to how good the solutions are. Basically, the fitness function's task is to measure the quality of the solutions in the population.

The fitness function is an essential part of the GA. It is critical that the fitness function does not favor the best solutions too much. If the best solutions are favored too much, the potential of exploring the search space may be reduced significantly. If local optima are given too high fitness values, these will be preferred over worse solutions, making it harder for the algorithm to escape local optima. To make sure every solution has a small chance of being selected, fitness scaling may be used [19]. Fitness scaling gives every solution a probability($P > 0$) to be selected, but the better solutions are given a higher probability than the worse solutions.

For tasks with a single objective, for example to find the shortest distance to a goal, the fitness function is pretty straight-forward to implement. However, for a multi-objective task, e.g. to find a both cheap and luxurious car, one has to weigh the two objectives against each other. Often a compromise between the two has to be found, since cheap cars are rarely luxurious and vice versa. For tasks with more than 2 objectives this becomes even more complex.

### 2.1.4 Selection of parents

The parent selection operator selects which solutions to use when creating the new, evolved solutions. The parent selection should favor the highest ranked solutions (the solutions with the highest fitness) when selecting parents. As mentioned in the subsection about the Fitness Function, it is often advantageous to give all solutions a chance of being selected, maintaining exploration of the search space. One method to achieve this is to use the aforementioned fitness scaling. The scaling can be done proportional to the rank of the solution, or proportional to the fitness of the solution. In the latter, one solution that is much better than the rest, can get a much higher fitness than the other solutions, making the algorithm prone to getting stuck in a local optima.

### 2.1.5 Variation operators

The individuals selected to be parents undergo a process called variation. The variation process may include two types of variation operators called recombination and mutation. In most GAs both of these are used. The main difference between the two is that mutation is a unary operator, and recombination is a binary operator. This means that mutation has only one input, while recombination has, at least, two.

#### Recombination

The binary variation operator, recombination, is also often called crossover. Recombination is usually applied on two selected parents at a time, merging information from the two parents. The principle is simple: you take two solutions with different features, and combine them into one or two new offspring. This is very similar to how two biological parents can create an offspring. The recombination operator is entirely stochastic, meaning that the parts selected from the two parents are selected randomly. The type of recombination used is dependent on what type of representation is used for the GA. If a permutation representation is used, the recombination operator must make sure that each number in the representation occurs only once.

#### Mutation

The mutation operator is applied to only one solution. The operator modifies the solution directly, and it is often only a slight modification of the solution. The mutation operator is, like the recombination operator, always stochastic, meaning that the result of the mutation is the result of a series of random changes. Because GAs can have different representations, this also affects how the mutation is implemented. E.g. when using permutation representation, the mutation operator must make sure that each number in the representation only occur once, just like for the recombination operator. Mutation is often done after recombination.

### 2.1.6 Selection of Survivors

The survivor selection operator is very similar to the parent selection operator, but happens later in a generation of the GA. The survivor selection selects the solutions that will survive into the next generation of the GA. The operator can pick solutions from both the old population and the recently created offspring. Often the best solutions are selected, but the concept of age (how many generations one solution has survived) is also used for survivor selection. If a solution is among the best, but has survived for more than a set number of generations, it might not be selected for survival into the next generation. There are of course many ways to combine best fitness and the concept of age to create the survivor selection. The survivor selection is the replacement step of the GA, where some solutions are replaced with others.

### 2.1.7 Termination condition

The Termination condition of the GA decides when the GA should stop or terminate. There are several possibilities for setting the termination condition. One is to stop the GA when the optimal solution is found (assuming an optimal solution is known). However, there is one problem with using only this condition to terminate. The GA is stochastic, meaning that there is no guarantee for this optimal solution to ever being found. Because of this, the termination condition has to be extended with at least one additional condition. There are several conditions that can be used for this. Some possible conditions that can be used are the following:

- The CPU has used the maximal amount of time allowed
- The GA has reached maximum number of generations/fitness evaluations allowed
- The population has not changed for a set number of generations
- A set number of the best individuals has not changed for a set number of generations

The actual termination condition is a disjunction between finding the optimal solution and one or more of the above conditions (or another condition that guarantees termination). The conditions above may also be used when there is no known optimal solution.

## 2.2 Evolutionary Robotics

Evolutionary Robotics is the use of Evolutionary Computing when developing a controller for an autonomous robot [27]. Another use for Evolutionary Robotics is in the autonomous evolution of the morphology of a robot using Evolutionary Computing algorithms [2] [21], although the main focus is on evolving controllers.

### 2.2.1 Current and Previous work on Evolutionary Robotics

Currently, there is a lot of activity in the field of Evolutionary Robotics. The long-term goal of Evolutionary Robotics is to be able to automatically design, and build, a robot that is optimal for a task, by only specifying the task at hand [8].

There is still a long way to go before this goal is reached, but the field is relatively new. With only approximately 20 years of research [8], many things have already been achieved. In [22], simple robots, that were able to crawl, were autonomously designed and manufactured successfully. In [17], a controller was automatically designed for a simulated insect, making the simulated insect capable of walking quickly and effectively according to a tripod gait [20]. And by using a Multi-Objective Evolutionary Algorithm, a controller for a simulated artificial flying animal was successfully developed, making the artificial animal able to fly in the simulator [26].

Evolutionary approaches has also been used in e.g. Swarm Robotics, were Evolutionary processes were used to improve the behavior [3] and make swarm-bots capable of self-assembly [12]. Swarm Robotics is the use of several, often simpler, robots that works together to complete more complex tasks [31], and it takes inspiration from the insect world.

### 2.2.2 Simulation

The use of a simulator combined with Evolutionary Robotics is becoming more common. The use of a simulator brings several advantages compared to the use of a physical robot for evolving the gaits. The main advantage is the speeding up of the evolution [36]. With enough processing power, a simulator has the possibility to speed up the evolving and testing of gaits [25]. It is even possible to evolve and test several gaits at once when more than one processing core is available.

Another important advantage when using a simulator is that this reduces the physical wear on the robot [10]. Physical wear will, after some use, affect the performance of the robot. In the simulator, this is not a problem, as the robot model may perform identically for an infinite amount of time. For example, in one experiment, the motors used in a robot began to shut down, due to stress, after a number of runs [10].

### 2.2.3 Reality gap

The biggest disadvantage with using a simulator to evolve gaits, is the difference between the simulator and the real world. No matter how accurately the robot and its environment are modelled, there will always be a gap between the simulator and the real world. This gap is called the 'Reality Gap' [35].

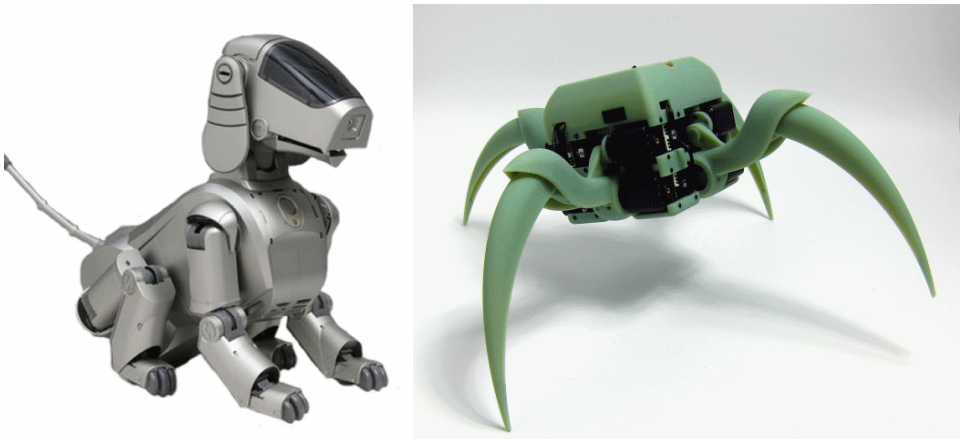
The reason that this reality gap exists is that the real world consists of an unlimited amount of variables, and a computer is never able to simulate all these variables perfectly. This difference makes it more difficult to use simulation to e.g. develop gaits [18]. If the 'Reality Gap' is large enough, the EA may find a kind of 'loophole' in the simulator. This may allow the model in the simulator to do things that are not possible, or ineffective, in the real world.

The 'Reality Gap' may be significantly reduced, but never completely removed. This can be done by modelling the robot and the environment carefully, and adjusting the physics so that it mimics the real world more realistically. However, if too much time is used for this, the speed advantage gained when using a simulator is reduced, and may even be lost completely [15].

Several other approaches for reducing the reality gap have been tested. In one approach, Jakobi suggests a minimal simulation approach. Here only the essential parts needed for the controller development are modelled in the simulator [1]. This approach assumes that the designer of the simulator is successfully able to decide which parts are needed for the simulator. Another approach suggests adding an extra objective to the simulation in addition to the fitness objective, a transferability objective [18]. This approach seeks to prevent the simulator from exploiting the phenomena in the simulator that are not possible in the real world.

### 2.2.4 Platforms used in Evolutionary Robotics

Several robots are used for research in Evolutionary Robotics, for example the commercial available AIBO robot made by Sony [29], and the research-only robots, Aracna [23] and Quadratot [10]. Figure 2.2 shows the Sony AIBO and the Aracna. In [29], two techniques are used for improving both the detection abilities and the walking speed of the AIBO robot. In [13], Evolutionary Algorithms were used for automatically developing gaits for the AIBO and a prototype robot from Sony. This was done by only using the built-in sensors to get feedback for improving the gait. The Aracna is a 4 legged robot, made for the use in Evolutionary Robotics research [23]. The aracna has all 8 motors located in the core of the body, instead of in the legs, giving it a mechanical advantage due to the very light legs.



(a) Picture of the AIBO. [1]

(b) Picture of the Aracna. [2]

Figure 2.2: Pictures of the Aracna and AIBO.

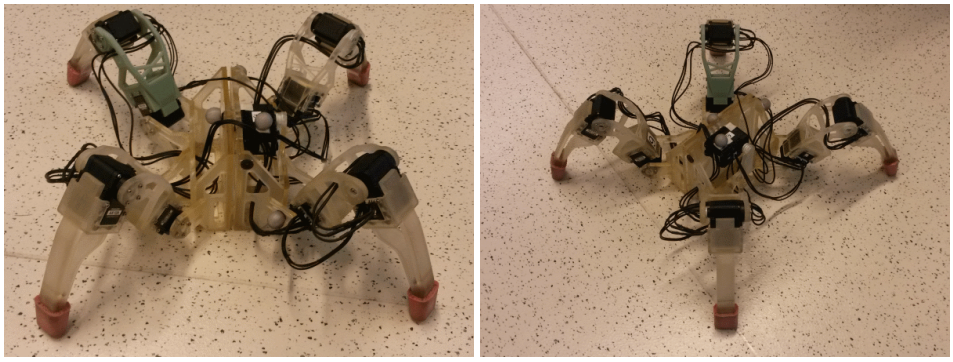
[1] Picture from: *http://www.sony-aibo.com/aibo-models/sony-aibo-ers110/*

[2] Picture from: *http://creativemachines.cornell.edu/sites/default/files/aracna_v2_green.jpg*

# Chapter 3

# Tools & Equipment

## 3.1 The Quadratot



(a) Quadratot from the front (b) Quadratot diagonally from the side.

Figure 3.1: Pictures showing the Quadratot that was used in the experiments conducted in this thesis.

One of many robots used for research on Evolutionary Robotics is the Quadratot. This is the robot used in the following experiments. The Quadratot is a quadruped robot developed at the 'Creative Machines Lab' at Cornell University [33], and is an open hardware platform. The Quadratot used in these experiments was 3D-printed at the Robotics and Intelligent Systems group at the Department of Informatics, University of Oslo. Figure 3.1 shows two images of the Quadratot used in this thesis.

The Quadratot is a quadruped robot, which means that it has 4 legs. Each of these 4 legs has 2 joints. In addition to these 8 joint, there is a joint connecting the two center parts of the robot. The joints are powered by Dynamixel AX-12A and Dynamixel AX-18A servos, manufactured by Korean manufacturer Robotis [24]. The main differences between the 2 servos are that the AX-18A has more torque and is faster than the AX-12A. The Quadratot used for the physical experiments in this thesis uses AX-12A servos for the outer joints of the legs, and AX-18A servos for the inner joints and center joint. Figure 3.2 shows the two servos.

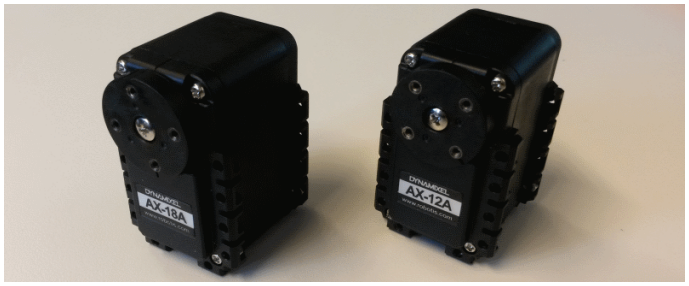



Figure 3.2: Picture showing the Dynamixel AX-12A and Dynamixel AX-18A servos used in the Quadratot.

## 3.2 Motion Capture

In Motion Capture, an object or persons movement is tracked and recorded, often using cameras. It is used in several areas including, but not limited to, Video Game Development, Filmmaking, Medical Applications and Robotics. In Robotics, it is often used to control or measure the movement of one or more robots. In this thesis, Motion Capture is used to measure the distance the physical robot can walk.

### 3.2.1 OptiTrack

The system used for Motion Capture in the later experiments is made by OptiTrack, and the software used is called OptiTrack Arena. The setup used has 8 cameras to track the movement of the robot, and the cameras track 4 reflective markers of the robot. The markers are passive markers, meaning that they are stationary compared to the robot. Figure 3.3 shows the user interface of OptiTrack Arena.

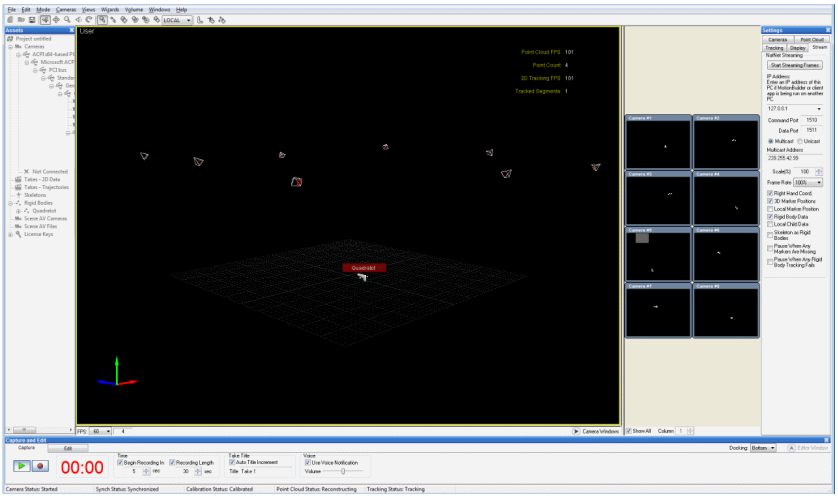



Figure 3.3: A screenshot of the user interface of OptiTrack Arena.

## 3.3 Software

This section gives an overview of the software used.

### 3.3.1 Plots and figures

Matlab was used to make the example plots used in Chapter 4, and all plots of the results in Chapter 5. Matlab is a high level language designed for numerical computation and visualization. OpenOffice draw was used to create Figure 2.1 in Chapter 2, figure **??** in chapter 3 and Figure 4.4 in Chapter 4.

### 3.3.2 Solidworks

SolidWorks was used to measure and weigh the different parts of the Quadratot when modelling it in the simulator. SolidWorks is a 3-dimensional design program, used by a large number of companies around the world. The Quadratot was designed in SolidWorks before it was 3D-printed.

### 3.3.3 Simulator Framework

The evolutionary part of the simulator framework is based on ParadisEO. ParadisEO is a general-purpose software framework used for e.g. fitness analysis [1]. The simulation part of the framework uses the NVidia PhysX physics simulation library to accurately be able to calculate all physics-related data. PhysX was developed by Ageia in 2004, but NVidia got the rights to PhysX after they bought Ageia in 2008. PhysX is one of the most common physics simulation engines used for Video Game physics [28]. A simple figure of the connection between paradisEO and PhysX is shown in figure 3.4.

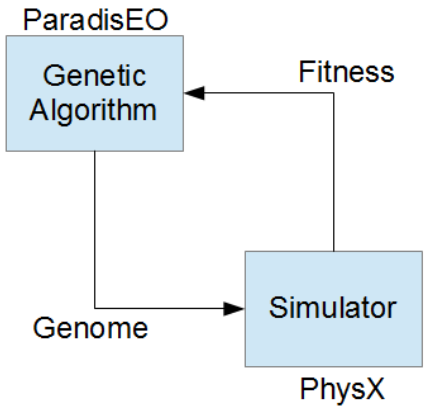



Figure 3.4: Simple figure showing the connection between paradisEO and physX in the simulator framework.

# Chapter 4

# Implementation

In this chapter, the methods used to set up the simulator and Quadratot for the experiments will be presented. First the extensions made in the simulator framework will be described. Then the implementation of the simulator model of the robot and the obstacles will be described. This is followed by a description of the genome and control system. Finally, the genetic operators and fitness measurement used in the GA will be explained.

## 4.1 Extension of the Simulator Framework

A large part of the simulator framework, used in the simulations, was already implemented. The implementation of the simulator can be separated into two main parts:

- The evolutionary part containing the GA
- The simulation part containing the simulator.

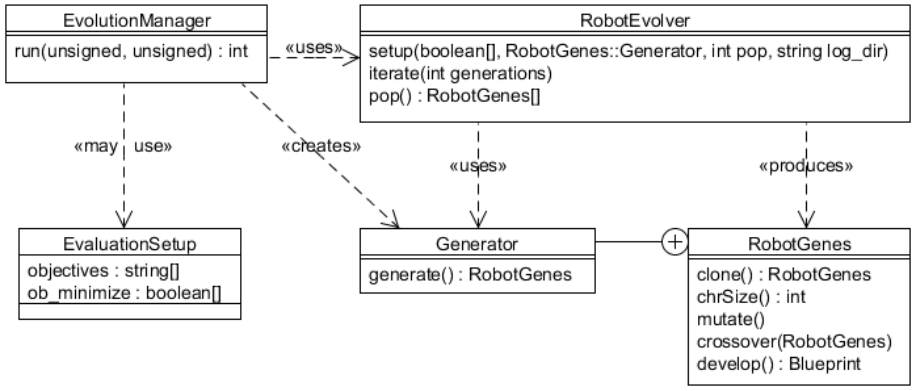



Figure 4.1: A UML-diagram of the Evolutionary part of the simulator framework [7].

Figure 4.1 shows a simple UML-diagram of the extensions made in the evolutionary part of the simulator framework. The different parts shown in the Diagram will now be described, together with extensions made in the simulation part of the simulator framework.

### 4.1.1 Evolutionary part of the simulator

The first of the two main parts is the evolutionary part. This is were the evolution of the gaits is done in the software framework. Most of this part was already implemented, but the objectives for the evolution had to be defined, and the evolutionary operators had to be implemented. The different main objects in the evolutionary part will now be described.

**Evolution Manager** The evolution manager specifies how many generations, and how large the population will be. The objectives used in the Evolution are also defined in the Evolution Manager. This part has to be implemented for each project using the simulator framework, if the objectives are different. The Evolution Manager uses the Evaluation Setup to define the goals for the evolution.

**RobotEvolver** The Robot Evolver was already implemented in the simulator framework. The RobotEvolver executes the Mutation and Crossover operators for the 'genes' of the parameters. As the name implies, the RobotEvolver is the part of the system that does the evolution on the parameters.

**Genes** The genes object of the simulator model was not implemented in the simulator framework. The genes object contains the function for mutation and crossover, and these are used by the RobotEvolver to evolve the parameters. The Genes also contains a generate function that initializes the parameters. This function is also used by the RobotEvolver.

### 4.1.2 Simulation part of the simulator

The other main part is the part where the robot and the environment were implemented. This part contains the simulator models of the Quadratot and the obstacles. It also contains the bbp viewer, the program used to view the simulations, both during, and after the simulation was completed. The different main objects in the simulation part will now be described.

**Blueprint** The blueprint is a description of the Quadratot that is hardware/simulation-independent. The blueprint is important for being able to run the simulated gaits on the hardware implementation of the Quadratot. The blueprint acts as a connection between the software and the hardware implementations of the Quadratot. Each solution has its own blueprint object, and the object is used to connect the evolved gait to the hardware platform.

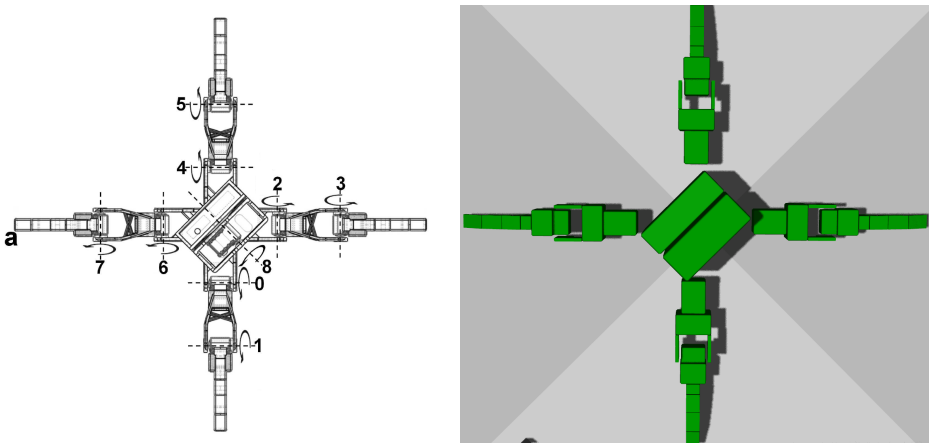
**Machine** Unlike the blueprint, the machine contains the simulator specific implementation of the robot. Here, the actual simulator model of the Quadratot is implemented. A more detailed description of the implementation of the model is found in the subsection called 'Modelling the Quadratot'. Each of the different hardware platforms, that uses the simulator framework, need to have their own implementation of machine.

**bbp Viewer** The bbp Viewer is the program used to view the modelled Quadratot's gaits when finished evolving the parameters. It is called bbp-viewer because the blueprint files produced by the simulator uses '.bpp' as the file extension. All gaits in the last generation are exported into '.bbp'-files. The BBP Viewer was already implemented into the simulator framework.

**Scenario** The scenario defines the environment the model of the Quadratot is evolved in. The obstacles used for evolving with obstacles were implemented in the Scenario file.

## 4.2 Simulator models

Since there was no existing model of the Quadratot in the simulator, this was the first thing that had to be done before one could use the simulator. Also, the obstacles had to be implemented in the simulator. Figure 4.2 shows the Quadratot from above.




(a) Sketch showing the 9 joints of the quadratot from above. [34]

(b) The simulator model of the Quadratot from above.

Figure 4.2: The Quadratot viewed from above.

### 4.2.1 Modelling the Quadratot

The Quadratot model was hard-coded into the simulator. It was created by making several boxes of different dimensions that were moved to their correct place. To find the correct dimensions and placements of the parts, both the model from Solidworks and the real Quadratot was used. The Solidworks parts was mainly used to make more accurate measurements, while the measurements on the real Quadratot was mainly used to measure where the parts are located relative to each other.

The first part modelled was one of the center-parts of the Quadratot, more specifically the part where the servo for the center-joint is located. Since the lower section of this part greatly resembles a box, and the shape of the upper part is of little importance in this model, this part was created using only one box. The same applies to the other center-part, which was modelled next.

The legs were modelled using a function, and since all 4 legs are identical, the function only needed to receive the placement of the legs, and which of the two center-parts it should be connected to, as parameters. The legs consist of several boxes for the inner and outer part of the leg. The legs also have two joints, the one connecting the leg to the center-parts, and the one connecting the inner and outer parts of the leg. Each center part has two

legs connected to it.

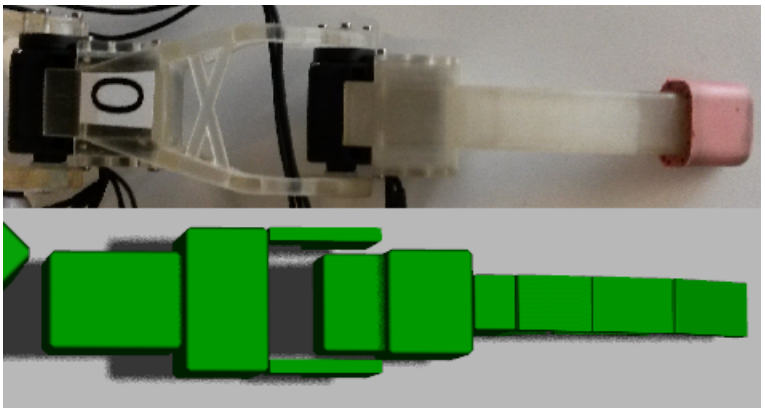


Figure 4.3: Picture of a leg shown from above, both on the real robot and the simulator model.

The image in figure 4.2a is a sketch of the quadratot, showing the 9 different joints. Joint number 1, 3, 5 and 7 in the figure, are the outer joints, from here on called the "knee-joints". The joints 0, 2, 4 and 6 in the figure, are the inner joints, from here on called the "hip-joints". The last joint, joint 8 in the figure, is the center joint. The image in figure 4.2b is a screenshot from the simulator, showing the quadratot from above. Figure 4.3 shows how the legs were modelled in the simulator compared to a real leg.

**Calculating the weight of the Quadratot in the simulator**

For the Quadratot to move more realistically, the Quadratot's weight also had to be entered into the simulator. The weights found in Solidworks, and the weights used in the simulator are shown in Table 4.1.

| Name of part | Weight in SW | Weight in sim. |
|---|---|---|
| `Entire Quadratot` | 606g | 895g |
| `Servo side of center part` | 154g | 228g |
| `Other side of center part` | 172g | 255g |
| `Inner part of leg` | 30g | 44g |
| `Outer part of leg` | 40g | 59g |

Table 4.1: Table showing the weight of the different parts, both the weight in SolidWorks and the weights used in the simulator. None of the 9 servos are included in these weights

To calculate the weight of the different parts, the whole robot was weighed. The weight of the 9 servos was already known to be 55 grams. In the simulator model all servos except the servo in the center was modelled as a separate box. Because of this, the weight of the 8 servos in the legs

was subtracted when calculating the weight of the 3D-printed parts. To calculate the weight of the other parts without disassembling the whole robot, Solidworks was used. Solidworks can calculate the weight of each part, but depending on the material used, this weight is not necessarily correct. To calculate the correct weight, the weights found in Solidworks were used as a relative weight to compare the parts relative to each other, thus making it possible to calculate the real weight of the parts.

### 4.2.2 Modelling the obstacles

The obstacles used in the simulator are a number of boxes in different sizes, spread around a defined area. This defined area is shown in figure 4.4. This area decides where the centers of the boxes are allowed to be placed. The placement of the boxes is decided by a random generator.

The sizes of the boxes are also decided by random generated numbers. The length and width are limited to between 1 cm and 21 cm, and the height is limited between 1 cm and 3 cm.

The random generator's seed is set to a fixed number for each run. This is to make sure the same obstacles can be recreated when the different gaits are tested, after the run is finished.

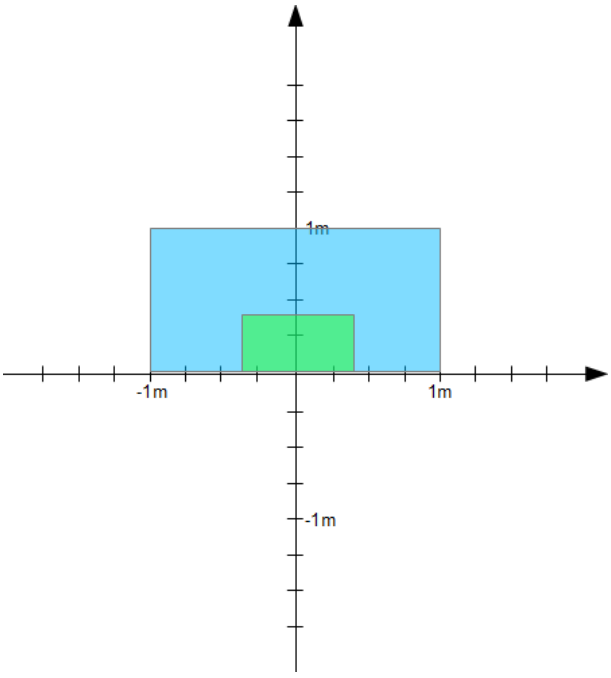



Figure 4.4: Figure showing the area where obstacles could be placed in the simulator. The area where the boxes can be placed is marked with blue, and the green area is an area around the Quadratot's starting position where there are no obstacles.

## 4.3 Genome and Control System

For the genetic algorithm to be able to evolve a gait, the simulator needed a control system for controlling the robot. This section explains how the control system works, and how the genome, used in the evolution, looks like.

### 4.3.1 Control System

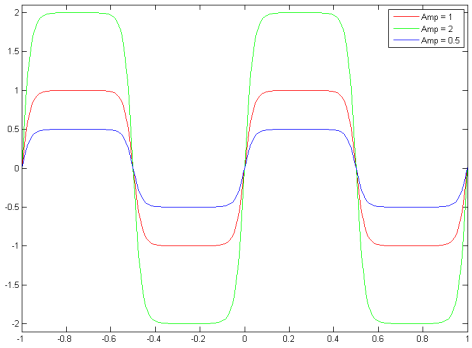
The control system was already implemented, and did not need to be designed or created before use. This control system is called AmpOffPhase, and uses the 3 parameters described over, i.e. Amplitude, Offset and Phase, to control the different joints. The function for the Control System is shown under.

$$\boxed{Amp \times tanh(4 \times sin(2 \times \pi * (time + Phase))) + Offset}$$

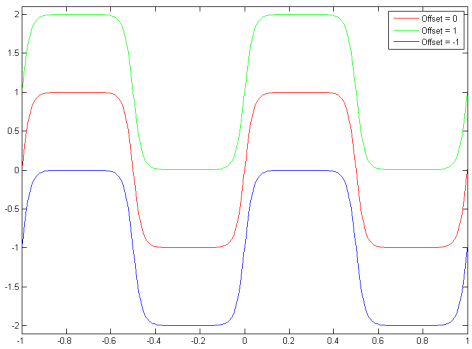
The effects of each of the three parameters, Amplitude, Offset and Phase, are shown in figure 4.3, 4.4 and 4.5 respectively.

The control system moves the joints using a combination of a sine and a hyperbolic tangent. The use of the combination of the sine and the hyperbolic tangent functions means that amplitude controls how much the joints move. This also means that the offset controls the center of the movement, and thus where the movement start and end, and that the phase controls when the movement is done compared to the other joints. Without the phase parameter, all the joints would move at the exact same time, thus making it impossible to create a useful gait for moving in a direction. The function is a periodic function, meaning that the frequency for the function is locked.

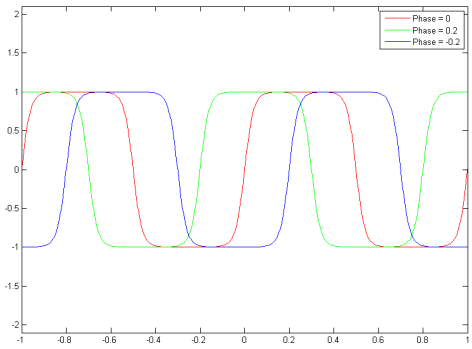
The hyperbolic tangent is added to create the flattening on the top and bottom of the amplitude of the wave, as seen in e.g. figure 4.5a. This makes the movement more suitable to a walking robot, because it enables longer contact with the floor.




(a) The plot shows that the joints movement is bigger when using a larger value for Amplitude.




(b) The plot shows that the offset value determines where the center of the movement is.




(c) The plot shows that the phase value determines when the movement is performed compared to other joints.

Figure 4.5: 4.5a shows the change of joint movement for different Amplitude values, 4.5b shows the shows the change of joint movement for different Offset values, and 4.5c shows the change of joint movement for different Phase values.

### 4.3.2 Representation of genome

For the Control System to evolve the gaits of the Quadratot, the movements of the joints had to be represented. There are, in total, 9 joints on the Quadratot. One center joint, 4 "hip-joints" and 4 "knee-joints". For the Control System used, each of these 9 joints had 3 parameters that could be evolved: Amplitude, Offset and Phase, making the total number of parameters 27. All 27 parameters are represented using single-precision floating-point variables.

All joints were limited to a set range. This was done to limit the evolution of the gaits in the simulator, as the real Quadratot also have limits on the joints. The limits was implemented into the mutation operator, and the center-, hip- and knee-joints had separate limits for each of the 3 parameters. Table 4.2 shows the limits of the joints used in the simulator. The limiting was done by using a clamp function. The clamping function sets all values under the minimum limit to the minimum value, and all values over the maximum limit to the maximum value.

| | **Min.** | **Max** |
|---|---|---|
| `Center joint Amp.(Degrees)` | 0 | 22.5 |
| `Center joint Off.(Degrees)` | -22.5 | 22.5 |
| `Center joint Phase(Degrees)` | -180 | 180 |
| `Hip joint Amp.(Degrees)` | 0 | 45 |
| `Hip joint Off.(Degrees)` | -63.0 | -17.2 |
| `Hip joint Phase(Degrees)` | -180 | 180 |
| `Knee joint Amp.(Degrees)` | 0 | 45 |
| `Knee joint Off.(Degrees)` | 61.4 | 119 |
| `Knee joint Phase(Degrees)` | 180 | 180 |

Table 4.2: Table showing the limits for each parameter of the different joints in the simulator.

Looking at table 4.2, the knee phase limit, shows a limit that allows only one value. This is not a written error, but a bug that was not discovered before the main experiments were finished. More about this bug is found in section 5.5.

## 4.4 Genetic Algorithm operators and Fitness Measurement

The genetic algorithm used has the possibility to use both mutation and crossover. Several combinations of the variation operators were tested, including tests without using the crossover operator at all.

### 4.4.1 NSGA-II

A large part of the Genetic algorithm used was already implemented into the simulator framework. E.g. both the parent selection and selection of survivors were already implemented. The type of Genetic Algorithm used was the Non-dominated Sorting Genetic Algorithm II, or NSGA-II for short [6]. The NSGA-II is designed to be a Multi-objective Evolutionary Algorithm, but in the following simulations, only one objective is used.

Initially, a random population is made. This part was not already implemented in the framework. All parameters were set to 0, except the offset for the hip- and knee-joints. These two parameters were set to the value closest to 0 within the limits shown in table 4.2. Then the mutate function is run several times on each individual to create the random population. The initial population is sorts according to nondomination, meaning that the algorithm gives a large number of solutions on, or in near proximity, to the Pareto front [5]. However, since only one objective is used in the following experiments, the algorithm simply sorted the population according to fitness.

The parent selection mechanism used is an ordinary binary tournament selection [11]. The offspring population is created by the use of regular mutation and/or crossover operators. The mutation and crossover operators are detailed in subsection 4.4.2 and 4.4.3 respectively. The survivor selection is done by grouping solutions into non-dominated sets, selecting the best sets to survive to the next generations. When using only one objective, the survivor selection will just select the N best solutions, N being the population size used, ensuring that the best solutions will always survive.

### 4.4.2 Mutation operator

The only mutation operator used was a creep mutation. Normally in creep mutation, each element has a probability for getting a small number added or subtracted. In this case, the probability for adding or subtracting was 1, thus always adding or subtracting a small number. The number added is decided by a normally distributed random number function. This function receives two parameters, the mean and the standard deviation of the normal distribution. To make sure that the probability of adding a number was the same as for subtracting a number, the mean of the distribution was set to 0. The standard deviation was set to 0.05 for all parameters.

### 4.4.3 Crossover operator

The crossover operator used is a variant of the uniform crossover. A pair of parents is chosen for the crossover. For each parameter in a pair of parents, the difference between the two parameters is calculated. Then the new parameters are created by adding or subtracting the difference multiplied with a small uniformly selected number between 0 and 1. Pseudo code for the crossover function is shown under.

```
float alpha0 = uniform(0, 1);
float alpha1 = uniform(0, 1);

for(all parameters){
        float diff = parent2.param - parent1.param;
        param1 += alpha0 * diff;
        param2 -= alpha1 * diff;
}
```

### 4.4.4 Fitness measurement

To be able to evolve the population towards better gaits, the genetic algorithm also needed a fitness function to rank the different solutions. The simulator tests each gait for 8 seconds, and therefore the simulator measures how far the Quadratot is able to go in a defined direction in these 8 seconds. The defined direction is showed in figure 4.6.

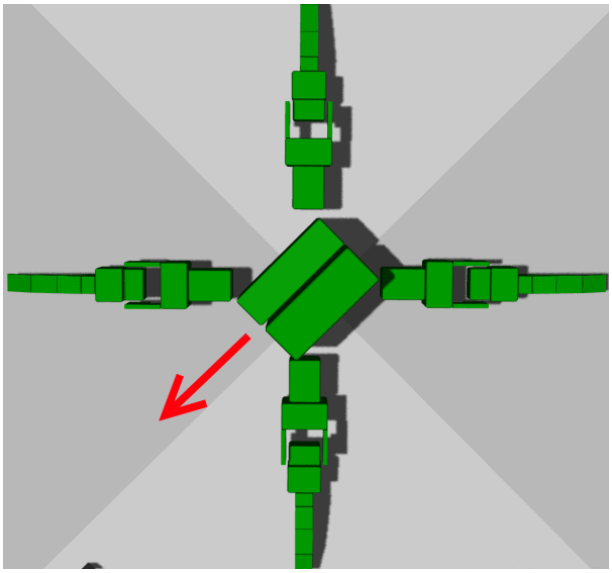


Figure 4.6: Picture showing the direction used when measuring fitness in the simulator.

# Chapter 5

# Experiments & Results

In this chapter the experiments conducted will be explained, together with the results of these experiments. First an initial experiment was conducted, to determine what kind of evolutionary operators should be used for the later experiments. After this initial experiment was finished, the simulator was used to evolve a large number of gaits, evolving both with and without obstacles, creating two different collections of gaits. Then a selection of the evolved gaits was tested on the physical Quadratot for comparison.

## 5.1 Experiment on evolutionary settings in the simulator

In the initial experiment, two main types of configurations where tested. In the standard configuration(config. 1), each parameter for every joint is mutated independently. This means that the amplitude, offset and phase for each joint is evolved independently of all the other parameters.

In the other configuration(config. 2), the amplitude and offset parameters were locked for the "knee" and "hip" joints. This means that all the "hip joints", of the legs has the same Amplitude and Offset, and the same apply to the "knee joints".

In addition to the two configurations mentioned over, both of these were tested with the crossover operator turned on(config. 3) and off(config. 4). This makes the total number of configurations tested 4. An overview of the 4 configurations is seen in Table 5.1.

| | **Description** |
|---|---|
| `Config.1` | Amplitude and Offset are free. Crossover enabled |
| `Config.2` | Amplitude and Offset are locked. Crossover enabled |
| `Config.3` | Amplitude and Offset are free. Crossover disabled |
| `Config.4` | Amplitude and Offset are locked. Crossover disabled |

Table 5.1: Table shows an overview of the 4 different configurations tested.

The reason for testing 4 different configurations was to determine which of the 4 that gave the best results. This was done because it was too time consuming to run each of the 4 configurations for all of the remaining experiments. The locked configuration has a smaller search space than the free configuration, thus were tested to see if this could be an advantage. The testing with crossover enabled and disabled was done to see if, and how much, the crossover could benefit the algorithm. The plan was to use the configuration with the best result when running the main simulation.

### 5.1.1 Results

The 4 configurations were all tested for 20 runs each. Each of these 20 runs consisted of 200 generation with a population of 256 individuals. The seed used for the obstacles was 1330 for all 4 tests.

The results from the 4 configurations are shown in 3 plots, Figure 5.1, Figure 5.2 and Figure 5.3. Figure 5.1 shows the mean result for each generation. Figure 5.2 shows the mean of all the best results from each run, for each generation. Figure 5.3 show the top 10 results from each run, for each of the 200 generations. The mean of the top 10 results are plotted to see if there was an outlier that got much better results than the rest of the gaits in the simulator. Plotting the mean of the top 10 would then be significantly lower than the max plot. Table 5.2 shows the values of the last generation shown in the plots 5.1, 5.2 and 5.3. All fitness values are shown as meters per 8 seconds.

| | Config 1 | Config 2 | Config 3 | Config 4 |
|---|---|---|---|---|
| `Mean(Fig 5.1)(m/8s)` | 1.3986 | **1.4043** | 1.1680 | 1.3015 |
| `Max(Fig.5.2)(m/8s)` | 1.5314 | **1.5767** | 1.3075 | 1.5126 |
| `Mean of top 10(Fig.5.3)(m/8s)` | 1.4897 | **1.5245** | 1.2670 | 1.4549 |

Table 5.2: Shows the fitness for the 200th generation for all 4 configurations tested. Config 1 and 2 are the free configurations with and without Crossover respectively. Config 3 and 4 are the locked configurations with and without Crossover respectively. These numbers are equal to the final points in the plots found in Figure 5.1, 5.2 and 5.3

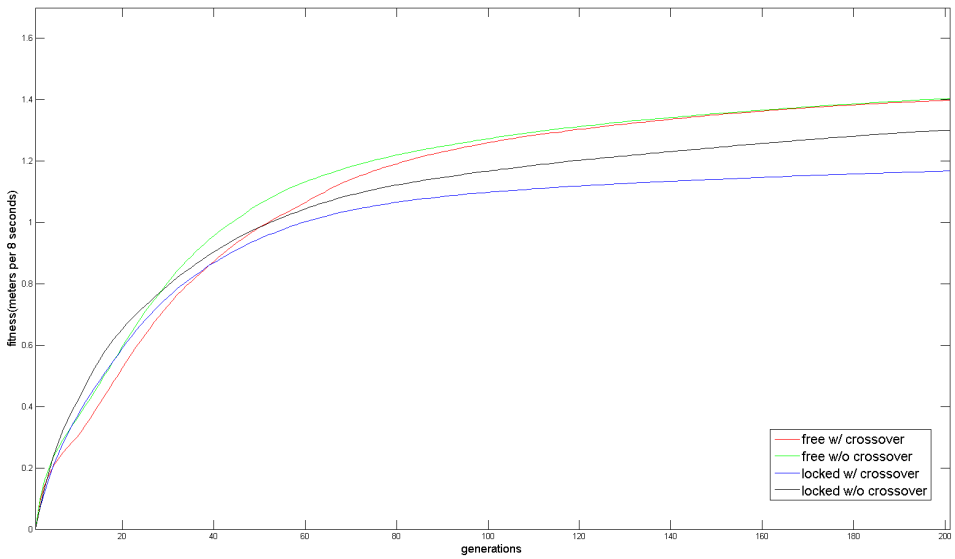



Figure 5.1: This plot was created by taking the mean of every result, for each generation, for all 20 runs. Each point on the "generations"-axis is created by calculating the mean of 5120 results.

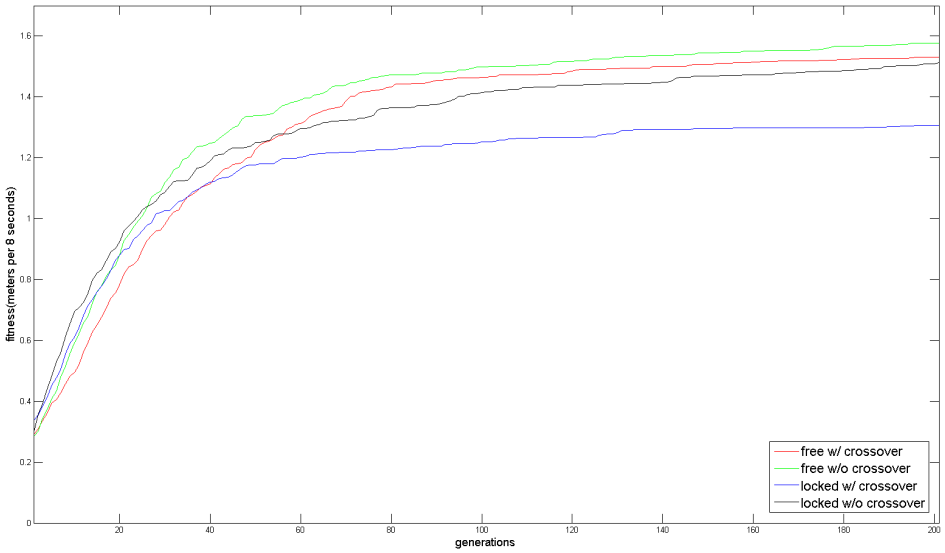



Figure 5.2: This plot was created by taking the max result, for each generation, for all 20 runs. Then the mean for each generation was calculated. Each point on the "generations"-axis is created by calculating the mean of 20 results.

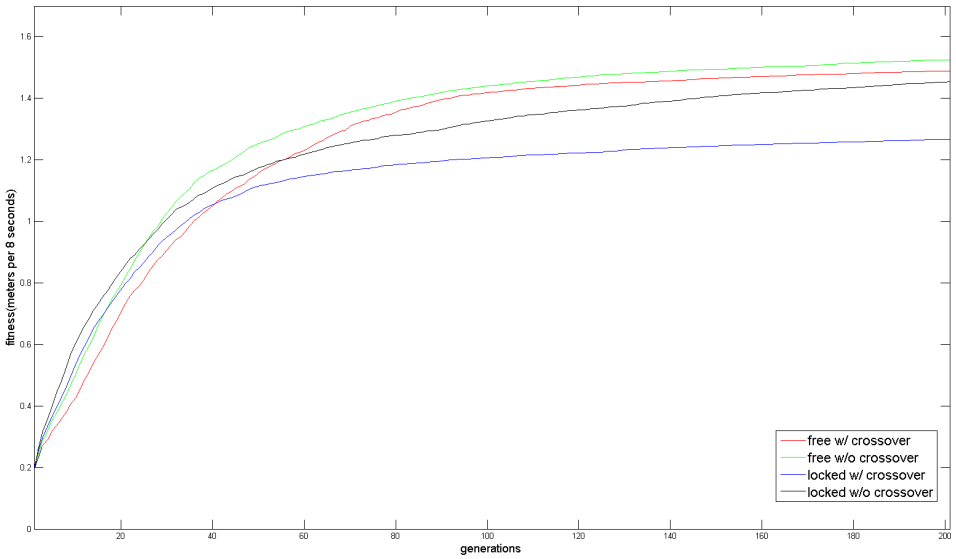



Figure 5.3: This plot was created by taking the 10 best results, for each generation, for all 20 runs, before the mean for each generation was calculated. Each point on the "generations"-axis is created by calculating the mean of 200 results.

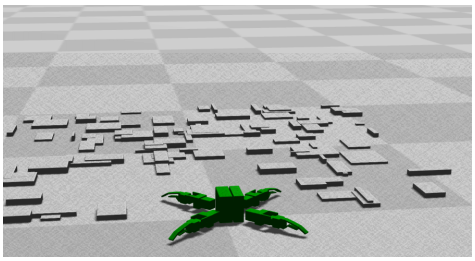
### 5.1.2 Discussion

In the plots shown in figure 5.2 and 5.3 it is quite easy to see that the green line, the free configuration without crossover, is better than the other configurations. This is also the case in the plot in figure 5.1, but the difference here is very small. The locked configuration without crossover is the best in all three plots until about 30 generations have passed, when the free configuration takes over as the best configuration. In both the free and the locked configuration, the configuration without the use of the crossover operator outperform its counterpart with crossover used. This suggests that the crossover used in this algorithm does not work very well together with the selection and mutation operator used in this experiment.

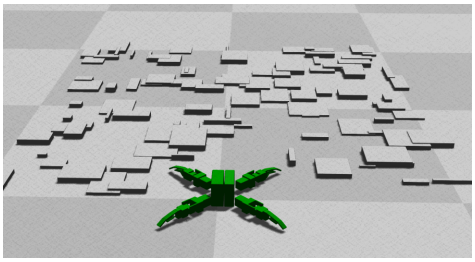
Another interesting observation is that the locked configurations both slightly outperforms the free configurations for the first 25-30 generations. This suggests that the smaller search-space is advantageous to early exploration of solutions. However, since the free configuration gives the best solutions after the 25th-30th generation mark, this configuration was selected to be used in the main runs.

## 5.2 Simulations

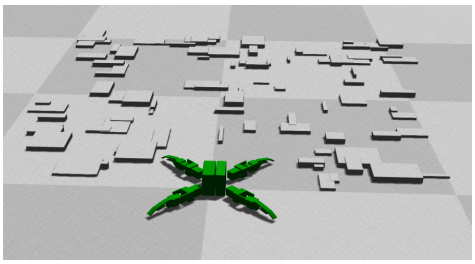
The first part of the main experiment was to evolve a large number of gaits that could be tested on the physical Quadratot. In total, 200 runs of the simulator were executed, each consisting of 200 generations with 256 individuals each, thus making the total number of gaits evolved 51,200. 100 of these runs were done using a flat environment for the robot. The other 100 runs were done with obstacles enabled. 5 different environments were used during the simulations with obstacles, making it 20 runs with each of the 5 environments. Figure 5.4a - 5.4e shows the different environments used in the simulations.



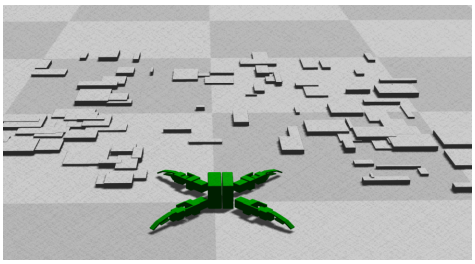
(a) Obstacle seed = 1



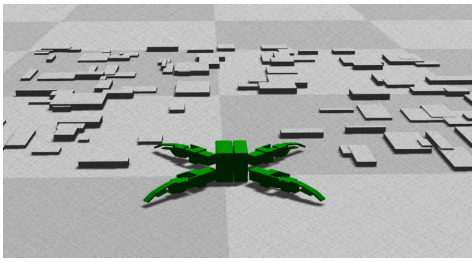
(b) Obstacle seed = 2319



(c) Obstacle seed = 3540



(d) Obstacle seed = 5678



(e) Obstacle seed = 7131102

Figure 5.4: The 5 different obstacle environments used in the simulation.

| Seed | 1 | 2319 | 3540 | 5678 | 7131102 |
|---|---|---|---|---|---|
| Mean(m/8s) | 1.346 | 1.317 | 1.486 | **1.533** | 1.417 |
| Max(m/8s) | 1.611 | 1.487 | 1.646 | **1.723** | 1.612 |
| Mean of Top 10(m/8s) | 1.520 | 1.417 | 1.603 | **1.659** | 1.552 |

Table 5.3: The table shows the mean, the mean of the max and the mean of the 10 best fitness values for simulator runs in the 5 different obstacle environments.

| | Obs. disabled | Obs. enabled | Difference |
|---|---|---|---|
| Mean(m/8s) | 1.87 | 1.42 | -24.1% |
| Max(m/8s) | 2.01 | 1.62 | -19.4% |
| Mean of Top 10(m/8s) | 1.97 | 1.55 | -21.3% |

Table 5.4: The table shows the mean, the mean of the max and the mean of the 10 best fitness values for simulator runs with obstacles enabled and disabled.

### 5.2.1 Results

Table 5.3 shows the mean, the mean of the max and the mean of the top 10 fitness values for the last generation, for each of the 5 different environments used in simulation. Because each of the environments were tested for 20 runs each, it is difficult to directly compare these sets of 20 runs with the set of 100 runs tested with the flat environment. In table 5.4 the results from the 100 runs tested with a flat environment is shown together with all 100 runs tested on the obstacle environments. Figure 5.5 shows a plot of the mean fitness values in all 100 simulator runs with obstacles enabled(green line) and all 100 simulator runs with obstacles disabled(red line). Figure 5.6 and 5.7 shows the max fitness values and top 10 fitness values for the same simulation runs respectively.

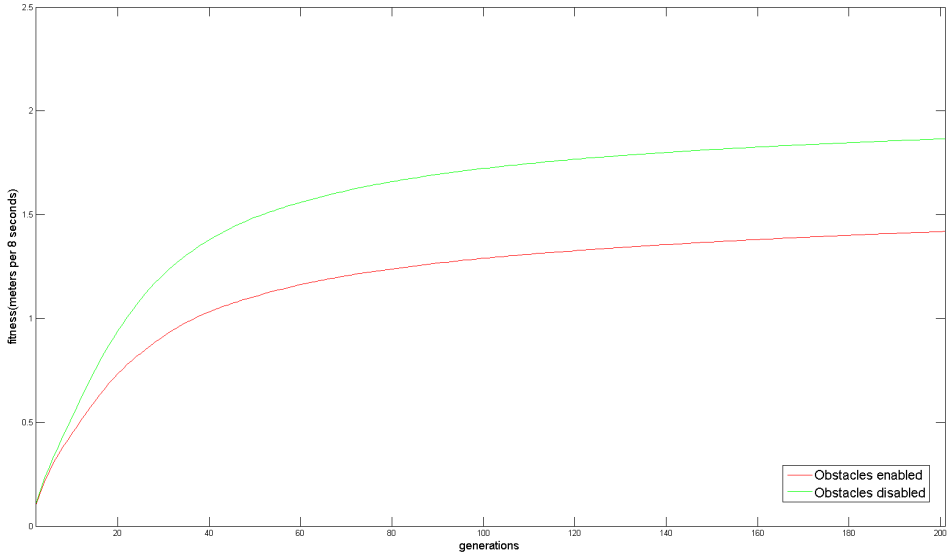



Figure 5.5: The figure shows the mean results of all 200 generations from simulations with and without the use of obstacles.

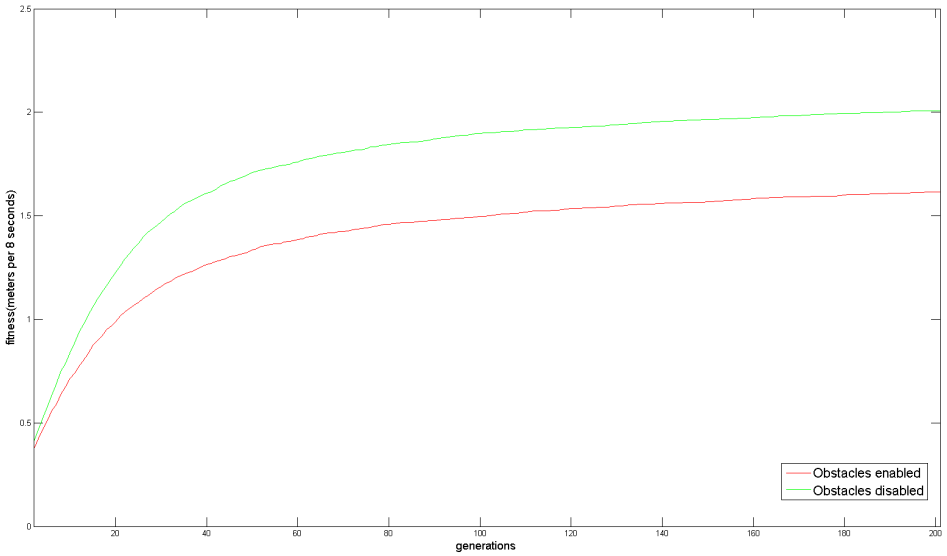



Figure 5.6: The figure shows the max results of all 200 generations from simulations with and without the use of obstacles.

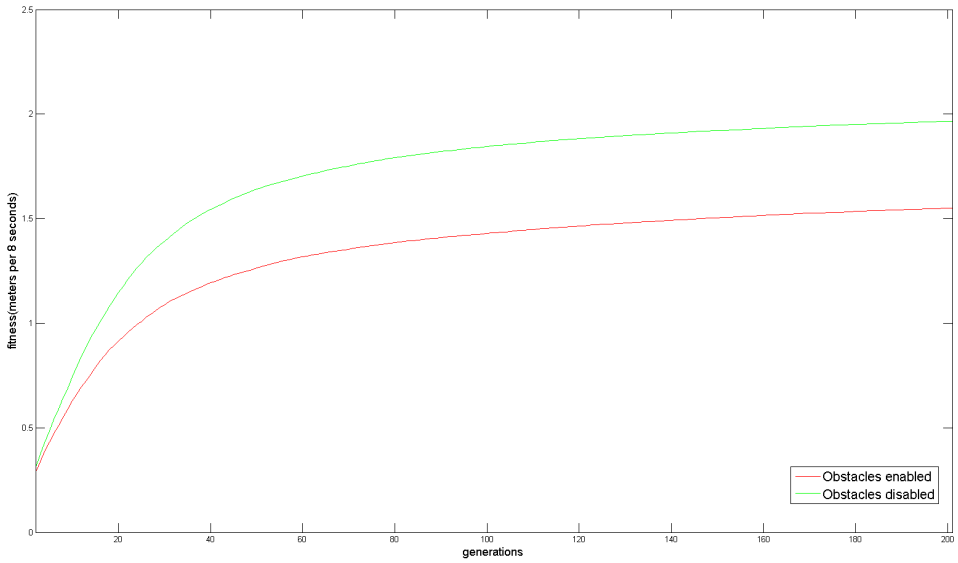



Figure 5.7: The figure shows the top 10 results of all 200 generations from simulations with and without the use of obstacles.

### 5.2.2 Discussion

Looking at the plots it is relatively easy to see that in the simulator, the test runs with obstacles disabled produces better results than the test runs with obstacles enabled. This result was expected, as the obstacles make the environment more challenging for the robot to traverse. As seen in table 5.4, the average gaits produced with the obstacles enabled are approximately 24% worse, than the average gaits produced with the obstacles disabled. The difference between the best gaits produced is slightly smaller, but it is still a significant difference. The best gaits produced with obstacles enabled are 19.5% worse than the best gaits with obstacles disabled. The plots in figures 5.5, 5.6 and 5.7, reveal that the gaits simulated in the flat environment evolves significantly faster to about 60 generations has passed. After this point, the gaits from both environments evolve at approximately the same speed.

In the results from the 5 different obstacle environments, there is also a significant difference between the different environments. The environment where the best gaits were produced is the environment created with the seed 5678. As seen in table 5.3 this environment has the best mean, max and top 10 results from all 5 environments. As seen in figure 5.4d, the boxes in this environment leave a little gap without any boxes through the rest of the boxes.

When running the gait in the bbp viewer, the robot has a tendency of using one of the rear legs to push. The front leg on the opposite side is pushed forwards, sliding over the floor. The two legs on the side are used for support. The robot is turned about 45 degrees before walking straight. In the environment with 5678 as the seed, the gap between the boxes lets the leg that is not lifted above ground the possibility to just slide through the gap without being stopped. This is a possible reason to why the gaits produced in this environment are faster than the gaits produced in the 4 environments.

## 5.3 Testing in simulator with obstacles enabled and disabled

In addition to the testing on the physical robot, the selected gaits evolved with the use of obstacles were tested in the simulator in a flat environment. The same was done on the gaits evolved in the flat environment, except it was done the other way around. 10 gaits from each of the two types of environments were selected. From the 5 environments with obstacles, the 2 best gaits were selected from each of the 5 environments. From the gaits evolved with obstacles disabled, the 10 gaits selected were the best gaits from 10 different runs out of the total number of 100 runs. For testing the gaits evolved in the flat environment on an obstacle environment, the environment with seed 5678 was used. This environment was selected because it was the environment which produced the best results.

### 5.3.1 Results

Table 5.5 shows the results from testing of the gaits evolved in a flat environment, when they were tested in an environment with obstacles. Table 5.6 shows the results from testing the gaits evolved in the obstacle environments in a flat environment.

| Gait nr.(m/8s) | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **Average** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Orig. fit.(m/8s) | **2.18** | 2.14 | 2.08 | 2.08 | 2.031 | 2.125 | 2.119 | 2.100 | 2.089 | 1.973 | 2.092 |
| Fitness obs.(m/8s) | 1.24 | 0.31 | 1.09 | 1.07 | 1.39 | 0.92 | 0.69 | **1.76** | 1.31 | 1.16 | 1.09 |
| Difference(%) | -43.22 | -85.73 | -47.72 | -48.46 | -31.66 | -56.8 | -67.53 | **-16.29** | -37.1 | -41.21 | -47.75 |

Table 5.5: Results from testing the gaits evolved in a flat environment in an obstacle environment.

| Gait nr. | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **Average** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Orig. fit.(m/8s) | **1.87** | 1.84 | 1.79 | 1.76 | 1.73 | 1.72 | 1.75 | 1.76 | 1.72 | 1.64 | 1.76 |
| Seed | 5678 | 5678 | 3540 | 7131102 | 1 | 2319 | 3540 | 7131102 | 1 | 2319 | — |
| Fitness flat(m/8s) | **1.91** | 1.86 | 1.84 | 1.83 | 1.69 | 1.79 | 1.77 | 0.98 | 1.84 | 1.82 | 1.73 |
| Difference(%) | +2.24 | +1.09 | +3.19 | +4.04 | -1.97 | +4.30 | +0.97 | -44.17 | +7.41 | **+11.03** | -1.253 |

Table 5.6: Results from testing the gaits evolved in an obstacle environment in a flat environment.

### 5.3.2 Discussion

Looking at table 5.5, all 10 gaits had significantly lower performance when run in an environment with obstacles, than they originally did when run in the flat environment. This result was expected, as the environment is more complex to traverse than the flat environment, and the legs had a tendency to get stuck because of the obstacles. On average, the fitness in this is almost halved compared to the fitness evolved in the flat environment. All 10 gaits tested performed worse in the environment with obstacles.

The test with the gaits originally evolved with obstacles, tested in a flat environment, produced results that were a lot better. 8 out of the 10 gaits tested did improve when tested in the flat environment compared to the environment with obstacles. Of the 2 gaits that had worse performance in the flat environment, only one of them was significantly worse, as shown in table 5.6. On average, the gaits tested in the flat environment was only slightly worse than the original gaits in the obstacle environment, the difference being that, on average, the gaits are only approximately 1.25% worse.

## 5.4 Testing on physical Robot

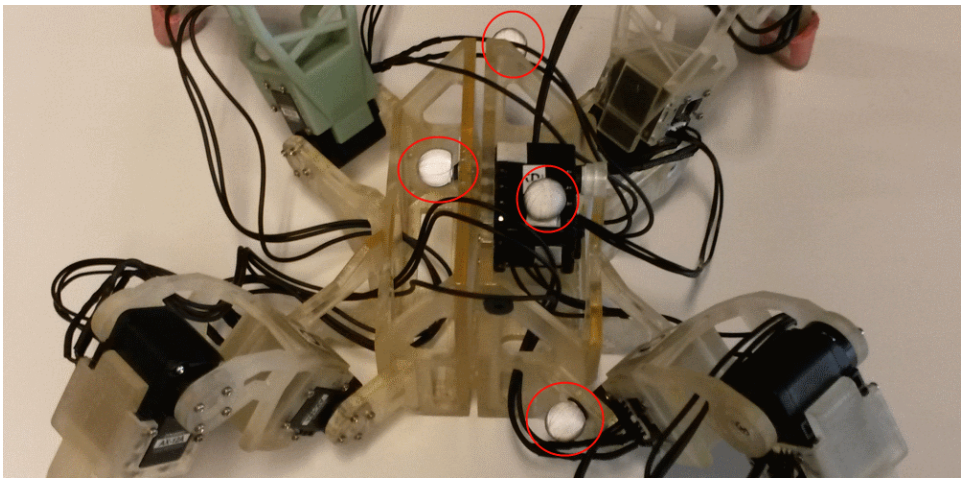


Figure 5.8: Picture showing the 4 reflective markers on the Quadratot, used for the motion capturing.

(a) The Quadratot in its starting position.

(b) The Quadratot in action while measuring fitness.

Figure 5.9: The area used for measuring the robot with the use of motion capture.

When the evolving of the gaits was finished, it was time to test the evolved gaits on the physical Quadratot robot. The measuring of distance on the physical Quadratot was done with the use of a motion capture studio. The motion capture studio used, has 8 cameras to track the motion of the robot when testing. The Quadratot was fitted with 4 reflective markers that the cameras could track when the Quadratot was moving. Figure 5.8 show the 4 reflective markers used for the tracking of the Quadratot. The program used for controlling the Quadratot was set up to run the selected gait for 8 seconds before stopping. This was to make it easier to compare the results from the physical measurements with the results from the simulator runs. The Quadratot was manually reset to its starting position after each 8-second run. All 20 gaits tested were tested 10 times each, making the total number of tests performed 200. All tests performed using the physical robot was done in the environment shown in figure 5.9a. The Quadratot in action while measuring fitness with motion capture is shown in figure 5.9b.

A Mann-Whitney U test was done on the two populations found in the physical experiment [30]. This test was done to compare the gaits evolved with obstacles enabled, to the gaits evolved with obstacles disabled. This test is used to decide if there is a statistically significant difference between the two populations. The test calculates a p-value that can be used to determine if there is a statistically significant difference between the two populations. The test was done on an online Mann-Whitney U-test calculator[1].

### 5.4.1 Results

Table 5.7 shows the results from the physical testing of the gaits evolved in the flat environment. The results from the gaits evolved with obstacles enabled are shown in table 5.8. The average of the results shown in table 5.7 and 5.8 are shown in table 5.9.

| **Gait nr.** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|---|---|---|---|---|---|---|---|---|---|---|
| `Orig.fit.(m/8s)` | **2.18** | 2.14 | 2.08 | 2.08 | 2.03 | 2.13 | 2.12 | 2.10 | 2.09 | 1.97 |
| `Sim.w/obs.(m/8s)` | 1.24 | 0.31 | 1.09 | 1.07 | 1.39 | 0.92 | 0.69 | **1.76** | 1.31 | 1.16 |
| `Average(m/8s)` | 0.26 | 0.43 | 0.53 | 0.52 | 0.69 | 0.87 | 0.37 | **0.89** | 0.68 | 0.27 |
| `Median(m/8s)` | 0.27 | 0.42 | 0.53 | 0.49 | 0.70 | **0.90** | 0.38 | 0.89 | 0.66 | 0.27 |
| `Max(m/8s)` | 0.35 | 0.55 | 0.55 | 0.71 | 0.93 | **1.03** | 0.58 | 0.94 | 0.98 | 0.37 |
| `Min(m/8s)` | 0.17 | 0.31 | 0.51 | 0.43 | 0.37 | 0.66 | 0.14 | **0.88** | 0.45 | 0.18 |
| `Std.dev.` | 0.06 | 0.07 | **0.01** | 0.10 | 0.19 | 0.11 | 0.12 | 0.02 | 0.19 | 0.07 |

Table 5.7: Results from physical tests on the Quadratot using gaits evolved with obstacles disabled.

| **Gait nr.** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|---|---|---|---|---|---|---|---|---|---|---|
| `Orig.fit(m/8s)` | **1.87** | 1.84 | 1.79 | 1.76 | 1.73 | 1.72 | 1.75 | 1.76 | 1.72 | 1.64 |
| `Seed` | 5678 | 5678 | 3540 | 7131102 | 1 | 2319 | 3540 | 7131102 | 1 | 2319 |
| `Sim.flat(m/8s)` | **1.91** | 1.86 | 1.84 | 1.83 | 1.69 | 1.79 | 1.77 | 0.98 | 1.84 | 1.82 |
| `Average(m/8s)` | 0.85 | **1.03** | 1.00 | 0.69 | 0.89 | 0.42 | 0.90 | 0.24 | 0.59 | 0.25 |
| `Median(m/8s)` | **1.05** | 1.03 | 1.00 | 0.67 | 0.89 | 0.42 | 0.91 | 0.25 | 0.61 | 0.24 |
| `Max(m/8s)` | **1.13** | 1.07 | 1.03 | 0.80 | 0.96 | 0.47 | 0.96 | 0.34 | 0.64 | 0.41 |
| `Min(m/8s)` | 0.25 | **0.99** | 0.95 | 0.62 | 0.84 | 0.39 | 0.81 | 0.15 | 0.43 | 0.14 |
| `Std.dev.` | 0.33 | 0.03 | **0.02** | 0.06 | 0.04 | 0.03 | 0.03 | 0.07 | 0.03 | 0.09 |

Table 5.8: Results from physical tests on the Quadratot using gaits evolved with obstacles enabled.

[1]Test used: *http://www.socscistatistics.com/tests/mannwhitney/Default2.aspx*

| | Obs. disabled | Obs. enabled | Diff.(%) |
|---|---|---|---|
| Avg. sim fitness(m/8s) | 2.09 | 1.76 | -15.8 |
| Average(m/8s) | 0.55 | 0.69 | +25.5 |
| Average Median(m/8s) | 0.55 | 0.71 | +29.1 |
| Median | 0.52 | 0.68 | +30.8 |
| Average Max.(m/8s) | 0.70 | 0.78 | +11.4 |
| Average Min.(m/8s) | 0.41 | 0.56 | +36.6 |
| Average Std.dev. | 0.09 | 0.07 | — |
| Avg. Reality gap(m/8s) | -1.54 | -1.07 | — |
| Avg. reality Gap(%) | -73.7 | -60.1 | — |

Table 5.9: Table showing the results from the gaits tested on the physical robot, both from the flat environment and the environments with obstacles. The left middle column show the average of the results found in table 5.7 and the right middle column show the average of the results found in table 5.8

### 5.4.2 Discussion

In table 5.7, only 2 out of the 10 gaits tested come close to the 1 meter per 8 second mark, namely gait 6 and 8. Of these two, gait number 8 is the most robust gait from table 5.7, having a standard deviation of only 0.02. Only gait 3 had a lower standard deviation in table 5.7, but this gait did perform significantly worse than gait 8.

In table 5.8, 5 out of the 10 gaits tested are around the 1 meter per 8 seconds mark, namely gait number 1, 2, 3, 5 and 7. Gait number 1 has the highest median and maximum results, but 2 of the 10 test runs did only get a result of under 0.3m over 8 seconds, and this significantly lowered the average of the 10 runs. Gait number 2 has the highest average, and second highest median and maximum results from table 5.8. The standard deviation is lower than for gait 1, making it a significantly more robust gait. Gait 2 is also better than all gaits found in table 5.7.

Looking at table 5.9, the results from the gaits evolved with obstacles enabled are better than the results from the gaits evolved in the flat environment. Even though the gaits evolved with obstacles enabled are significantly worse in the simulator, both the average and median results from the physical tests are over 25% better than the gaits evolved with obstacles disabled. While the average minimum results are over 35% better with gaits evolved with obstacles enabled, the max results are only 11% better. This suggests that the best gaits evolved with the obstacles enabled are not necessarily much better, but it has a better average. It also suggests that a greater part of the solutions will perform well on the real robot. This means the gaits evolved in the environments with obstacles are more robust than the gaits evolved in the flat environment.

The p-value found in the Mann-Whitney U test for the two populations were 0.001. A p-value of 0.001 means that the difference is statistically significant.

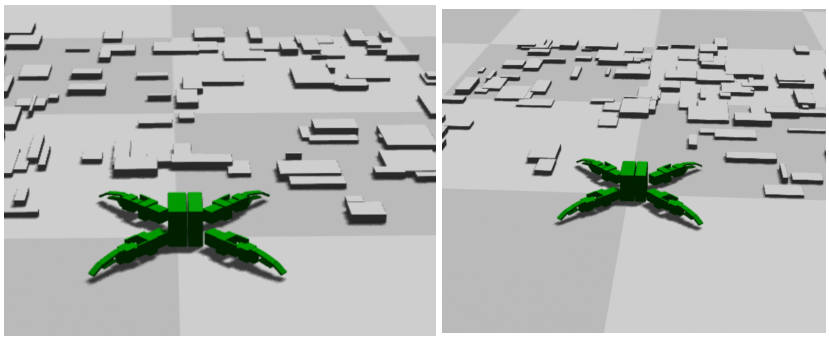
Even though the data collected from physical tests on the robot is limited, the difference in the reality gap is quite significant. The gaits evolved in the environments with obstacles enabled are, on average, approximately 60% worse than the gaits in the simulator, while the gaits evolved in the flat environment are almost 75% worse. The average standard deviation is also slightly smaller for the gaits evolved with obstacles enabled, once again suggesting that the gaits are more robust than the gaits from the flat environment. This shows that the use of obstacles, to create a kind of noise in the environment, may be useful for evolving a more effective gait for a walking robot. The small amount of data may, of course, hide the truth. However, this experiment suggests that it, at least, has the possibility for evolving better gaits.

## 5.5 Updated simulator and physical experiments

When inserting the numbers used to limit the amplitude, offset and phase in the simulator into table 4.2, an error was discovered. The limits used on the phase parameter for the outer joints were set to go from 180 degrees to 180 degrees, limiting the phase of the 4 outer joints to one set value for all runs. The correct limit should have been from -180 degrees to 180 degrees. This error does not mean that the results found in section 5.4 are wrong, but they are the results of evolution with an extra limit that would decrease the possibilities of the gaits being as good as they could be.

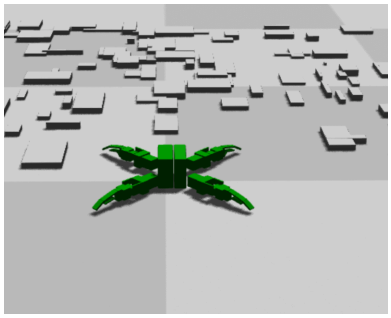
To repeat all experiments using the correct limit would have been too time-consuming, thus only small parts of the experiments have been repeated using the correct limits. Gaits were evolved using a population of 128, evolving over 100 generations. The number of obstacle environments have been limited to three different obstacle environments. 10 runs were done on each of these environments, together with 30 runs on the flat environment, making it a total of 60 runs. For the physical testing, 20 different gaits were tested, 10 from the flat environment, and 10 from the 3 obstacle environments combined. Each gait has been tested 5 times, using the motion capture equipment to measure the fitness.

The Mann-Whitney U-test was performed on these results as well.



(a) Obstacle seed = 2323

(b) Obstacle seed = 7891



(c) Obstacle seed = 5315

Figure 5.10: The 3 different obstacle environments used in the revised simulations.

### 5.5.1 Results

Figures 5.11, 5.12 and 5.13 shows plots of the mean, the mean of the best, and the mean of the 10 best results from the revised simulator runs. Table 5.10 shows the results from the last generation from figures 5.11, 5.12 and 5.13. Tables 5.11 and 5.12 shows the results from the revised motion capture runs with and without obstacles enabled, respectively. Table 5.13 show the average results from table 5.11 and table 5.12.

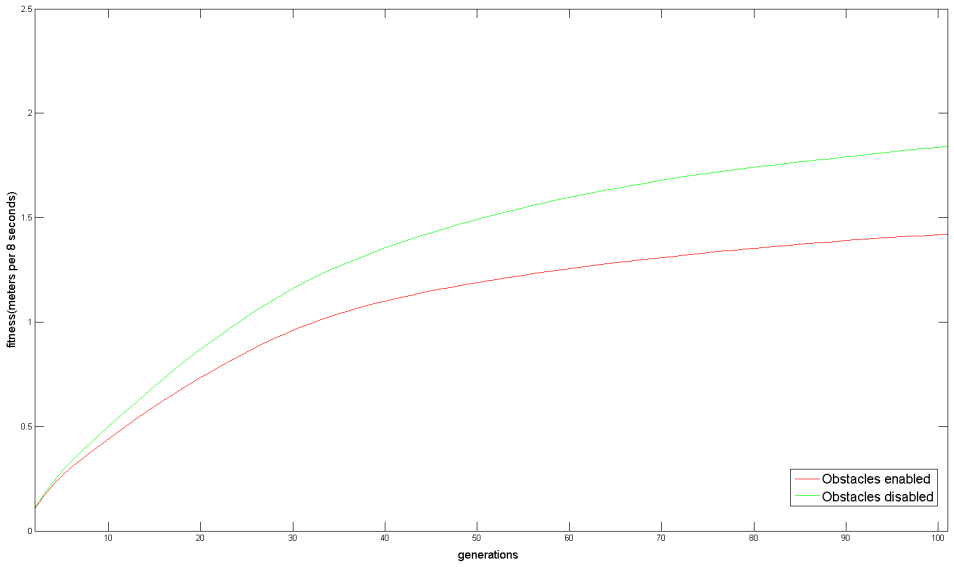



Figure 5.11: Plot showing the mean of the results from the revised simulator runs.

| | Obs.disabled | Obs.enabled | Difference |
|---|---|---|---|
| Mean(m/8s) | 1.84 | 1.41 | -23.4% |
| Max.(m/8s) | 2.04 | 1.63 | -20.1% |
| Top 10(m/8s) | 1.97 | 1.54 | -21.8% |

Table 5.10: Table showing the mean, mean of the max., and mean of the 10 best results for the revised simulator runs.

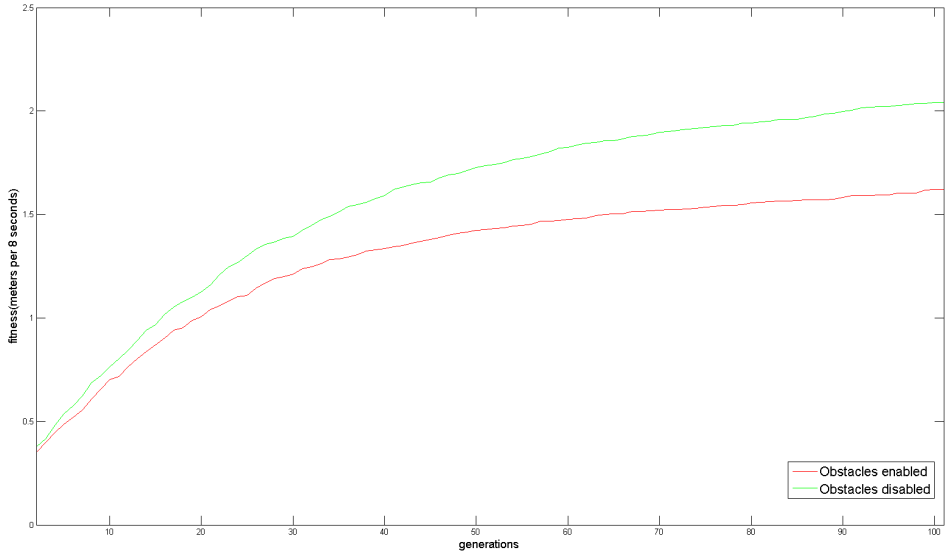



Figure 5.12: Plot showing the mean of the best results from the revised simulator runs.

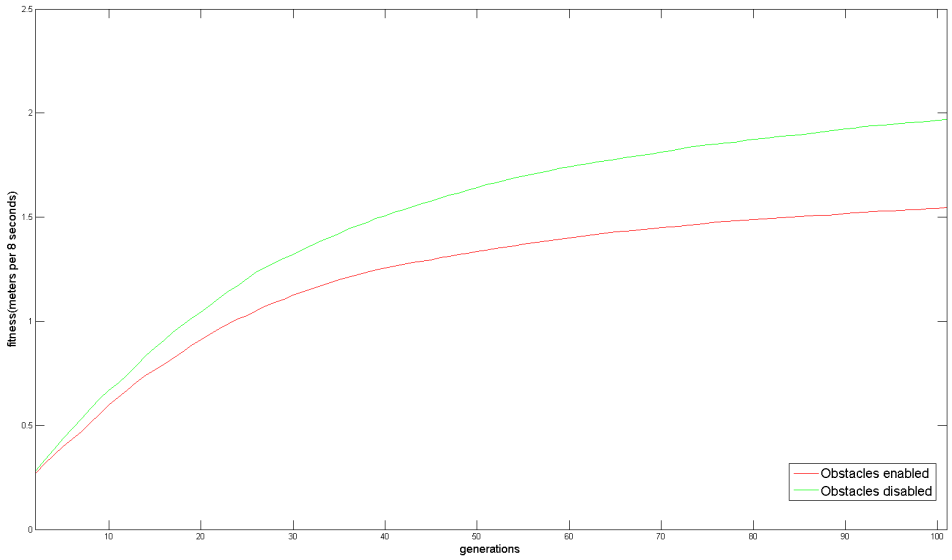



Figure 5.13: Plot showing the mean of the top 10 results from the revised simulator runs.

| **Gait nr.** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|---|---|---|---|---|---|---|---|---|---|---|
| `Orig.fit(m/8s)` | **2.34** | 2.23 | 2.17 | 2.16 | 2.15 | 2.10 | 2.26 | 2.18 | 2.14 | 2.10 |
| `Average(m/8s)` | 0.90 | 0.72 | 0.79 | 0.77 | 0.88 | 0.59 | 1.04 | **1.47** | 0.83 | -0.04 |
| `Median(m/8s)` | 0.83 | 0.71 | 0.81 | 0.78 | 0.90 | 0.61 | 1.06 | **1.46** | 0.83 | -0.04 |
| `Max(m/8s)` | 1.18 | 0.78 | 0.81 | 0.79 | 0.95 | 0.63 | 1.10 | **1.54** | 0.86 | -0.03 |
| `Min(m/8s)` | 0.79 | 0.69 | 0.75 | 0.75 | 0.74 | 0.49 | 0.99 | **1.39** | 0.80 | -0.06 |
| `Std.dev.` | 0.16 | 0.03 | 0.02 | 0.02 | 0.08 | 0.06 | 0.05 | 0.06 | 0.02 | **0.01** |

Table 5.11: Results from revised physical tests on the Quadratot using gaits evolved in a flat environment.

| Gait nr. | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Orig.fit(m/8s) | **1.90** | 1.77 | 1.68 | 1.67 | 1.76 | 1.70 | 1.74 | 1.64 | 1.65 | 1.71 |
| Seed | 2323 | 2323 | 7891 | 7891 | 5315 | 5315 | 2323 | 7891 | 5315 | 2323 |
| Average(m/8s) | 0.36 | 0.38 | 0.74 | 0.54 | 0.92 | **0.97** | 0.88 | 0.76 | 0.82 | 0.76 |
| Median(m/8s) | 0.35 | 0.36 | 0.74 | 0.52 | 0.93 | **0.96** | 0.88 | 0.75 | 0.81 | 0.76 |
| Max(m/8s) | 0.39 | 0.47 | 0.82 | 0.60 | 0.97 | **0.99** | 0.93 | 0.83 | 0.83 | 0.79 |
| Min(m/8s) | 0.32 | 0.31 | 0.67 | 0.49 | 0.86 | **0.95** | 0.85 | 0.74 | 0.80 | 0.73 |
| Std.dev. | 0.03 | 0.07 | 0.07 | 0.05 | 0.04 | **0.01** | 0.03 | 0.04 | **0.01** | 0.02 |

Table 5.12: Results from revised physical tests on the Quadratot using gaits evolved with obstacles enabled.

| | Obs. disabled | Obs. enabled | Diff. |
|---|---|---|---|
| Avg. sim fitness(m/8s) | 2.18 | 1.71 | -21.6% |
| Average(m/8s) | 0.80 | 0.71 | -11.3% |
| Median(m/8s) | 0.79 | 0.71 | -10.1% |
| Max(m/8s) | 0.86 | 0.76 | -11.6% |
| Min(m/8s) | 0.73 | 0.67 | -8.2% |
| Std.dev. | 0.05 | 0.04 | — |
| Avg. Reality gap(m/8s) | -1.38 | -1.00 | — |
| Avg. reality Gap(%) | -63.3 | -58.5 | — |

Table 5.13: Table showing the average results from the gaits tested on the physical robot. The left middle column show the average of the results found in table 5.11 and the right middle column show the average of the results found in table 5.12

### 5.5.2 Discussion

The results from the new simulator runs are almost identical to the results found in the old simulator runs(Section 5.2). The gaits evolved in the flat environment are, naturally, still significantly better than the results from the environment with obstacles, due to the less complex environment. However, these results were evolved in half the number of generations, and with half the population size, compared to the old simulator runs. This makes it difficult to compare them directly to the old results. Still, this suggests that they could be improved even more, if allowed to run for the full 200 generations. Figure 5.14 shows the mean simulator results from the old and the new experiment.

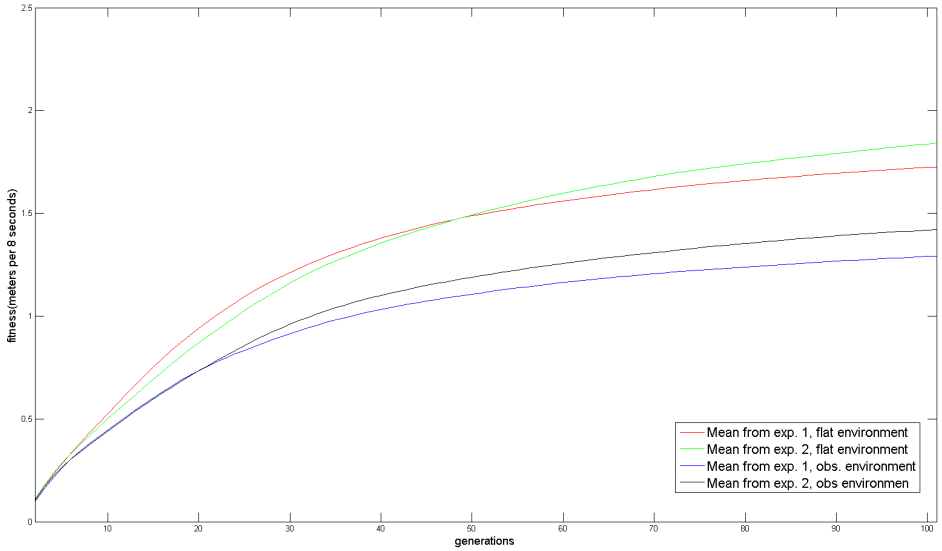



Figure 5.14: Plot showing the mean of both the new and the old simulator runs.

From the results shown in table 5.13, the Reality Gap is smaller in the gaits evolved in the environment with obstacles. However the difference in the Reality Gap is significantly smaller than in the previous experiments.

The gaits evolved in the environment with obstacles enabled are 58% worse on the physical robot than in the simulator. For the gaits evolved in the flat environment, the gaits on the physical robot were about 63% worse, resulting in a Reality Gap slightly bigger than for the gaits evolved in the environment with obstacles.

An interesting observation is found when comparing table 5.9 to table 5.13. The average and median results from the environment with obstacles from the new experiments are almost identical to the results from the old experiment. However, the average and median results from the flat environment are significantly better in the new experiments compared to the old experiments. This suggests that the gaits evolved in the environment with

obstacles may be less affected by the more limited movement caused by the phase parameter limit.

Another observation is that the results on the physical robot vary less than they did in the old experiments. This was an expected result, as the legs are no longer limited to move the 'knee-joints' at the same time. In the previous experiment, the limited leg movement may have forced the GA to evolve unstable gaits. Now, the legs could work together to make the Quadratot move forward, instead of working against each other. However, this did not apply to gait number 10 from table 5.11. This gait did not move forwards at all, but did move slightly sideways because the front and rear leg worked against each other, pushing it back and forth, without really going anywhere. One of the runs of gait 10 is shown in figure 5.15.

When observing the Quadratot when testing the 20 different gaits, most of the gaits were very similar to the gaits observed in section 5.4. However, one of the gaits did separate itself by using both of the rear legs to push, and both of the front legs to pull the Quadratot forward. This gait was much more similar to how a normal, four legged, animal walks.

The Mann-Whitney U-test gave a p-value of 0.13888. This p-value determines that the difference between the two populations is not statistically significant.

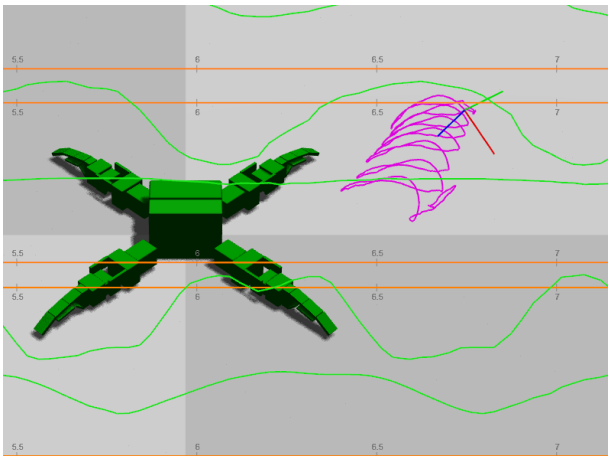


Figure 5.15: Picture showing gait nr. 10 from the flat environment. The purple line shows the movement measured by the motion capture system on one of the 10 tests.

# Chapter 6

# Discussion

This chapter begins with an overall discussion of the results from chapter 5, then the conclusion follows, before suggestions for future work is given.

## 6.1 Overall Discussion

The results from the physical experiments from section 5.4, show that the gaits evolved in the environment with obstacles have a significantly smaller Reality Gap than the gaits evolved in the flat environment. Even though the results from 5.4 do have a significantly larger difference in Reality Gap than the results from 5.5s, the difference is there, also in the results from 5.5.

However, since the second collection of simulator runs from 5.5 had a smaller amount of runs, a smaller population and a smaller number of generations, it is difficult to compare these directly. The results from table 5.4 and 5.10 are very similar, both for gaits evolved with obstacles enabled and gait evolved with obstacles disabled, suggesting that, if allowed to run for 200 generations, the simulator results from 5.5 would be better than the results from table 5.4.

In total, the Reality Gap is very large, both for gaits evolved using obstacles, and gaits not using obstacles, meaning that the simulator could be tuned significantly better to behave more realistically. The physical robot did slide a lot more than the simulator model, suggesting that friction is one parameter that could need more tuning. The power of the servos was also difficult to model, and could also need more tuning.

## 6.2 Conclusion

This thesis suggested including noise in a simulator, in the form of obstacles, when evolving in the simulator, to reduce the Reality Gap between the robot in the simulator and the real robot. The obstacles were introduced as several boxes in the simulator, with size and placement determined by random generated numbers. Each run using obstacles had a set seed, to secure the ability to reproduce the environment in the simulator at a later point in time.

A significant amount of runs were done in the simulator, both with obstacles included and with a flat environment. Then a selection of gaits was selected to be tested on the real, physical Quadratot robot. This process was partly repeated due to a small error in the settings of the Genetic Algorithm used. This error did not make the old results invalid, but it was done to see if there could be a significant improvement with this error fixed.

The first round of results shows a statistically significant reduction in Reality Gap for the gaits evolved in the obstacle environment, compared to the Reality Gap found in the gaits evolved in the flat environment. The gaits evolved in the obstacle environment did get an average reduction of fitness of about 60% on the physical robot compared to the simulator fitness. For the gaits evolved in the flat environment the average reduction was about 74%, which is a significantly higher Reality Gap.

The second round of results also shows a reduction in Reality Gap for the gaits evolved in the obstacle environment compared to the gaits evolved in the flat environment. However, the difference is smaller than in the first round of results. According to the Mann-Whitney U-test, the difference is too small to be considered statistically significant.

The results from both rounds of physical experiments do support the hypothesis that using a more complex environment for simulation, may improve the transfer of a simulated solution to reality. The amount of data collected from the physical tests is still quite small, making it difficult to conclude with anything specific. However, the results do show that the possibility of reducing the Reality Gap, by including some kind of noise in the simulator, is there.

## 6.3 Future Work

In the results found, the reality gap was quite large. If the reality gap can be reduced, the robot can obtain a significantly higher fitness. The most obvious way of reducing the reality gap in this case is to improve the parameters of the simulator. The friction numbers used in the simulator proved to be a weak point in the modelling of the robot and the environment. The Quadratot did slide significantly more in the physical tests than in the simulations. With improved friction calculations, the fitness of the gaits evolved should improve significantly.

Another weak area of the simulator model is the modelling of the servos in the joints. By calculating the power of the servos more accurately, the model should become significantly more realistic.

The obstacles used in the previous experiments were the same for several runs. When all gaits in one run are evolved in the same environment, the gaits do have a tendency of over fitting to the environment, creating a gait that especially suits that environment. There are several methods to prevent this from happening. One method could be to use completely random obstacles for each evaluation. One could also evaluate each gait in several different environments, the average score from all environments creating the fitness of the gait.

As seen in the first physical experiment, one of the environments, namely environment 5678, did produce significantly better results than the other 4 environments. Using this knowledge, one can develop environments producing more robust gaits, and use these in evolution.

In the previous physical experiments, gaits were only tested on a flat environment. Physical tests could also be done on environments with obstacles.

# Bibliography


[1] Homepage of ParadisEO. *http://paradiseo.gforge.inria.fr/index.php?n=Main.MO*.

[2] 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*, pages 627–634. ACM, 2010.

[3] Guy Baele, Nicolas Bredeche, Evert Haasdijk, Steven Maere, Nico Michiels, Yves Van de Peer, Thomas Schmickl, Christopher Schwarzer, and Ronald Thenius. Open-ended on-board evolutionary robotics for robot swarms. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 1123–1130. IEEE, 2009.

[4] Jeff Clune, Benjamin E Beckmann, Charles Ofria, and Robert T Pennock. Evolving coordinated quadruped gaits with the hyperneat generative encoding. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 2764–2771. IEEE, 2009.

[5] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and Tanaka Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. *Lecture notes in computer science*, 1917:849–858, 2000.

[6] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, 2002.

[7] University of Oslo Department for Informatics. About the simulator framework. *http://robin.wiki.ifi.uio.no/EvoRobSim*.

[8] S. Doncieux, J.-B. Mouret, N. Bredeche, and V. Padois. Evolutionary robotics: Exploring new horizons. In S. Doncieux, N. Bredeche, and J.-B. Mouret, editors, *Studies in Computational Intelligence, New Horizons in Evolutionary Robotics*, volume 341, pages 3–25. Springer, 2011.

[9] Agoston E Eiben and James E Smith. *Introduction to evolutionary computing*. springer, 2003.

[10] Kyrre Glette, Gordon Klaus, Juan Cristobal Zagal, and Jim Torresen. Evolution of locomotion in a simulated quadruped robot and transferral to reality. In *Proceedings of the 17th International Symposium on Artificial Life and Robotics*, pages 1139–1142. ALife Robotics, 2012.

[11] David E Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. *Urbana*, 51:61801–2996, 1991.

[12] Roderich Groß, Michael Bonani, Francesco Mondada, and Marco Dorigo. Autonomous self-assembly in swarm-bots. *Robotics, IEEE Transactions on*, 22(6):1115–1130, 2006.

[13] Gregory S Hornby, Seichi Takamura, Takashi Yamamoto, and Masahiro Fujita. Autonomous evolution of dynamic gaits with two quadruped robots. *Robotics, IEEE Transactions on*, 21(3):402–410, 2005.

[14] Gregory S Hornby, Seiichi Takamura, Jun Yokono, Osamu Hanagata, Takashi Yamamoto, and Masahiro Fujita. Evolving robust gaits with aibo. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 3, pages 3040–3045. IEEE, 2000.

[15] Nick Jakobi. Running across the reality gap: Octopod locomotion evolved in a minimal simulation. In *Evolutionary Robotics*, pages 39–58. Springer, 1998.

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

[17] Jérôme Kodjabachian and J-A Meyer. Evolution and development of neural controllers for locomotion, gradient-following, and obstacle-avoidance in artificial insects. *Neural Networks, IEEE Transactions on*, 9(5):796–812, 1998.

[18] Sylvain Koos, Jean-Baptiste Mouret, and Stéphane Doncieux. Crossing the reality gap in evolutionary robotics by promoting transferable controllers. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 119–126. ACM, 2010.

[19] Vladik Kreinovich, Chris Quintana, and Olac Fuentes. Genetic algorithms: what fitness scaling is optimal? *Cybernetics and Systems*, 24(1):9–26, 1993.

[20] T-T Lee, C-M Liao, and Ting-Kou Chen. On the stability properties of hexapod tripod gait. *Robotics and Automation, IEEE Journal of*, 4(4):427–434, 1988.

[21] Hod Lipson, Josh C Bongard, Viktor Zykov, and Evan Malone. Evolutionary robotics for legged machines: From simulation to physical reality. In *IAS*, pages 11–18, 2006.

[22] Hod Lipson and Jordan B Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406(6799):974–978, 2000.

[23] Sara Lohmann, Jason Yosinski, Eric Gold, Jeff Clune, Jeremy Blum, and Hod Lipson. Aracna: An open-source quadruped platform for evolutionary robotics. In *Artificial Life*, volume 13, pages 387–392, 2012.

[24] Robotis Co. Ltd. Webpage about the dynamixel servos. *http://www.robotis.com/xe/dynamixel_en*.

[25] Lisa Meeden and Deepak Kumar. Trends in evolutionary robotics. In *Soft computing for intelligent robotic systems*, pages 215–233. Springer, 1998.

[26] Jean-Baptiste Mouret, Stéphane Doncieux, and Jean-Arcady Meyer. Incremental evolution of target-following neuro-controllers for flapping-wing animats. In *From Animals to Animats 9*, pages 606–618. Springer, 2006.

[27] Stefano Nolfi and Dario Floreano. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. Bradford Books, 2004.

[28] Nvidia. About the nVIDIA PhysX physics engine. *https://developer.nvidia.com/gameworks-physx-overview*.

[29] Michael J Quinlan, Stephan K Chalup, and Richard H Middleton. Techniques for improving vision and locomotion on the sony aibo robot. In *Proceedings of the 2003 Australasian Conference on Robotics and Automation*, 2003.

[30] B Rosner and D Grove. Use of the mann–whitney u-test for clustered data. *Statistics in Medicine*, 18(11):1387–1400, 1999.

[31] Erol Şahin. Swarm robotics: From sources of inspiration to domains of application. In *Swarm robotics*, pages 10–20. Springer, 2005.

[32] Haocheng Shen, Jason Yosinski, Petar Kormushev, Darwin G Caldwell, and Hod Lipson. Learning fast quadruped robot gaits with the rl power spline parameterization. *Cybernetics and Information Technologies*, 12(3):66–75, 2012.

[33] Jason Yosinski. Webpage about the quadratot. *http://quadratot.yosinski.com/*.

[34] Jason Yosinski, Jeff Clune, Diana Hidalgo, Sarah Nguyen, J Zagal, and Hod Lipson. Evolving robot gaits in hardware: the hyperneat generative encoding vs. parameter optimization. In *Proceedings of the 20th European Conference on Artificial Life*, pages 890–897, 2011.

[35] Juan C Zagal, Javier Ruiz-del Solar, and Paul Vallejos. Back to reality: Crossing the reality gap in evolutionary robotics. In *IAV 2004 the 5th IFAC Symposium on Intelligent Autonomous Vehicles, Lisbon, Portugal*, 2004.

[36] Juan Cristóbal Zagal and Javier Ruiz-Del-Solar. Combining simulation and reality in evolutionary robotics. *Journal of Intelligent and Robotic Systems*, 50(1):19–39, 2007.