UiO : Department of Informatics
University of Oslo

Lifetime learning in evolutionary robotics

Robot control system evolution using memetic algorithms

Else-Line Malene Ruud

Master's Thesis Spring 2015

Lifetime learning in evolutionary robotics

Else-Line Malene Ruud

4th May 2015

Abstract

Inspired by animals' ability to learn and adapt to changes in their environment during life, hybrid evolutionary algorithms which include local optimization between generations have been developed and successfully applied in a number of research areas. Despite the possible benefits this kind of algorithm could have in the field of evolutionary robotics, very little research has been done on this topic.

This thesis explores the effects of learning used in cooperation with a genetic algorithm to evolve control system parameters for a fixed-morphology robot, where learning corresponds to the application of a local search algorithm on individuals during evolution. Two types of lifetime learning were implemented and tested, i.e. Baldwinian and Lamarckian learning. On the direct results from evolution, Lamarckian learning showed promising results, with a significant increase in final fitness compared with the results from evolution without learning.

Machine learning is sometimes used to reduce the reality gap between performance in simulation and the real world. Based on the possibility that individuals evolved with Baldwinian learning can develop a potential to learn, this thesis also examines if learning could be advantageous when such a method is used. On this topic, the results obtained in this thesis showed promise in some sample sets, but were inconclusive in others. In order to conclude in this matter, a larger quantity of samples would be necessary.

Acknowledgements

I would like to sincerely thank my two supervisors, associate professor Kyrre Harald Glette for the continued support during the work on this thesis, and PhD candidate Eivind Samuelsen for the helpful, interesting discussions and for the great help with the simulator.

I would also like to thank my fellow students for all the academic and not-so-academic conversations, and for making the thesis work in the lab feel less like work.

Lastly, I want to thank my family and friends for love and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In robotic system design and development, a large number of components must be considered simultaneously, such as the motor system, morphology, control policy, sensory apparatus, etc. [10]. All of these components are closely interdependent and together determine the behaviour of the robot, making optimization of each component a challenge, since changing of one component is likely to influence the functioning of the others. One way of dealing with this challenge is to examine the entire robotic system as a whole, involving the interaction between body, brain and environment, also called embodied cognition [3].

Nature has mastered this procedure perfectly through the use of evolution, which has produced a vast number of examples of embodied intelligence. Evolutionary robotics (ER) attempts to recreate evolution as a mechanism instead of its biological results, while the latter is often what has been used as inspiration in mainstream robotics. The use of metaheuristics, that is, evolutionary algorithms (EAs) in this case, is the main difference between ER and mainstream robotics. Thus, ER and mainstream robotics also have differing goals, the latter is mainly concerned with optimization of behaviour for a given robot, while the former aims at creating general algorithms for generation of robots, where morphology, control and sensory apparatus can be designed simultaneously [6, 26, 41, 44].

A well-known disadvantage of evolutionary algorithms is that they do not guarantee when or even if the global optimum will be found. Recent studies on hybrid EAs show that including additional methods during evolution can lead to higher robustness and possibly better solutions [24, 30, 33], e.g. by using local optimization of an individual's behaviour between generations. In nature, a similar process is evident. Living organisms have an *a priori* ability to learn during their lifetime, and are thereby able to adapt to changes that may occur in their environment, hence improving their own fitness during life.

In evolutionary robotics, much research has been done on evolution and learning separately, but the effects of a combination of the two have been less explored. Some work has been done on evolution of plastic agents with neural network control systems, were the robot is allowed to explore and learn from its environment in periods during evolution [14]. For parametric

control systems or robot morphology, a different form of learning must be used, e.g. local optimization using a local search algorithm. Even less research has been done on this field, with only a few contributions [32].

Evolutionary robotics has yet to make its way into mainstream robotics. One of the reasons for this is that traditional robot engineering still produces significantly better performing robots, which are more relevant for solving real world problems. In order for ER to catch up with mainstream robotics, a number of different approaches need to be explored. Using a hybrid EA to evolve control systems or robot morphology could lead to enhanced performance of the final solutions. Since traditional evolutionary algorithms are of a largely stochastic nature, and learning offers local optimization of a less stochastic manner, it is possible that learning can also lead to more robustness in the results from evolution.

1.1 Research goals

The main goal of this thesis is to investigate how well a hybrid evolutionary algorithm works for evolution of control systems for a given robot, in comparison with a standard genetic algorithm. The hybrid EA in question is a memetic algorithm, which basically consists of a combination of a genetic algorithm and a local search algorithm. Investigation involves testing the performance of evolution with and without learning under equal conditions, followed by comparison of the results obtained with each configuration. The idea is that learning can aid evolution by moving individuals towards local optima during evolution, thereby increasing the probability of finding the optimal solution, which is not guaranteed using a standard genetic algorithm. In summary, the main goal is to:

- Set up a memetic algorithm which can be used to evolve parametric control systems for a given robot, and then test its ability to successfully evolve solutions in different settings in simulation.

Since evolution of the control system parameters is done in simulation, another topic of interest is how learning affects differences in performance of the evolved solutions in simulation and in the real world. In most cases, the performance of solutions evolved in simulation has considerably worse performance when transferred to the physical robot, mainly because of inaccuracies in the simulator [17, 40]. Whether this decrease in performance is larger or smaller when a memetic algorithm is used should therefore also be examined. Thus, a second goal of the thesis can be summarized as follows:

- Investigate the difference between simulated and real world performance of control system parameters evolved using a memetic algorithm, in comparison with a standard genetic algorithm.

1.2 Outline of the thesis

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

Chapter 2 contains general background information about the theory which this work is based on, including an overview of memetic algorithms and evolutionary robotics, as well as previous research done on these topics. Chapter 3 presents the tools and software used to prepare and conduct the experiments, such as the robot on which the experiments where performed.

Chapter 4 describes the implementation of the memetic algorithm additions, and gives an overview of how the experiments are organized. Chapter 5 then outlines the experiments, and presents the results as well as subsequent short analyses for each experiment. Finally, chapter 6 contains a general discussion of the overall results from the previous chapter, along with a conclusion and suggestions for future work on the topic.

Chapter 2

Background

Nature has always been a source of inspiration in the field of robot design. Evolution has produced organisms that are perfectly adapted to their environments. Consider for example the emperor penguin: living in Antarctica, it faces one of the harshest environments on the planet, with temperatures down to -40°C and intense wind speeds. However, with its specifically adapted morphological features, like its layers of fat and feathers, the emperor penguin has no problem handling this rough environment. Its oily coat and streamlined body also makes it an expert swimmer, and the characteristic dark back and white belly provides good camouflage when hunting [48]. These traits were formed through millions of years of evolution, and leave the penguin well adapted to solve the problems in its environment. Since the results of evolution in nature are this specialized, simply copying them into a specific problem solving robot will probably lead to a design that includes traits unnecessary or even obstructive for solving the problem at hand. Evolutionary Robotics (ER) attempts to mimic the process of evolution instead of its biological results, to create robots that are evolved to handle the environment of a certain problem. Often, the control parameters of an existing robot are improved using this method, but one of the final goals of ER is the evolution of both control parameters and morphology [3, 44]. Although this task is extremely difficult due to the vast number of iterations and the many parameters that are often needed to describe solutions and environments, one advantage of ER is that it is not restrained by biological processes, and can therefore include problem-specific algorithms such as lifetime learning in the evolutionary search. This is what constitutes the essential ideas behind Memetic Algorithms.

2.1 Memetic Algorithms

Evolutionary Algorithms (EAs) are population based optimization algorithms inspired by the basic concepts of biological evolution and genetics, such as natural selection and inheritance. Memetic Algorithms (MAs) use this as a basis, and in addition includes ideas about lifetime learning, such as Lamarck's theory of evolution and the Baldwin effect [1].

2.1.1 Cultural evolution

The word “meme” was introduced by Richard Dawkins in [7], and is used to describe a unit of human cultural evolution, equivalent to “gene” in biological evolution. A meme can be described as a small building block in some kind of knowledge or skill that an individual can acquire by learning. This means that the meme can be modified during the lifetime of the individual, and then, if it is good or interesting enough, be passed on to the next generation. This separates memes from genes, as the latter will be transmitted to the next generation unaltered, if the individual’s fitness is good enough. With this concept of memes in mind, Moscato [29] defined the term Memetic Algorithms. MAs combine the analogies of genes and memes by including local search as a sort of lifetime learning in a Genetic Algorithm (GA).

2.1.2 Genetic Algorithms

GAs are a subset of Evolutionary Algorithms, which again is a subset of Evolutionary computation (EC).

In general, EAs/GAs perform iterative search or optimization of a problem by evolving a population of candidate solutions, or individuals, towards an optimum, by modifying these using mutation and/or crossover operations. The new individuals go through a selection process, where individuals with higher fitness values have a higher chance of being selected to the new generation. The same process is then repeated on the new generation, until a termination criterion is reached, i.e. a certain number of iterations have been performed. For a more detailed description of EAs and its variants, see [12].

Evolutionary computing, as part of the field of artificial intelligence, is becoming an established research area in computer science, and includes a large set of techniques, including EAs and GAs. It has reached popularity due to its ability to perform fast global search on high complexity problems that require satisfactory, but not necessarily optimal, performance, and because it can be applied to a large number of problems without much altering. This can be done because of the fact that it makes few assumptions about the final solution, which also makes it relatively straight forward to implement on different systems.

2.1.3 Lifetime learning

A well-known disadvantage of GAs, and EAs in general, is that there is no guarantee that the global optimum will be found, as the whole process is characterized by stochasticity and lacks the ability to exploit local information. They do, however, guarantee a near optimal solution, as EAs perform well on global search due to their ability to explore large areas of the fitness landscape. The idea behind memetic algorithms is to combine the explorative qualities of GA with the exploitative qualities of heuristic local search, so a global optimum should have a higher probability of being

discovered with this hybrid global-local search approach. The application of local search is sometimes referred to as lifetime learning, see [30]. There are two main models of lifetime learning, Lamarckian and Baldwinian [24, 46]. Lamarckian learning transmits the improvement caused by local search back to the population by encoding the improved phenotype back to a genotype, which is then used in selection and reproduction in the normal GA way. Although Lamarck's theory of evolution has been more or less discarded as a correct description of biological evolution, computational evolution is not bound by biological constraints, and with an applicable phenotype to genotype encoding there is no reason why this could not be implemented. Baldwinian learning is based on the more biologically accepted mechanism of the Baldwin effect [1]. In this method, the fitness of the individual is altered after the local search and thereby affects the selection process, but the results of the local search are not inherited by the new generation. An early simulation of Baldwinian learning can be found in [19], where it was stated that learning alters the search space, and that the shape of the search space indicates if learning will be a positive influence or not. Figure 2.1 shows an example of how a fitness landscape can be altered using Baldwinian learning.

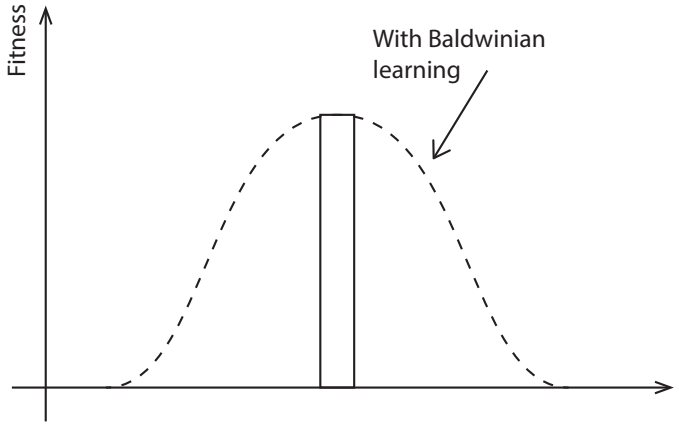


Figure 2.1: Illustration of how the Baldwin effect can affect a fitness landscape. Without learning, the fitness landscape consists of a single spike, so using a GA to optimize the problem would basically reduce it to a completely random search, since each individual would get a fitness of either zero or the maximum fitness. With Baldwinian learning, the fitness landscape is smoothened, since each individual can move towards the optimum at some cost of learning, thereby allowing a wider range of fitness values.

Most successful MAs to date implement Lamarckian learning [24]. Although it poses the additional challenge of encoding the phenotype into a corresponding genotype, the fact that the results of the local improvements are placed back in the population seems to give better results in general.

2.1.4 Overview of the process

Figure 2.2 shows an outline of a standard run of a memetic algorithm. The first step is to initialize a population. This is often done randomly to avoid bias, but can also be done using known information about the problem. Local search is then performed on the initial population, before the main loop is entered, starting with parent selection based on the current fitness values of each individual. In some configurations, not only fitness affects the parent selection, but also other parameters, like age. Mutation and/or recombination of the parents is then performed, generating new individuals which form the new generation, on which local search then can be done. All individuals now represent local optima, or near local optima depending on the configurations of the local search algorithm. If the implementation uses Lamarckian learning, the new individuals receives a new genotype encoded from their improved phenotype before continuing, otherwise only the fitness stays improved. A new population is then made through survivor selection of the old and the new individuals, which can be done using either an elitist or a generational approach. With an elitist algorithm, which is the more common, individuals of high fitness are kept in the new population, as opposed to a generational algorithm, where all individuals of the last population are replaced by the new. This is repeated until a specified termination criterion is reached, often after a certain number of iterations. For a more detailed explanation, see [12] or [31].

2.1.5 Local search and fitness landscapes

The success of the MA largely depends on the choice of local search. This is rarely a non-trivial choice to make, as the effectiveness of different local search algorithms varies over the local structures of the search space, in correspondence with the No Free Lunch Theorem [49]. Much research has been done on the topic of fitness landscapes and MAs, including [5, 24, 28, 36]. [28] emphasizes the importance of fitness landscape analysis when considering MAs, and suggests a few methods for determining both global and local structures. To find the best performing local search, random walk correlation analysis is used to analyse the local structure. A high correlation between neighbouring points indicates a smooth fitness landscape, whereas a low correlation suggests a rugged fitness landscape with many local optima. Correlations between local and global optimums are also examined, to determine the effectiveness of mutation versus recombination based MAs, and to denote the performance of MAs over certain types of landscapes compared to other heuristics. [5] uses the term *phenotypic plasticity* to describe the change in an individual's fitness due to lifetime learning, and demonstrates how this can have a smoothing effect on the fitness landscape. Their findings suggest that this effect is beneficial in rugged multi-peaked landscapes, but that it may slow evolution down if the landscape is shaped by a simple function without multiple local optima. Random walk provides a baseline for their experiments, and is used as a way of measuring evolution rate on different fitness landscapes. [24]

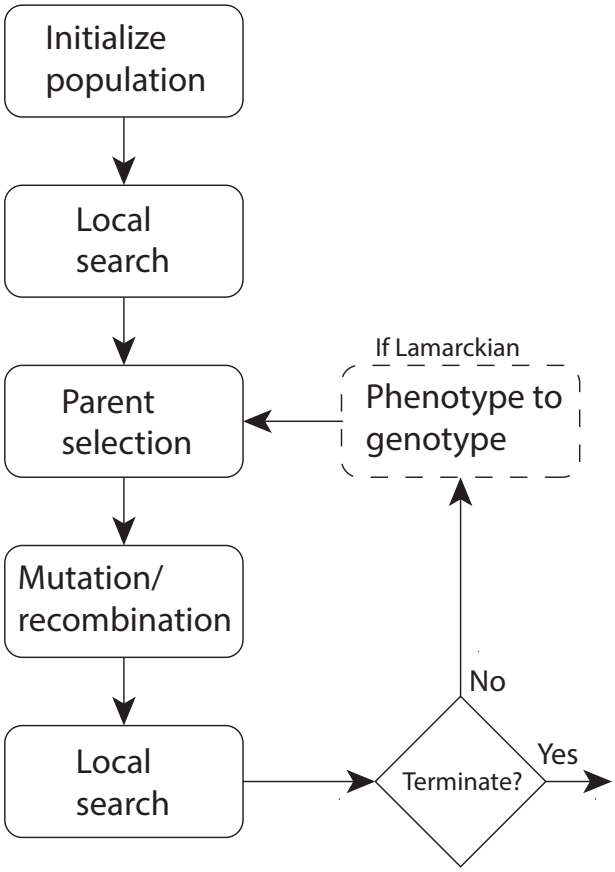


Figure 2.2: Overview of MA process

proposes benchmark analyses for connectivity structures for Lamarckian memetic algorithms, and attempts by this to introduce generality in this field, as most earlier research has been concerned with specific problems and proof-of-concepts. Their results are similar to those in [28], confirming that connectivity/correlation and local optimum structure in general influences the performance of MAs notably. More specifically, local structures of the fitness landscape influence the effectiveness of the local search, while global structures affect the evolutionary meta-search. They suggest using a statistical analysis method, like random walk correlation analysis, to create an idea of the fitness landscape and thereby find the best local search algorithm. In [36], the effects of shifting the balance between individual and population adaptation on changing environments are explored, as a possible solution to limited computational power. This is done by introducing a "lifetime parameter" which sets the degree of individual level versus population level adaptation, or exploitation versus exploration, respectively.

2.1.6 Adaptive MAs

As the choice of local search is of real importance, finding good methods for this has been some of the main topics in recent studies on MAs. [35] proposes a classification of adaptive MAs, and shows how adaptive MAs are capable performing more robustly than traditional MAs. This classification is revisited in [30], and an updated version is proposed. Four main categories are described; Adaptive Hyper-heuristic, where local search algorithms are coordinated using fixed rules; Meta-Lamarckian learning, where the success of the different local searches affects how often they are applied; Self-Adaptive and Co-Evolutionary, where local searches are evolved alongside the candidate solutions; Fitness Diversity-Adaptive, where fitness diversity is used to select the most appropriate local search algorithm. In [34], the term Meta-Lamarckian learning is used to describe methods that use multiple local search algorithms during a run of a Lamarckian MA. Their proposed approach uses a pool of local searches which compete and cooperate during the evolutionary meta-search, thereby avoiding having to manually select the best local search algorithm beforehand. On problems with *a priori* unknown fitness landscapes, this approach could be useful.

2.2 Variants of evolutionary computing

GAs and MAs are only a small subset of a wide range of techniques that constitute the field of evolutionary computing. Although this thesis mainly makes use of memetic algorithms, there are a few other variants which are relevant.

2.2.1 Evolution strategies

Evolution strategies (ES) focus on the use of mutation for the creation of offspring, although recombination is also sometimes used [12]. ESs are mainly used for optimization of continuous parameters, making the typical mutation method addition of random noise extracted from a Gaussian distribution. In addition to the continuous parameters, the genotype also often contains the mutation variables, such as the standard deviation parameter σ representing the mutation step sizes. This means that ESs are typically self-adaptive, as the mutation parameters are evolved alongside the optimization parameters. The assumption here is that different σ's perform differently under different circumstances, i.e. space or time. Self-adaptation can be present on different levels, from standard mutation on an individual level with one σ for all parameters, to correlated mutation on a coordinate level with one σ for each of the parameters, accounting for different degrees of change along different axes of the search landscape. ESs also differ from standard EAs on parent selection, as parents will be drawn from the population when needed by the recombination operator. In general, selection involves a parent individual and λ mutants generated from that parent, which compete with each other for becoming the parent in

the next generation. Three main selection methods exist, $(1+\lambda)$-ES, where the mutants *and* the parent compete for survival, $(1,\lambda)$-ES, where only the mutants compete while the parent is discarded, and $(\mu/\rho,\lambda)$-ES, where μ parents are used in combination with a recombination operator to generate λ mutants.

2.2.2 Multi-objective optimization

One of the main challenges of multi-objective optimization is that problems which have more than one objective will produce a set of optimal solutions, rather than one single optimal solution as is the case with single-objective optimization, due to trade-off between the objectives. This set of optimal solutions is known as the Pareto front, the calculation of which often requires repetitive application of methods, depending on the algorithm. The population based evolutionary algorithms provide an efficient way of computing multiple solutions in one run, making them ideal for optimization of multi-objective problems. Evolutionary multi-objective optimization (EMO) has for that reason been a popular field of research over the last few years, leading to a number of interesting methods and research papers on the subject [8]. Most of these include the concepts of domination and explicit diversity maintenance.

Domination

The concept of domination provides a means to compare multi-objective solutions, and is therefore used in most multi-objective optimization methods. In short, it works as follows: A solution a is said to dominate another solution b if a is no worse than b on all objectives, and if a is better than b on one or more objectives. Both of these conditions must be true for domination to be present. A set of solutions will after this definition have a subset of non-dominated solutions, all of which dominate all the solutions outside this set. This non-dominated set constitutes the Pareto-optimal set or Pareto front, see Figure 2.3.

Diversity maintenance

In EMO and for EAs in general, diversity or spread of solutions is another important concept. For EMOs in specific, diversity is often forced upon the population in order to preserve different niches. Two main methods exist, fitness sharing and crowding. Fitness sharing attempts to place a number of individuals in a niche in accordance with the shared fitness of that niche, before survivor selection is performed. This is done by adjusting the fitnesses according to distances between each individual and the surrounding individuals within a certain distance, defined by a sharing parameter σ_{share}. This method works well for distributing solutions to different niches, but it also has several drawbacks, such as the need to specify the sharing parameter, as well as high computational complexity. If crowding distance is used instead of fitness sharing, the need for a

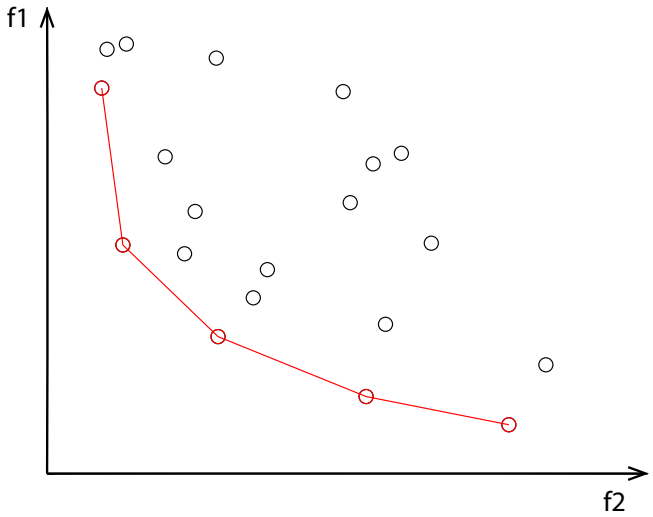


Figure 2.3: Illustration of a Pareto front. Fitness functions f1 and f2 are minimized.

sharing parameter is avoided, and the computational complexity can also be reduced. Crowding distance is calculated as the average distance between an individual and the two closest individuals on either side, in fitness space. Using crowding distance as a diversity measure should lead to an even spread of solutions, by providing an advantage to individuals with large crowding distance when ties are encountered during selection [8].

NSGA-II

One of today's most widely used EMO methods is the non-dominated sorting genetic algorithm II or NSGA-II [8]. NSGA-II is an improvement of NSGA, one of the first and best performing evolutionary multi-objective optimization methods of its time [9]. NSGA-II made changes to some of the main issues of NSGA, by introducing elitism, crowding distance as a parameterless measure for diversity, and a non-dominated sorting approach of less computational complexity than its predecessor (from $O(MN^3)$ to $O(MN^2)$). With these changes, it is able to find solutions near the true Pareto-optimal front, through survivor selection based on both fitness and diversity. Through extensive use, NSGA-II has been shown to have limitations when solving problems with four or more objectives, which has lead to the recent second extension of the algorithm, the NSGA-III [20].

The main loop of NSGA-II starts by combining the current parent and offspring populations, P_i and Q_i, from which the next parent population P_{i+1} will be formed, see Figure 2.4. The solutions in this combined population are then classified using a non-dominated sorting algorithm, resulting in the population being sorted into a set of non-dominated fronts, where F_1 is the best. Thereafter, the new empty parent population is filled with the solutions from the non-dominated fronts, starting with the best front and continuing until the population is of the correct size. Since the

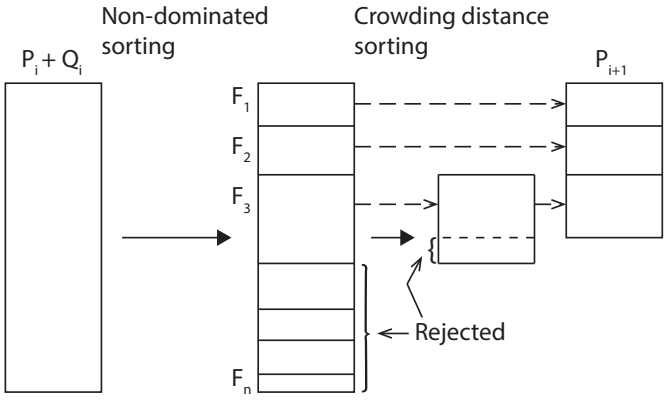


Figure 2.4: Outline of NSGA-II procedure

combined population is twice the size as the new parent population, not all fronts will be included. Solutions from the last front that can be added may not all fit, so solutions are selected according to diversity using crowding distance. A new offspring population Q_{i+1} is then created from P_{i+1} using crowded tournament selection, in which a solution wins if it has a higher rank, or, when the ranks are equal, it has a better crowding distance.

2.2.3 Simulated annealing

Simulated annealing (SA) is a probabilistic search mainly used for global optimization for its ability to escape local optima, but can also be an alternative as a local search algorithm on problems with a rugged search space. Although SA is strictly not part of the EA family because it is not population based, it is related in the sense that it is also a generic metaheuristic. The algorithm is inspired by annealing in metallurgy, and uses a simulated temperature to achieve convergence towards the end of the search. At the beginning of the search the temperature is high, causing large changes with large probability, thereby exploring large areas of the search space without concentrating on local optima. During search the temperature decreases, and along with it the probability of large changes. In other words, the standard deviation of the Gaussian distribution used for creating new solutions decreases, and so does the probability of selecting worse solutions. This probability is called the acceptance probability, and is calculated as

$$P(e', e, T) = \exp(\frac{e' - e}{0.5T}) \tag{2.1}$$

where e' and e are the new and best solutions, respectively, and T is the temperature. e' is here a worse solution than the best so far, if it was better it would have been selected with a probability of one. The standard deviation of the Gaussian distribution is proportional to the square root of

the temperature, $\sigma = a\sqrt{T}$, where a is a scaling factor. The large probability of choosing worse solutions at the beginning of the search prevents getting stuck in local optima before a larger area of the search space has been evaluated. At the end of the search, the temperature is low, and the search converges towards the best local optimum discovered.

2.3 Evaluating performance of EAs

Because of the stochastic nature of evolutionary algorithms, evaluation of their performance usually involves running a number of experiments in order to obtain enough data to provide valid performance measures. In [12], three such measures are listed, mean best fitness (MBF), success rate (SR) and average number of evaluations to a solution (AES). After a satisfactory number of experiments have been run, MBF can be calculated as the average of the best fitnesses over all runs. The best fitness can be defined as the best fitness at termination of the EA run. This gives a good indication of how well the algorithm performs on average. For some problems, best-ever or worst-ever fitness might be more interesting than the average performance, depending on the goal of the algorithm. Success rate is a measure of how often the algorithm succeeds in finding an optimal solution, and is calculated as the percentage of successful runs out of the total number of runs. A successful run must be defined beforehand, this often being that the best fitness at termination is over a certain threshold. On some problems, however, such a threshold cannot be defined since the optimal solution is unknown, and SR is not applicable.

MBF and SR are both measures of effectiveness, but in some situations it is also important to have a measure of efficiency of the EA. AES is defined as a general way of measuring efficiency, independent of processing speed. It is calculated as the average number of evaluations performed before a solution is found. Because it relies on the definition of a solution, it suffers from the same limitation as SR, and is therefore not applicable to all problems. In some situations, it can also be a misleading measure of efficiency, e.g. if there in an imbalance in the duration of the evaluations or there are other parts of the EA cycle that are computationally demanding compared to the evaluations.

Although average performance measures are a good way of evaluating EAs, it is sometimes more interesting to look at peak performance, depending on the type of problem being solved. For design problems, where only one excellent solution is required, peak performance is normally more interesting than average, while for repetitive problems, where results are needed repeatedly and often, average performance is more important.

2.3.1 Statistical analysis

When comparing the performance of two algorithms, or just different configurations of the same algorithm, making a claim of superiority of one over the other should not be done without using a statistical test

that supports this claim, by showing that there is a significant difference between the two. The amount of evolutionary computation methods for optimization done in recent years has demonstrated the importance of statistical analysis for comparison between these methods. Statistical studies usually make use of parametric tests based on average and variance, but recent studies have also considered non-parametric tests for analysis of results [16]. A parametric test is more robust if there is enough knowledge about the problem to make accurate assumptions since the test will be better adapted to the problem, however, this also makes it more restrictive. Non-parametric tests can be used for comparison between algorithms without requiring specific conditions, making them applicable to a wider range of problems in general.

One of the main methods in statistical inference is hypothesis testing, where a set of sample data is employed to test a hypothesis. There are two hypotheses involved in this, the null hypothesis H_0 and the alternative hypothesis H_1, where the latter is the one which is expected to be correct. H_0 is the hypothesis that the sample data are drawn from the same population, meaning that there is no significant difference between the sets of sample data. Similarly, H_1 is the hypothesis that there *is* a difference, that the data is drawn from different populations. To show statistical significance, it is desirable to be able to reject H_0, by concluding that this hypothesis is very unlikely. One way of doing this is to calculate a *p*-value, which can be obtained as a function of the resulting test statistic of a statistical test. The *p*-value shows if a test is significant or not. If it is lower than a certain chosen threshold, usually 1% or 5%, H_0 can be rejected and the test is significant. It is also an indication of how significant the test is, if it is much lower than the threshold, the null hypothesis can be rejected with much confidence. When testing statistical significance, a large sample size is preferred, as this increases the probability of rejecting the null hypothesis, and makes the test more powerful.

In [16], Garcia et. al. state that non-parametric tests are preferable when analysing results from continuous optimization using evolutionary algorithms, because the initial conditions necessary for parametric tests are not satisfied, this being that the data distribution is of a known form. Garcia et. al. used the Wilcoxon matched-pairs signed-ranks test in their study, but in these experiments the similar Wilcoxon rank-sum test will be used, due to the fact that the latter assumes unpaired data, while the former assumes paired.

Wilcoxon rank-sum test

The Wilcoxon rank-sum test, also called the Mann-Whitney U test, is a non-parametric test first proposed by Wilcoxon in [47] and further developed by Mann and Whitney in [27]. The test investigates the null hypothesis that two populations are equal, using a sum of ranks procedure. This involves calculating a U statistic, which indicates if the null hypothesis can be rejected or not. The U statistic is calculated by first sorting and assigning ranks to all observations in the data samples, then adding the

ranks together within each sample, and finally calculating U using the function

$$U = mn + \frac{m(m+1)}{2} - T \tag{2.2}$$

where m and n are the sample sizes, and T is the sum of ranks for the sample with size m. The U statistic can then be used to calculate a p-value. Because of the ranking procedure, the Wilcoxon rank-sum test can only be used on ordinal data. For the same reason, it is a good method for comparing the equality of two samples' median, but this also means that it is not easily adapted to comparison of other parameters, such as the mean difference [42].

2.3.2 Data visualisation

Often, the best way to extract information from a large data set is to visualize it. When evaluating evolutionary algorithms, it is often interesting to see how the population developed through the evolution, e.g. how fast it converges. One way of achieving this is to plot progress curves, showing the mean of the best fitness in every generation in a set of equivalent evolutionary runs, to get an idea of how well the algorithm performs on average, and how efficient it is. To get an idea of how robust the algorithm is, a plot of fitness variance can be used instead. A low variance indicates high robustness, since this means that there is little spread in the results.

It can also be interesting to plot all fitness values in a population for each generation in a single evolutionary run, since this would should how much spread there is within the population. In evolutionary algorithms, a large diversity is usually desired, making this kind of plot helpful. Another way of doing this could be to plot a set of percentiles, showing where most of the solutions lie on the fitness scale. This kind of grouping can also be depicted using a box plot, see Figure 2.5. A box plot shows the groupings of data in the form of quartiles, where the first, second (median) and third quartiles are represented as lines in the box. In addition, limits are shown in the whiskers, and outliers are indicated as dots or crosses outside the whiskers. Such a plot contains a lot of information, and is useful when comparing different algorithms or configurations.

2.4 Evolutionary Robotics

Evolutionary robotics (ER) has yet to become an established part of mainstream robotics. To this day, robot design in general involves manual design of the robot's shape, and the use of machine learning for control policy optimization on the hand-designed robot. This process is extremely time consuming and demands a lot of resources, however, the results still outperform the current results obtained through ER [3]. Still, ER keeps gaining popularity for a number of reasons, one being that improving control policies and morphology can be done automatically while making few assumptions about the final system. This is mainly based on the

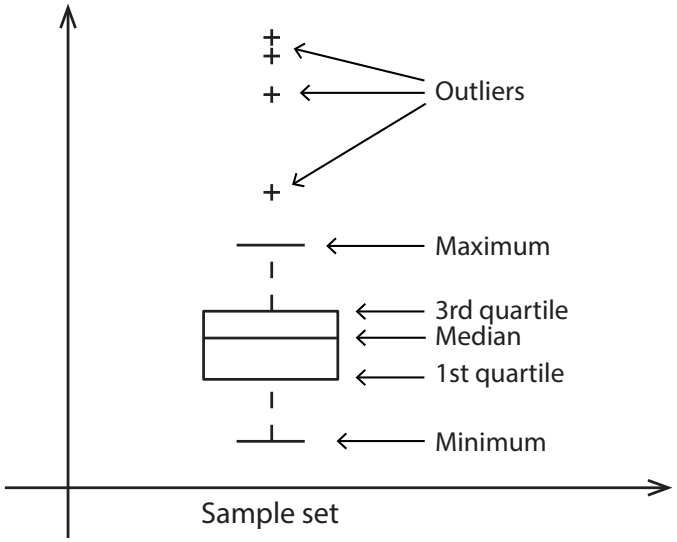


Figure 2.5: Example of a box plot, with descriptions of the different parts.

formulation of a fitness function or a novelty search. Another advantage is that evolutionary algorithms can find solutions that are non-intuitive for human designers. This makes robotic design in ER more efficient, automatic design of robot control and/or morphology can and has created vast numbers of robots, obviously of varying quality. ER has also been shown to exploit rather than fight against morphological or environmental features, e.g. in [15], evolution of locomotion patterns on a monkey-like robot produced a pattern which exploited the momentum of the robots body, similar to what primates do when swinging from one tree to the next.

Another benefit of ER is that it can be interesting not only to roboticists, but also to biologists, as robots might evolve traits that are also seen in nature, and through this possibly help explain why these traits exist.

One disadvantage of ER, and EAs in general, is that the optimal solution is rarely found, and the number of iterations performed before a near optimal solution is found is often vast. Incorporating MAs in ER could lead to an improvement in this area, when looking at some of the promising research done on MAs.

See [3] for a more detailed overview of the field.

2.4.1 Evolving robots

So far, ER researchers have mainly been evolving control policies, often in the form of neural networks. However, full automation of robot design is desirable, and some research has also been done on the evolution of morphology, or more ideally, morphology and control policy simultaneously, starting with [43, 44]. Sims used genetic algorithms to evolve both morphology and control of virtual creatures, which in the end were capable of swimming, walking, jumping or following a light. Similar simulated creatures were brought into the physical world through the GOLEM Project [37].

Although these early attempts at full automation were fairly simple, they show the potential to automatically create natural robots. A more recent experiment conducted by Cheney et al. [6] addresses the issue of lacking improvement since Sims [44], and present a new approach which includes multiple material types, like soft muscles and tissue. They use a CPPN-NEAT encoding [6] which is shown to produce more advanced creatures in simulation, and the multiple materials make evolution of more natural looking creatures possible.

2.4.2 Variants

In addition to classic legged or wheeled locomotive robots, there exists a wide array of variants suitable for artificial evolution, such as modular, swarm, and soft robotics.

Modular robotics

Modular robotics makes repetition or reuse of evolved sections or modules possible, which is also often seen in biological organisms, see [2, 26, 41]. Modularity seems a great advantage when evolving complex morphology, and could potentially increase the evolvability of robotic systems [2].

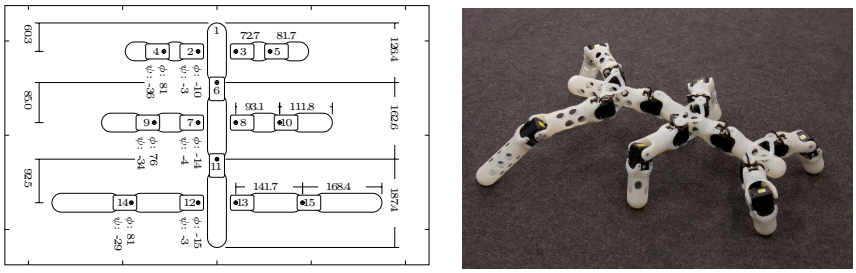


Figure 2.6: Example of a modular robot [40].

Swarm robotics

Swarm robotics takes its inspiration from social insects like ants, in other words, simple individuals that work collectively in a group. Early work includes [38], which focuses on evolving controllers for cooperative behaviour in small, homogeneous robots. The results showed that the robots developed distinct roles in the team, and worked together to complete a coordinated movement task.

Soft robotics

Soft robots are, as the term suggests, built using soft materials in addition to rigid parts. Such robots could potentially handle environments that

standard discrete robots struggle with, i.e. by climbing walls or squeezing through holes. However, this flexibility also makes controller design difficult due to the fact that deformation of one part of the robot will consequently cause deformation of another part. Artificial evolution would then be ideal for controller or morphological design for robots of this kind. This is attempted in [39], where NVidia's PhysX is used to simulate evolution of soft robot gaits, as well as for soft-body modelling.

2.4.3 Challenges

There are still a number of unresolved challenges within ER which can explain why it has yet to produce a robot which is superior to one of manual design. Current research mainly focuses on these challenges.

To achieve evolution of adequate controllers/robots in simulation the task environment must be well described, often demanding a large number of parameters. An increasing expressiveness rapidly increases the evaluation time of each robot, creating a complexity problem, and demanding extensive computational power. [3] mentions co-evolution of robots and task environments and evolvability of algorithms as possible solutions to this problem.

Another key element in EAs is the fitness function, which describes the quality of a solution. It is near impossible to design an unbiased fitness function, resulting in biased solutions, even though one of the goals of EAs is to do as few assumptions as possible about the final result. One possible adaptation of this can be to omit the fitness function and instead do novelty search [25]. Novelty search does not evaluate each individual on a certain performance like the fitness function, it rather compares the difference in functionality between new individuals and what has been discovered before. Significant difference is rewarded, leading to higher complexity, similar to natural evolution.

Reality gap

The large number of iterations needed for artificial evolution to produce good results is the reason why simulation is such an important part of the process. Ideally, evolution would be performed on the physical robotic system, but for most applications this process is too time consuming, and also involves a lot of wear and tear on the physical robot. The continuous and noise filled properties of the real world make creating a realistic physics based simulator a difficult task, and an inaccurate simulator would lead to badly transferable solutions due to the fact that evolution exploits all aspects of the environment. Solutions may become overly adapted to a simulated environment that does not accurately match the real world. This difference in performance between a simulated solution/controller and the transferred real world solution/robot is called "the reality gap". [21] found that adding noise to the simulation can create better correspondence between real world applications and simulations if the level of noise is

appropriate, as noise blurs the fitness landscape in simulation. The more accurate a simulator is, the more time the optimization process will take.

In [50], a different approach is used, namely a back-to-reality algorithm, which does co-evolution of simulator and robot/controller. This involves regular validations of the simulation model in the real world, followed by updates in fitness. [23] also uses a robot-in-the-loop approach, namely the transferability approach, but here a Pareto-based multi-objective evolutionary algorithm is used, where the objectives to be optimized are fitness and transferability. Transferability is measured using a simulation-to-reality disparity measure, which for most potential solutions is approximated using interpolation of a few real world measures. This approach ensures that the optimal solutions found in simulation are transferable, but it does not necessarily find the best real world solutions, which is yet to be done.

2.4.4 Recent work in ER

In addition to some of the work already mentioned, extensive research has been performed on the subject of ER. [11] lists two methods in ER as current trends, evolutionary aided design and online evolutionary adaptation.

Evolutionary aided design

Evolutionary aided design aims at using EAs to find promising strategies, and then using more traditional design techniques based on the results. In this way an evolutionary algorithm is used more as an analysis tool than for optimization, which can be useful for systems where possible optimization parameters are not obvious, possibly because of stochasticity or non-linearity. In [18], this technique is used to find controllers for a homogeneous swarm of micro air vehicles (MAVs). They first applied an evolutionary algorithm to automatically evolve neural controllers, then analysed the resulting behaviours and reverse-engineered these using hand-design to provide simple controllers that were easy to parameterize for different scenarios.

Online evolutionary adaptation

Online evolutionary adaptation involves continuously running an algorithm on the robot, in order to online deal with possible changes on the robot or in the environment. This means that human intervention is unnecessary, which is obviously advantageous where such is unavailable, like in hazardous environments. [45] applied a form of online evolution to a population of robots, called embodied evolution, in which an evolutionary algorithm was distributed among a group of robots which then evolved through mutation and recombination between robots. An advantage of this approach is that evolution could be done outside simulation, thus avoiding transferability issues. Another is that evolution could be done "in the field" without human intervention. [4] also uses an online approach, by

letting a legged robot detect changes in its own morphology, and then adapting the controller by synthesizing new models. In any case, the idea is that optimization is done without interfering with the robot performing its task. Since online evolution is a real-time operation, including MA could be advantageous as it has been shown that MAs can provide solutions more efficiently than plain genetic algorithms in some cases.

2.4.5 Learning with neural networks

Memetic algorithms are not the only area of interest when it comes to combining evolution and learning, much of the recent research has revolved around evolution and learning in the shape of neural networks. In this case, learning corresponds to the training of the neural network during evolution, which is often done in intervals [33]. In [14], a population of plastic individuals were evolved on a small mobile robot using a simple genetic algorithm where the genotype contains a set of parameters describing learning rules and properties of the synapses, but where the weights of the synapses are set to a random low value at each learning stage. The resulting weights after learning are not encoded back into the genotype, as the phenotype and genotype are in different search spaces. In this way, evolution can provide better conditions for learning, and thereby is not only evolution aided by learning, but evolution additionally guides learning. Although this experiment was performed in a stationary environment, it also posed as a step towards a similar experiment with changing environment, as it was expected that this kind of learning would be advantageous in such a setting.

In [33], one such experiment is described, in which the changing environment consisted of the walls changing between being dark or light, causing an impact on the sensor activation levels. The results of this experiment indicated that when learning was activated, individuals were selected partly for their ability to learn successfully, in other words, they evolved a "predisposition to learn" rather than to behave. This was concluded from the individuals' ability to solve the problem before and after learning. Since the non-learning individuals outperformed the learning individuals before learning, while the opposite was the fact after learning, it can be assumed that evolution selects individuals after their success in setting up good conditions for learning. This supports the Baldwin effect hypothesis.

2.5 Combining Memetic Algorithms and Evolutionary Robotics

The fields of Memetic Algorithms and Evolutionary Robotics both lack maturity, so it is not surprising that very little research has been done on combining the two. Still, some work has been done on the field, including [32] and [22]. [32] applies a compact memetic algorithm to a cartesian robot for optimization of the control system, with good results. Their

Memetic compact Differential Evolution (McDE) algorithm is designed for problems where high power computational components are unavailable, i.e. due to space/cost requirements or limited hardware, making it ideal for robotics. In [22], a hybrid genetic algorithm including bacterial foraging was used to optimize the parameters of the PID controller of an automatic voltage regulator. This algorithm showed promising results, and could potentially be used for other similar optimization problems, such as the development of robot controllers.

Another reason for why this is a little explored field, could be that finding a good local search algorithm for ER is not trivial. The performance of MAs is as mentioned highly dependent on fitness landscape, which is usually unknown in ER. However, the stochastic local search used in [32] produced good results, which seems promising for other ER applications. Moreover, although the exact shape of the fitness landscape is mostly unknown, it will in most situations in ER be multi-peaked. MAs have demonstrated good performance on such landscapes, so further research on this is of interest. A possible solution to the problem of choice of local search could be to use an adaptive memetic algorithm, as recent research on this topic indicates that this could lead to good results on unknown fitness landscapes.

Chapter 3

Software and tools

This chapter presents the different tools and software used in the implementation and experiment parts of this thesis. This includes the evolutionary simulation system, the robot, as well as the motion capture system used in the real world experiments. Table 3.1 shows an overview of the software used.

Area of use	Name	Version
Development environment	Microsoft Visual Studio	12.0.30501.00
Figures	Adobe Illustrator	16.0.0
Image editing	Adobe Photoshop	13.0.1
Statistics and graphs	Matlab	2013b and 2014a
Motion capture	Arena	1.7.3

Table 3.1: Table showing the software used during the work on this thesis.

3.1 Simulation system

The evolutionary simulation system developed at the ROBIN research group consists of a framework which uses PhysX and ParadisEO for physics simulation and evolutionary computing, respectively.

3.1.1 PhysX

PhysX [1] is a multi-threaded physics engine SDK managed by NVidia, one of today's main GPU manufacturers, and is widely used in games and by developers. As a middleware physics engine, it provides real-time physics simulations for developers to use without the need to implement complicated calculations for physical mechanics. In other words, PhysX allows developers to simulate physics using high level coding without specific knowledge about physics or simulations. The SDK is free for

[1] http://www.nvidia.com/object/physx_faq.html,
http://www.geforce.com/hardware/technology/physx

commercial and non-commercial use on Windows, and is also available on a series of other platforms [2].

3.1.2 ParadisEO

ParadisEO is a white-box, C++ based, object-oriented framework for implementation and analysis of metaheuristics focused on design and code reuse, which is portable across Windows and Unix systems and licensed under the CeCill license[3]. It provides a range of features, including evolutionary algorithms. The module ParadisEO-MOEO focuses on multi-objective optimization, and its features include multi-objective metaheuristics for evolutionary algorithms, like NSGA-II.

3.1.3 Simulation framework

The ROBIN simulation system, implemented in C++, uses the ParadisEO framework for the evolution of robots, where the ParadisEO-MOEO features are used to run NSGA-II. Figure 3.1 shows a rough overview of the system. PhysX version 3.3 beta-2 is used to simulate the robot and its movements, and thereby evaluate each individual's fitness value. For efficiency, the framework consists of a server-client system, where the server manages the evolutionary part while the client handles the simulator and evaluations. This enables the user to set up a number of clients to perform simulations in parallel, only limited by population size and the availability of computational power.

The main components of the evolutionary part can be said to be the EvolutionManager and the RobotEvolver, in addition to the RobotGenes. Before evolution starts, the EvolutionManager sets up the evolution parameters from a configuration file and then initiates the RobotEvolver, which generates the initial population using the specified RobotGenes and handles the evolution. During evolution, the RobotEvolver sends out jobs to the simulation part through a WorkManager, so each individual can get its objective vector assigned.

On the simulation side, the SimWorker receives a blueprint describing the corresponding individual which is to be simulated, from the server. It handles this using an instance of EvorobEvaluation, which, among other things, sets up the simulator, the simulation environment and the evaluators. The simulation environment is again managed by EvorobScenario, which has access to the blueprint, simulator and simulated robot, making it ideal for specifying custom environments or behaviour.

3.2 The robot

A simple four-legged modular robot was used, one which had previously been developed at the Robotics and Intelligent Systems research group

[2] http://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/guide
[3] http://paradiseo.gforge.inria.fr

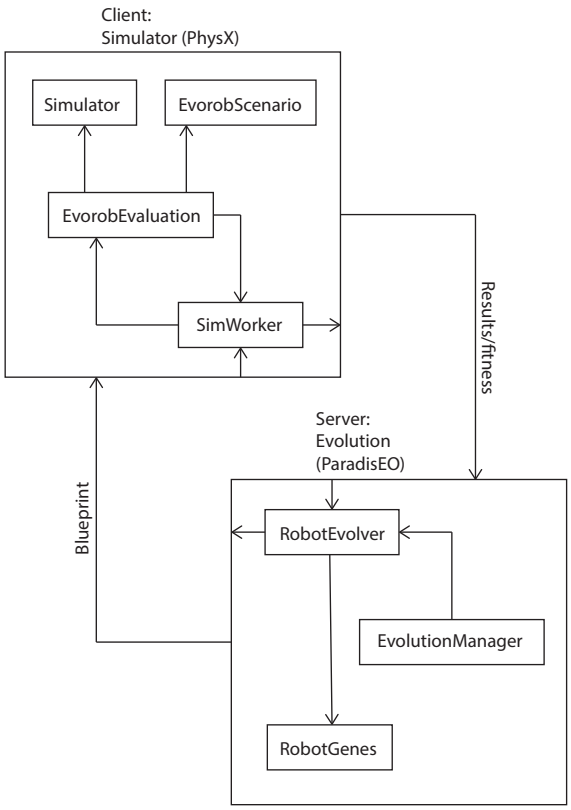
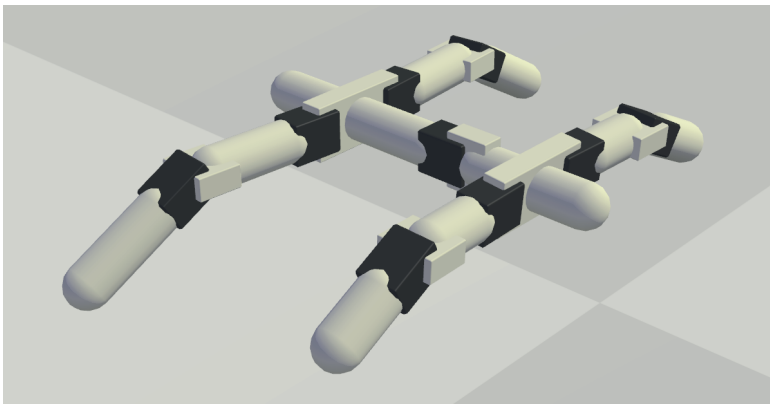


Figure 3.1: Overview of simulation framework

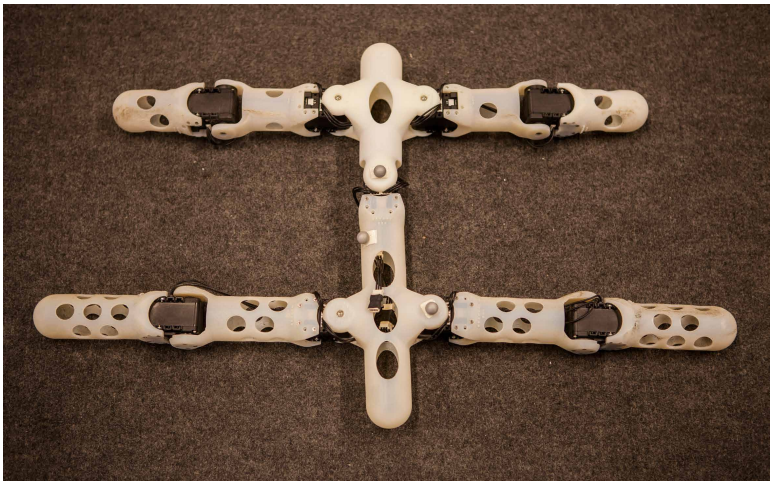
(ROBIN) at the Department of Informatics, University of Oslo. A simulated model was evolved in simulation through evolution of both morphology and control, which was then used to create a physical version to be tested in reality. The full procedure is described in [40], where the robot used in these experiments is listed as number three in the set of five robots which were used there. The control system parameters that were originally coevolved with the morphology of the robot result in a bounding gait, where the front legs are in phase and the rear legs are in phase, but out of phase with respect to each other. After evaluation of this gait in simulation and in the real-world, it was found that the performance decreased somewhat when tested on the physical robot, mainly due to the difference in friction between the simulated environment and the carpet floor in the lab.

The robot has nine degrees of freedom, made up of nine revolute joints, where one is a body joint, four are hip joints and four are knee joints. See Figure 3.2 for images of the robot.

In the simulator, each part of the robot, excluding the joints, is modelled by a white capsule of a specific length and radius, with hinges where they are connected. See Figure 3.3 for an overview of some of the specifications for the robot. If the morphology was evolved further, these specifications would change during evolution, but for the experiments conducted here,



(a) In simulation



(b) The physical robot

Figure 3.2: The robot

only the control system was evolved, so the specifications supplied here are fixed. Each joint is modelled as a revolute joint with an accompanying motor which powers it. These motors are modelled as linear servos with the specifications of a Dynamixel AX-18A servo, and have the shape of black boxes with the dimensions of the same servo.

The physical version of the robot has been build using 3D-printed parts based on the specifications of the simulated model, where each bodily part has been printed in the form of a hollow plastic capsule with sockets for attaching motors in the joints. They were printed on an Object Connex 500 multi-material 3D printer, using a material consisting of a mixture of VeroWhitePlus and DurusWhite, called DurusIvory [40].

As for the simulated model, the joints of the physical robot are powered by Dynamixel AX-18A servos. Each printed part has a slot where the servo fits, see Figure 3.4. There are certain limitations to the physical robot compared to the simulated robot which are worth mentioning, one of which is that the servos are for obvious reasons unable to apply infinite force, which could happen in simulation if robot limbs get locked in impossible

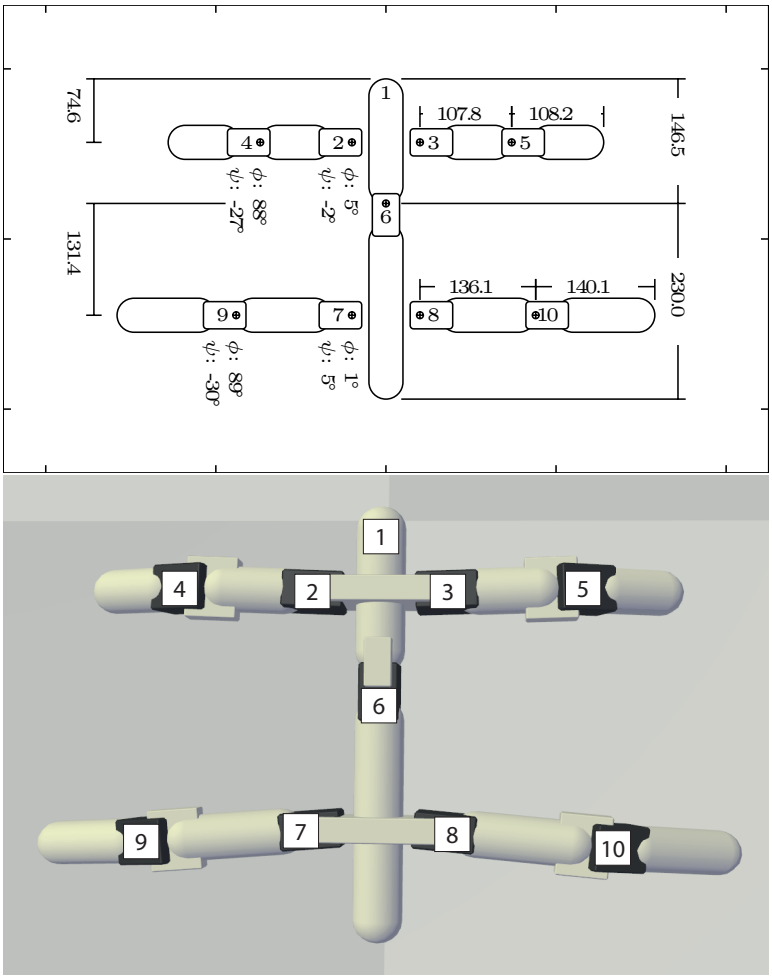


Figure 3.3: Above: Schematic view of the robot, borrowed from [40]. Lengths are in millimeters. Below: Equivalent view of the robot in simulation. Parts are numbered according to the above schematic.

positions. The simulation framework should, however, recognize these situations and filter out the corresponding individual.

3.2.1 Control systems

There are several possible control systems which are applicable, two of which are used in the experiments. In [23], a periodic controller was used to govern the movement of each DOF of a hexapod robot, described by the following function

$$\gamma(t, \alpha, \phi) = \alpha \cdot tanh(4 \cdot sin(2\pi \cdot (t + \phi))) \tag{3.1}$$

where α and ϕ represent amplitude and phase shift, respectively, and t is time. In other words, the movement of each joint is determined by two parameters, making the total number of parameters for the robot 18. A low number of parameters is advantageous for evolution, as it makes the search

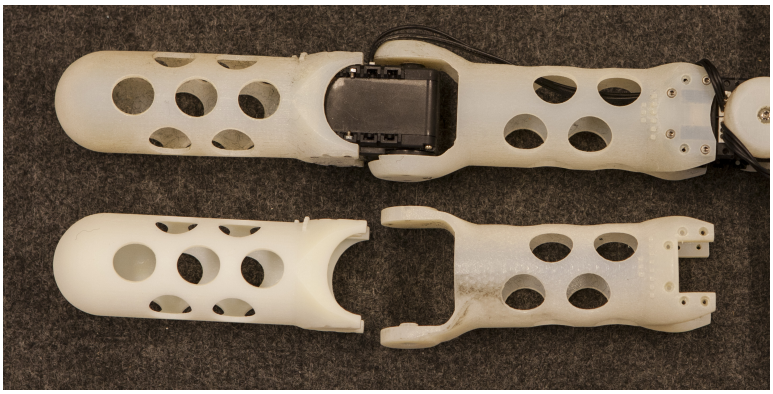


Figure 3.4: Back leg of the robot, with and without servos mounted

space reasonably small, thereby increasing the chance of finding the global optimum as opposed to a larger number of parameters. The tanh function is used to keep the signal constant in parts of the cycle, thereby allowing the robot to stabilize itself in these periods.

The search space can be decreased further by introducing symmetry to the system, by managing the movement of each *pair* of joints, instead of each joint separately. Such a simplification is supported by observations from nature, where this kind of symmetry is prevalent. The way this is done is by letting each joint pair share the amplitude parameter, but keeping separate phase parameters to enable the two joints to move out of phase with respect to each other. For the first joint, the first phase parameter represents the phase, while for the second joint, phase is calculated by addition of the two phase parameters.

One limitation of this controller is that movement will always be centered around the same point, defined by the zero position of the servo. To enable movement around an arbitrary centre, the controller can be generalized by adding an offset parameter, thus allowing for a larger range of gait patterns, and possibly also the range of robots on which it can be used. The periodic function describing this controller can then be written as

$$\gamma(t, \alpha, \phi, \delta) = \alpha \cdot tanh(4 \cdot sin(2\pi \cdot (t + \phi))) + \delta \tag{3.2}$$

where δ represents the offset. However, this generalization increases the number of dimensions and thereby the search space, making it computationally more demanding to find optimal solutions with optimization methods for robots with many DOF. For this robot, the number of parameters is 27 with this control system. Again, dimensionality can be reduced by introducing symmetry, by letting each joint in a joint pair share the amplitude and offset parameters, reducing the number of parameters for each joint pair from six to four. Figure 3.5 shows a plot of the joint motions of two instances of these controllers.

In the remainder of this thesis, the first control system with only phase and amplitude parameters will be referred to as *AmpPhase*, while the

second control system which also includes offset will be referred to as *AmpOffPhase*.

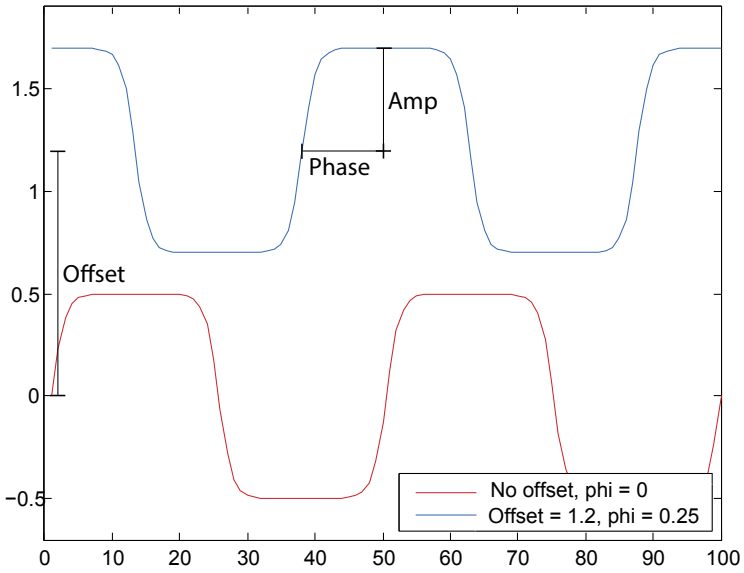


Figure 3.5: Plot representation of joint motion. The blue line represents a controller where offset $\delta = 1.2$, phase $\phi = 0.25$ and amplitude $\alpha = 0.5$, while the red line represents a controller with no offset, and where $\phi = 0$ and $\alpha = 0.5$.

3.3 Motion capture system

There exists a wide range of motion capture techniques commonly used in robotics, such as inertial measures using accelerometers and gyroscopes, magnetic methods with measurements of electromagnetic fields, mechanical measurements of joint orientation displacement using potentiometers, or acoustic techniques involving ultrasonic transmitters and microphones [13]. In addition, there is optical motion capture, which is probably the most widely used technique. Optical motion capture systems consist of one or more cameras which detect the movement of specific markers which are placed on the robot. These markers can be either active or passive, depending on the requirements of the system. Passive markers reflect infrared light generated from the cameras to a larger extent than most other materials, thereby enabling tracking of these markers through thresholding of recorded light. Active markers emit light themselves, permitting longer distances between camera and marker. Another advantage is that each marker can be easily identified by sequentially switching on each marker, while passive markers are sometimes swapped due to disturbances. However, active markers require power sources to either be wired or attached to the robot, which can be avoided with passive markers.

Figure 3.6: OptiTrack Flex 3 cameras in the Motion Capture lab.

The motion capture equipment at ROBIN consists of a set of 12 OptiTrack Flex 3 cameras[4] which track passive markers, see Figure 3.6 for pictures of the cameras. The cameras emit infrared light, and record video of up to 100 frames per second, with a 640 x 480 resolution. Through a USB 2.0 interface, the cameras send data to the Arena software (version 1.7.3 in the lab) for recording and analysis of rigid body tracking.

[4]http://www.optitrack.com/products/flex-3/

Chapter 4

Implementation and experimental setup

In order to have the option of lifetime learning in the evolutionary search, a number of additions have been made to the evolutionary simulation system developed at ROBIN, which was used in the project. The first part of this chapter focuses on the implementation of these additions as well as the genome of the robot, while the second part gives a general description of how the experiments are set up.

It should be mentioned that the terms *learning* and *local search* are used somewhat interchangeably here, and that *lifetime learning* is only used about learning during evolution, while *learning* means that local search is applied to a solution, which can also happen after evolution.

4.1 Incorporation of local search in the evolutionary framework

In a memetic algorithm, local search is applied to all offspring individuals before they undergo survival selection with the parents. Since the fitness landscape is unknown in these experiments, and each phenotype must be tested in simulation in order to find the fitness, each control system found in each iteration of the local search must be evaluated in simulation. This means that local search must be included on the client side of the simulation system, since phenotypes must be systematically evaluated throughout the search. At the start of every evaluation in the simulator, a scenario is used to include custom environments and behaviours. The scenario has access to both the simulator and the simulated realization of the robot, making it ideal for incorporating a local search procedure.

4.1.1 Learning scenario

Figure 4.1 shows an overview of how local search was incorporated in the evolutionary framework. A scenario called LearningScenario was implemented, which was used to set up the necessary parts for local search.

The LearningScenario has access to all relevant information about the robot, such as the control system parameters and the simulated model, as well as to the simulator itself, allowing it to make changes to the simulator directly.

LearningScenario includes local search by creating an instance of SimAdapter, which is added to the simulator. The SimAdapter manages the local search, by handling the robot score during simulation and by updating the local search evaluations at regular intervals. It also has access to a list of chosen local search algorithms, where only the first one will be used during evolution, as well as a set of parameters controlling the evaluation details. The robot is evaluated in the simulator over a set of evaluation periods, resulting in a mean score which can then be used in the local search algorithm.

The local search algorithm is implemented as a ControlLocalSearch, where the main loop of the algorithm is in the evaluate function. ControlLocalSearch contains a controllable subject and its control parameters, and can only be used for local search in control systems. During search, the best score and accompanying control parameters found so far are stored continuously, so they can be easily fetched after the search is finished. This happens when a set number of iterations are completed. Every time the evaluate function is called, one iteration of local search is performed.

4.1.2 Choice of local search

The performance of the local search in a memetic algorithm largely depends on the choice of local search and the shape of the fitness landscape [34]. In this case, the fitness landscape is unknown, making it difficult to know which local search algorithm best suits the problem. However, since each iteration of local search requires an evaluation of the solution in the current state, there is a trade-off between the number of generations in evolution and the number of iterations of local search, which limits the duration of learning. Thus, a hill climbing procedure might be a good choice, since they are able to find improvements fast. In these experiments, two local search algorithms were tested, *one plus lambda* (OPL or $(1 + \lambda)$) and *simulated annealing* (SA), where the former was used in the majority of the tests, while the latter was mainly used for comparison. SA is described in the background chapter, being a well-known optimization algorithm, while OPL is presented in the following section.

One plus lambda

One plus lambda is a generation based hill climbing local search algorithm, based on the evolution strategy $(1 + \lambda)$-ES. In the first generation, the start solution is used to generate lambda new solutions by adding small random values drawn from a Gaussian distribution to the control parameters. After the new solutions have been evaluated, the best one is chosen, and the procedure is repeated from this solution. There are two search parameters that must be set in advance, lambda and a reevaluation

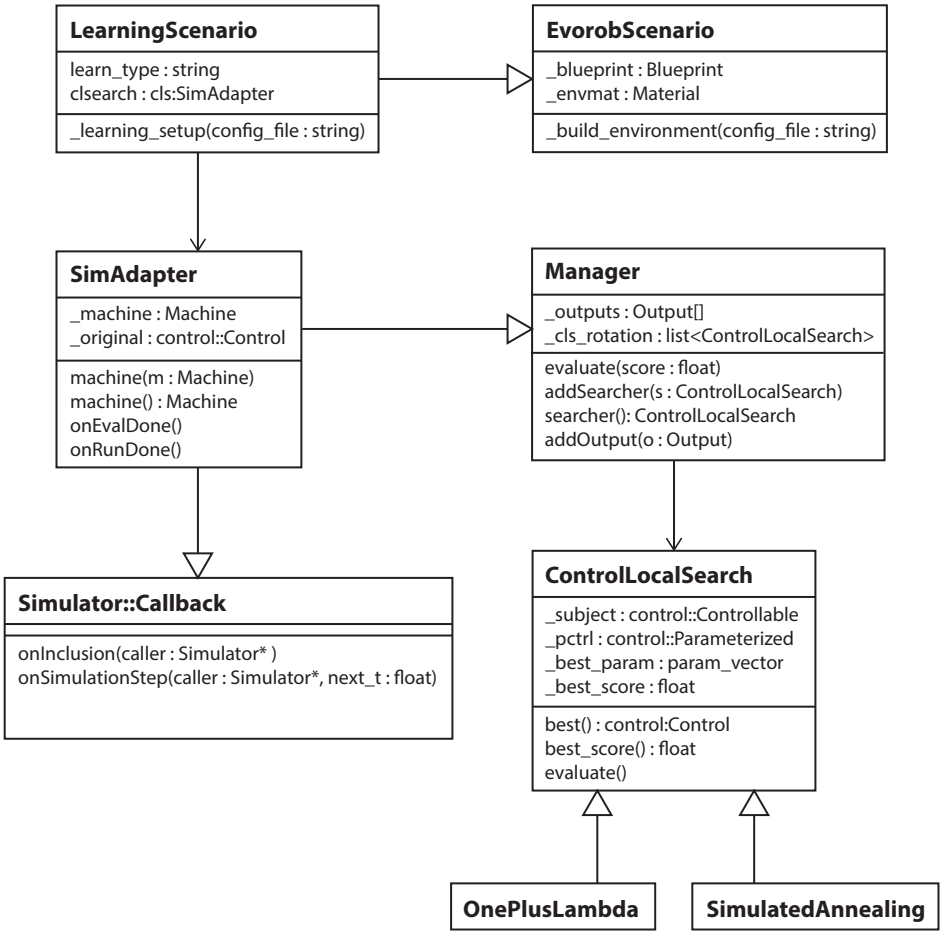


Figure 4.1: Overview of the incorporation of local search. Only the main variables and functions are included. The illustration borrows concepts loosely from UML, arrows with unfilled arrowheads indicate inheritance, while the other arrows show associations. Variables are listed before functions. Underscore before variable names indicate private or protected variables.

parameter. This reevaluation parameter is the main difference from $(1+\lambda)$-ES. If no improvement has been detected for a certain number of iterations, determined by the reevaluation parameter, reevaluation takes place. A reevaluation results in the individual's score being assigned to the mean score of the new evaluation and all previous reevaluation means, thereby reducing the possible problem of noisy measurements. For the evolution experiments here, the reevaluation parameter is set to something larger than the number of iterations of local search, so there are in practice no reevaluations during evolution.

A low value of lambda would lead to a greedier search than with a large lambda, since less new solutions are generated around the current best solution in each generation. In these experiments, a small lambda could be advantageous, since this would result in a deep search from the start

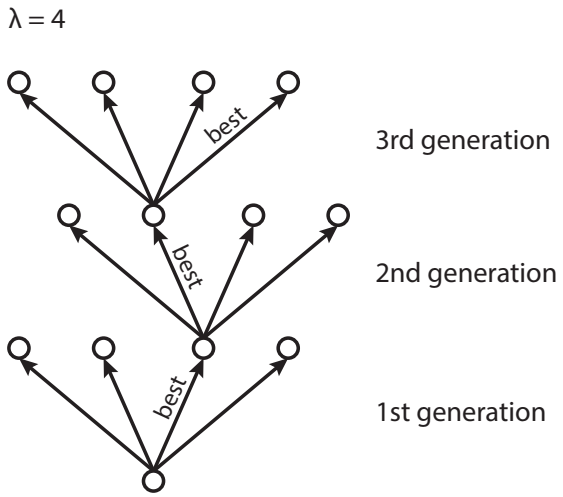


Figure 4.2: Basic illustration of OPL search algorithm, over three generations with lambda set to four. One iteration corresponds to one evaluation in simulation, meaning that this illustration includes 12 iterations of search.

solution. However, less of the fitness landscape will be evaluated in close proximity of the best solutions, so a possible local optimum might not be discovered. With a larger lambda, the search becomes wider and more refined, as more of the area around the current solution will be evaluated, leading to a larger possibility of approaching the true gradient, but at the cost of a less deep search.

Because *simulated annealing* is more stochastic than OPL, it does not exploit local structures as well as OPL, but it has a better chance of escaping local optima. In a memetic algorithm, OPL is probably a better choice of local search than simulated annealing, since the genetic algorithm already provides global exploration of the search space, which will hopefully be enough to discover areas of high fitness.

4.2 Genetic algorithm and robot genes

The simulator framework developed at ROBIN uses NSGA-II for the evolutionary computations, for efficient evolution on multi-objective optimization problems. Since these experiments only consisted of single-objective optimization, NSGA-II would not have been necessary, but was still used since it was already part of the system. However, a traditional elitist genetic algorithm would probably have performed adequately, and would possibly have had a lower complexity. When NSGA-II is used with a single objective, crowding distance essentially becomes redundant, since every front will only consist of one individual, unless there are fitness values that are identical. In practice, all individuals will simply be ranked after fitness

value in one-dimensional space, and the N best will be selected for survival, N being the population size.

4.2.1 Robot genome

As the robot is of a fixed morphology, only the control system needs to be represented in the robot genes. The control system consists of a set of parameters which are stored in a vector in a parameterized control class. The number of parameters depends on the choice of control system, 15 with the symmetric version of the *AmpPhase* system described in [23], and 27 with the asymmetric version including offset. All joints are paired with the corresponding joint on the opposite side of the body to allow symmetry, except for the waist joint. When robot genes are generated for new individuals, the control parameters are initialized to zero, and then mutation is run ten times on the genes to create a wide spread of individuals.

Distance between genes is calculated as Euclidean distance divided by the square root of the number of parameters

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\frac{\sum_{i=1}^{n}(x_i - y_i)^2}{N}} \tag{4.1}$$

4.2.2 Genetic algorithm variation operators

In order to create diversity in the population, two variation operators were implemented: mutation and recombination.

Mutation

Since the control parameters are represented by floating-point numbers, a continuous distribution is used for mutation, in this case a Gaussian distribution. Mutation is performed by adding an amount randomly drawn from the Gaussian distribution to each parameter, with probability one. A Gaussian distribution with mean $\mu = 0$ and standard deviation $\sigma = 0.1$ is used, as this will cause most of the mutations to be small, two thirds will be within one standard deviation, while the less likely possibility of large mutations is still present. Mutation can then be described as

$$child_i = parent_i + \mathcal{N}(0, \sigma^2) \tag{4.2}$$

where *i* indicates the *i*th allele in the gene.

Recombination

An arithmetic linear crossover was implemented, in which two parent individuals create two offspring. This is achieved in the following way: For each parameter in both parents, the distance between the two parents is calculated, simply by subtraction of the first parent from the second. This distance is then weighted, with a different weight for each parent drawn

randomly from a uniform distribution, and either added to or subtracted from the original value of the corresponding parameter, for parent one and parent two respectively. After recombination, the resulting children will lie somewhere along the geometric line between the two parents in genotype space, where the distance from the parents is dependent on the weights.

$$d_i = parent_{2,i} - parent_{1,i} \tag{4.3}$$

$$child_{1,i} = parent_{1,i} + \alpha_a d_i \tag{4.4}$$

$$child_{2,i} = parent_{2,i} - \alpha_b d_i \tag{4.5}$$

α_a and α_b represent the weights for the first and second parent respectively, drawn from the uniform distribution $\mathcal{U}(0, 1)$.

4.2.3 Measuring fitness

The robots are evolved with a single objective, to maximize forward movement. This is measured as the displacement of the robot over a set of time periods, where one period is equivalent to one second, corresponding to one cycle in the control system [40]. More specifically, each control system is evaluated over eight time periods in simulation, after which the mean displacement per period is assigned as the final evaluation value. Distance is measured in the direction the robot was headed at the start of the period. In the start position for the first evaluation, the robot is headed along the negative z-axis, since the ground plane is on the x-z plane, as shown in Figure 4.3.

When local search is included, the simulator runs evaluations for every iteration of the search before assigning the score to the individual. The score is set to the best result that was obtained in the local search. During search, the simulator goes through all the iterations of the search without resetting the robot in the start position. After evaluation, the fitness of the individual is set to the evaluation score. Figure 4.4 shows an illustration of how the evaluation procedure works.

4.3 Experimental setup

The main goal of this thesis is to investigate how a memetic algorithm performs when used to evolve robotic control systems. A large part of the experiments therefore involves simulated evolution of robots, to obtain data containing results with different learning configurations, including the baseline evolution without learning. In brief, the simulator was used for the evolutionary experiments, evaluation of evolved gaits and for running local search on the evolved solutions, while only the two latter of these experiments were done on the physical robot.

4.3.1 Evolutionary setup in simulation

A number of different configurations were used when running evolution, in order to investigate how the memetic algorithm performs in different

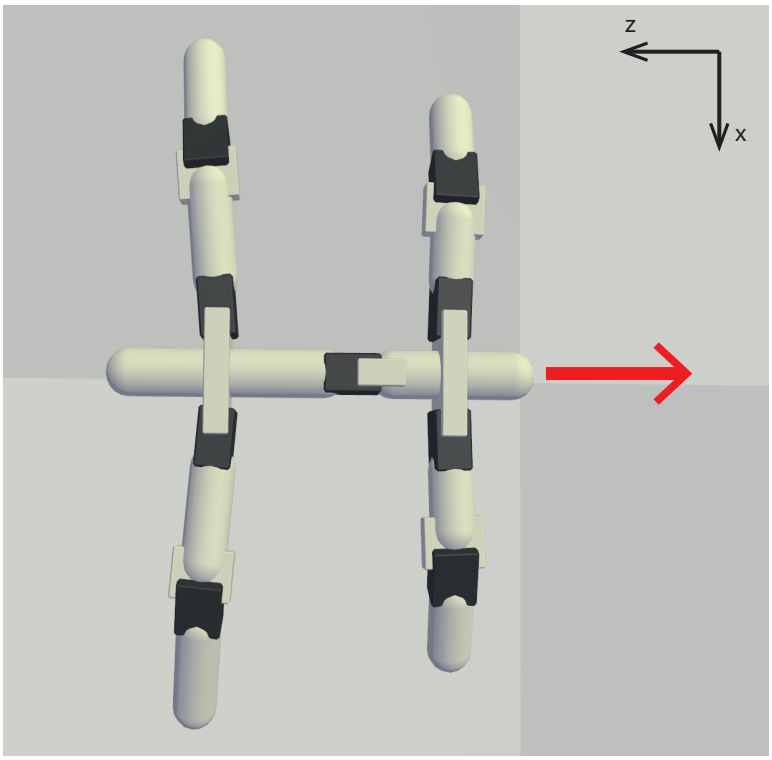


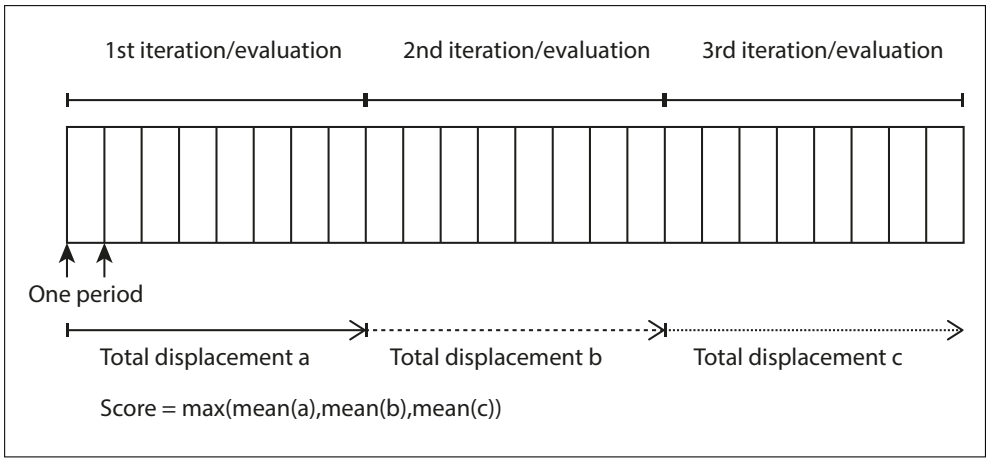
Figure 4.3: Start position of robot, showing the direction in which movement is measured.

scenarios. Table 4.1 shows the possible choices for each setting in a configuration, which are specified before the start of an evolutionary run. If OPL is chosen as local search, the parameters involved in this search must be specified as well.

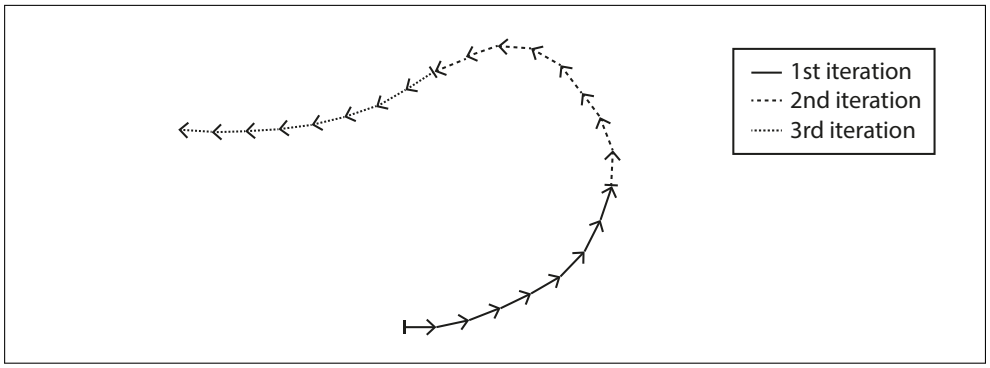
In every generation of an evolutionary run, local search is performed over a chosen number of iterations for every individual in the population. Because an iteration of local search involves one evaluation of a control system in the simulator, as can be seen in Figure 4.4, the number of generations and iterations must be balanced. Without learning, each individual is evaluated once in each generation, but with learning, each individual is evaluated as many times as there are iterations of local search in each generation. The number of iterations of local search must therefore be inversely proportional to the number of generations, and is chosen so the total number of evaluations, basically iterations times generations, is the same for comparable runs. This is calculated as

$$generations_{ls} = \frac{generations_b}{iterations_{ls}} \tag{4.6}$$

where $generations_b$ is the number of generations for the baseline configuration, typically evolution without local search, and $generations_{ls}$ and $iterations_{ls}$ are the number of generations and iterations of the configuration with local search, respectively. To make sure the number of evaluations is the same for comparable runs, the number of iterations are



(a) Total displacement in each evaluation equals the sum of the displacement over each period, as illustrated in (b).



(b) Example path of robot in motion.

Figure 4.4: Illustration of how a robot is evaluated in simulation, over three iterations of local search.

chosen so that the remainder is zero. This is assuming the population size is the same for the comparable runs, if it differs, this must also be taken into account. In these experiments, the population size is the same in all experiments, making this easy to handle. The value of λ in OPL will not affect this relationship, as it only determines how wide the search is, and not the number of evaluations.

For each configuration, a set of 30 evolutionary runs were done, to be able to compare performances.

4.3.2 Changing environments

In addition to testing different configurations of the memetic algorithm, some evolutionary experiments were done with a changing environment in the simulator. Because learning individuals exploit local structures of the fitness landscape, it is possible that they will be better able to adapt to sudden changes in the environment. Two kinds of environments were implemented in addition to the default, one with sphere obstacles, as shown

Learning	Baldwinian Lamarckian
Local search	One plus lambda Simulated annealing
Crossover	None Linear
Iterations	Positive integer
Generations	Positive integer
Control system	AmpOffPhase AmpPhase symmetric

Table 4.1: Possible configurations for evolution.

in Figure 4.5, and one with low friction. In the obstacle environment, sphere obstacles with a radius of 0.025m were spread over the ground plane, with a spacing of 0.2m with an offset of 0.05m in the x-direction for every other row. The obstacles are fixed in place, and cannot be moved around by the robot. The offset was added to prevent the robot from walking between and along the obstacles. The low friction environment was implemented simply by changing the friction parameters of the ground plane, see Table 4.2 for the specific parameters.

Friction parameters	Static	Dynamic	Restitution
Original env.	0.2	0.15	0.4
Low friction env.	0.01	0.005	0.3
Robot body	0.3	0.3	0.3

Table 4.2: Friction parameters in the simulator.

4.3.3 Physical robot setup

The same setup as in [40] was used for the physical robot, and the experiments performed in the motion capture lab at the ROBIN area. The experiments are automated, so the robot can move about over a longer period of time without needing to be reset in the starting position. If the robot is about to move out of bounds, the control system is replaced by one that turns the robot left or right, and the robot turns until it faces the centre of the work area. The turning control systems were already developed through evolution. If this happens in the middle of an evaluation, the results of that evaluation are discarded, and a new one is attempted when the robot is ready to continue.

The robot is wired for power and control signals. To avoid the robot getting tangled in the wire and possibly unplugging itself, the amount of loose wire is controlled by a motorized pulley system, see figure 4.6. The motion capture equipment was used to track the robot displacement, and

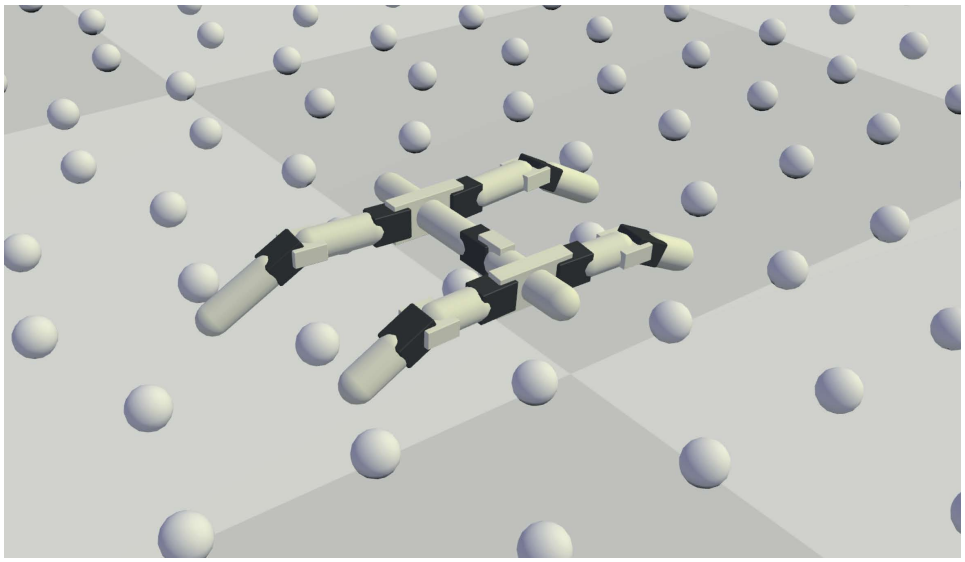


Figure 4.5: Environment with sphere shaped obstacles.

to detect the position of the robot, which is used in cooperation with the motorized pulley system and the turning mechanism. Because the motion capture system uses passive markers, three reflex balls where placed on central parts of the robot, as shown in Figure 4.7.

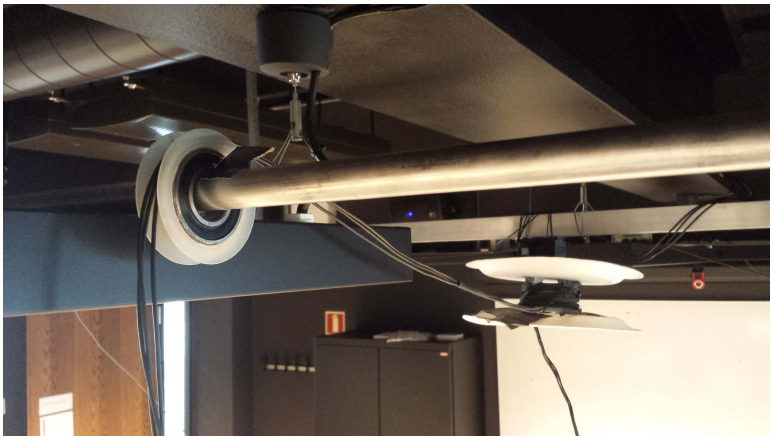


Figure 4.6: Pulley system mounted near the ceiling.

4.3.4 Evaluation of performance

Although the fitness value is based on the movement of the robot, and thus indicates the quality of the robot gait, it may not be an adequate description of the gait performance over time. The fitness value represents the mean movement over a limited set of periods, so to get a better evaluation of how the control systems perform over a longer time, a final evaluation of the evolutionary solutions can be done. Because this is a time consuming process, only the most interesting solutions were evaluated in this manner.

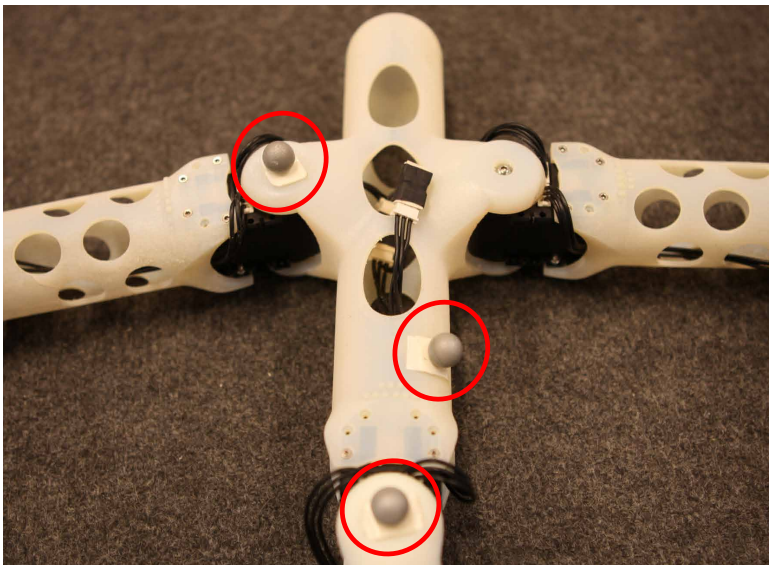


Figure 4.7: The passive markers attached to the robot.

Evaluation is done in the same way as fitness measuring during evolution, but with fewer periods per evaluation, more specifically, 4 instead of 8. This reduction in evaluation length is partly because a control system is tested over a large number of evaluations, making long evaluation time unnecessary, and partly because of the limited work space of the physical robot. When a control system is evaluated on the physical robot, an evaluation is stored if it is completed before the robot reaches the edge of the work area, so the robot should for efficiency be able to complete at least *one* evaluation when starting at an edge.

This evaluation setup is used both in simulation and on the physical robot, to make evaluation results comparable across the two platforms.

4.4 Aspects of learning and evaluation

When implementing and using a memetic algorithm for evolution of robot control systems, a few topics should be addressed, such as the possibility of implementing Lamarckian learning and the effects of lifetime learning on the reality gap.

4.4.1 Baldwinian and Lamarckian learning

Implementing Lamarckianism is not always trivial, since it involves encoding the improved phenotype back into the genotype, which can be a difficult or impossible task if the phenotype and genotype search spaces are separate. However, in these experiments, only the control systems are evolved, so local search is only applied to the control system parameters. These are the same in both genotype and phenotype space, so improved control parameters that have been discovered during local

search can simply be assigned as the individual's new control parameters. Where Lamarckian learning is possible, it has been shown to find better results than Baldwinian learning in most problems, since the results of local search are stored directly. Since Lamarckian learning easily can be implemented for this problem, both types of learning are interesting. It is to be expected that Lamarckian learning will lead to better results than Baldwinian learning, however, since Baldwinian learning in theory should store a potential to learn, it is possible that the results of running local search after evolution is finished have a larger improvement than without lifetime learning or with Lamarckian learning. This can be interesting in a reality gap point of view.

4.4.2 Testing on the physical robot

For various reasons, the physical robot was only used for a few tests, as opposed to the simulated version which was tested on extensively. Firstly, the physical robot is subjected to wear and tear, making it unsuitable for running evolution on, as this involves running a vast number of evaluations on the robot. This would eventually cause damage to the physical robot, leading to a halt in the experiments where damaged or worn parts must be reprinted and replaced. The simulated model will obviously not have this problem, and can be used to run an unlimited number of evaluations without reduction in accuracy or deviation from the original model. Secondly, running evolution on the physical robot is extremely time consuming, and requires constant supervision as it can get tangled in its wires for power and signals. The latter can be avoided if the robot was running on batteries and it had its own attached controller, but this is not the case with the robot used in these experiments. In addition, the robot may need to be repositioned in a starting position. In simulation, none of these problems exist. Although simulated evolution can still be time consuming, it will not be near as demanding as in the physical world, since evaluations can be sped up considerably in comparison. The simulated robot will never be obstructed by wiring, and it is automatically placed in the start position at the beginning of evaluations.

These are some of the reasons why a simulated environment is considered a much better suited platform for running evolutionary experiments than a physical robot, however, the ultimate goal is not to find a robot that can perform optimally in simulation, but one that will also perform well when transferred to the physical robot. This is all but trivial, as modelling a simulator which perfectly portraits the physical world is impossible. A good solution in simulation might thus not correspond to a good solution in reality, the simulated solution may become overly adapted to the simulator by exploiting features that don't exist in reality, or there may be aspects of the real world setting which are hard or impossible to simulate accurately. This difference between simulation and reality forms the basis of the notorious reality gap problem.

4.4.3 Learning and the reality gap

An interesting question now is whether learning will affect transferability or not. Since local search exploits local structures of the fitness landscape, one could expect that overfitting with respect to the simulator may be more frequent with this kind of learning, seeing as the fitness landscape in simulation will probably differ from the real world fitness landscape. However, if the shapes of the fitness landscapes are approximately the same, transferability should not be much affected. Making a prediction on this is difficult, since the shape of the fitness landscape is unknown.

If learning is applied on the resulting solution after evolution, that is, the robot is allowed to learn while running, solutions which were evolved using Baldwinian learning might have an advantage over non-learning solutions when it comes to transferability, since they might have a better potential to learn. This can only be the case if the fitness landscapes are of about the same shape.

In the setup used for these experiments, the main difference between the real word and the simulator is friction. In [40] it was found that most materials used for body parts had significantly lower friction against the floor carpet than the material modelled in the simulator.

Chapter 5

Experiments and results

The experiments were conducted focusing on comparison between different settings of a memetic algorithm and a normal genetic algorithm. In addition, the different configurations of the memetic algorithm were compared. The first part of the experiments consisted of evolving control system parameters under different conditions in simulation, the results of which were used in the later experiments. The second part includes evaluation of selected evolved gaits in both the simulator and on the physical robot, and finally running local search on the same gaits to examine their learning abilities.

Each experiment is presented in the same way. First, the experiment details are explained, including a table with the specific settings. Then the results are presented and analysed. It should be noted that the symmetric control system which uses amplitude and phase parameters only is referred to as *AmpPhaseSym*, while the asymmetric control system with amplitude, phase and offset parameters is called *AmpOffPhase*.

5.1 Evolving control system parameters

A few investigatory experiments were done initially, involving testing the two chosen control systems to get an idea of their relative performance. They were done with both Baldwinian and Lamarckian learning, as an initial comparison between the two and the baseline evolution without lifetime learning. Following this, a series of different configurations were tested. Since *one plus lambda* (OPL) was expected to outperform *simulated annealing* (SA), most of the experiments used the former as the local search algorithm, while the latter was only tested in the investigatory experiments. In the experiments that use OPL as local search, a λ of 2 was used. This was because a low λ gives a deep search, possibly resulting in individuals moving closer to local optima, which should be advantageous in these experiments. In order to keep the number of parameters low, further tuning of λ was not performed here.

Unless otherwise specified, each data set in the experiment consists of 30 evolutionary runs, and the total number of evaluations over one evolutionary run is 1600, corresponding to 1600 generations in normal

evolution without lifetime learning. All of the experiments use a population size of 40 individuals.

5.1.1 Investigatory experiments - *AmpPhaseSym*

Local search	**Parameters**
OPL	λ: 2, reevaluation: 100
Control system	**Crossover**
AmpPhaseSym	None
Iterations	**Learning**
1, 10, 20	Baldwinian/Lamarckian

Table 5.1: Experiment details for first investigatory experiment

The first experiments conducted involved comparing the performance of Baldwinian and Lamarckian learning, both against each other and against evolution without learning. It was expected that Lamarckian learning would outperform Baldwinian learning, at least in the immediate results retrieved after evolution. Lamarckian learning was also expected to be able to perform slightly better than evolution without learning, because of its ability to exploit local structures. This obviously depends on the choice of local search algorithm. However, no assumptions were made on how well Baldwinian learning would perform next to evolution without learning, since it was unknown how much of an advantage its learning potential would provide.

A investigatory experiment was set up to test this, where evolution with Baldwinian and Lamarckian learning was tested with 10 and 20 iterations, as well as two baseline evolution sets without learning. See Table 5.1 for details. When the number of iterations of local search is one, corresponding to no learning, Lamarckian and Baldwinian learning are obviously the same. Their results could therefore be merged, but were kept separate here to make sure that they actually are approximately equal.

Learning	Iterations	Mean fitness	Median fitness	Best fitness
Baldwin	1	0.4177	0.426	0.5113
	10	0.393	0.4105	0.4923
	20	0.3651	0.3729	0.4515
Lamarck	1	0.413	0.4299	0.5432
	10	0.4332	0.451	0.5323
	20	0.4771	0.4711	0.5849

Table 5.2: Results from the final generations in all runs, with *AmpPhaseSym* as the control system. The mean and median is calculated over the best fitness in the final generation of each run. The best result in each column is marked with a grey background.

Results

As Table 5.2 and Figure 5.1 shows, the assumption that Lamarckian learning should outperform Baldwinian learning holds true with this setup, with a significant difference between the sets of fitnesses after the final generation. What is more interesting, Lamarckian learning seems to yield better final results than without learning. The difference between Lamarckian learning with 20 iterations and no learning in the final generation is significant, based on a two-tailed Wilcoxon rank-sum test with a significance level of 0.01. When the two sets without learning were merged, the resulting p-value from the Wilcoxon rank-sum test was 0.0011. The results from the configuration with 10 iterations of Lamarckian learning are not significantly larger than the results without learning, but there is a certain tendency towards slightly better performance. Figure 5.2 shows that the standard deviation decreases with increasing number of iterations of local search in this experiment, and especially so for the configurations with Lamarckian learning.

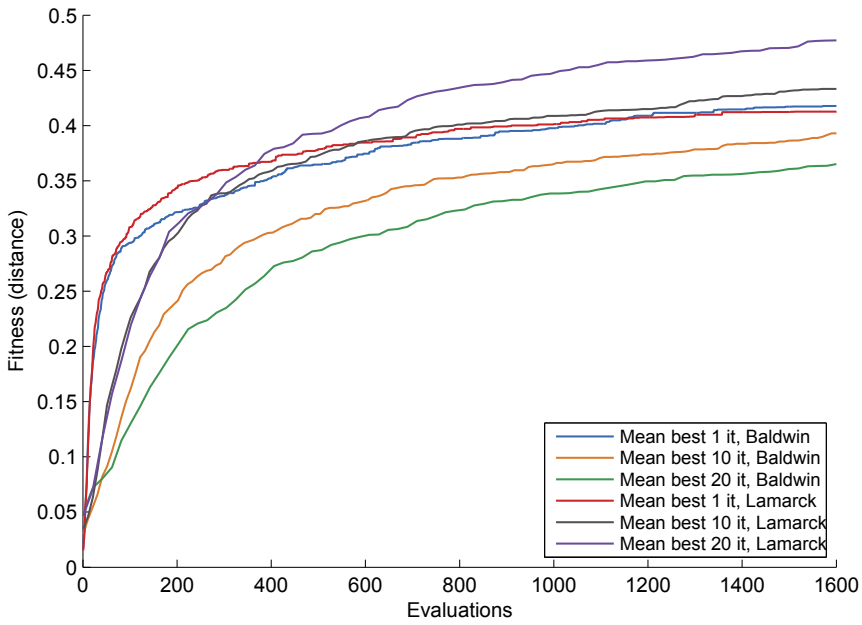


Figure 5.1: Fitness plot showing the mean of the best fitness in each generation, with both Baldwinian and Lamarckian learning.

Analysis

The results from the Lamarckian learning configurations indicate that a certain minimum number of iterations of OPL are necessary before the effects of the local search is beneficial over the baseline genetic algorithm. With 20 iterations, the search can reach further than with 10 iterations, and thus does more local exploration which increases the chances of finding the local optimum. This seems to be an advantage over evolutionary selection

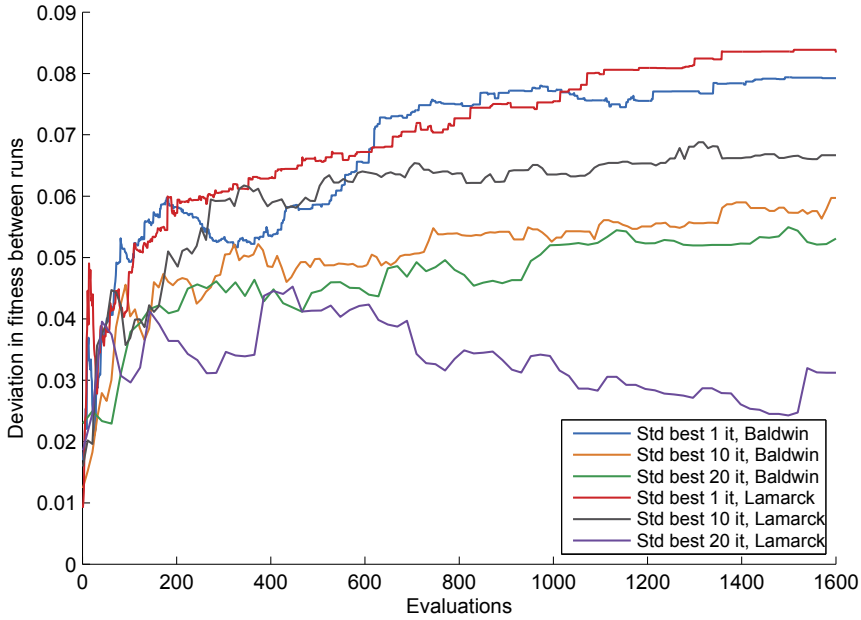


Figure 5.2: Plot of standard deviation of the best fitnesses in each generation.

over more generations. Whether there is a ceiling number of iterations where this ceases to be the case is the topic of Subsection 5.1.5 in this chapter.

Figure 5.1 shows that the lifetime learning configurations take longer to converge than the baseline. The reason for this is probably that there is less global exploration when local search is included, so it will normally take longer to find an area of high fitness. When a new generation is generated, more global exploration is achieved due to the variation operators of the genetic algorithm. With a memetic algorithm, generating a new generation requires more evaluations than with a normal GA because of the local search, resulting in it taking longer to achieve the same level of global exploration.

The difference between Lamarckian learning and Baldwinian learning is as expected. Baldwinian learning obviously performs local exploration as well, but since it only stores the new fitness value and not the discovered results in the control parameters, the fitness value only says if it is close to a good solution or not. This seems not to be enough to outperform evolution with no learning, probably because a possible good solution that was discovered during local search will only be used if it is found by chance during evolution, because of the randomness in the variation operators. Comparing the two configurations of Baldwinian learning with 10 and 20 iterations, it can be concluded that increasing the number of iterations leads to less global exploration because it reduces the number of generations in evolution, and thereby reduces the chance of finding high fitness areas. An individual that has a high fitness value because it is close to a local optimum will probably still affect the evolution positively, but

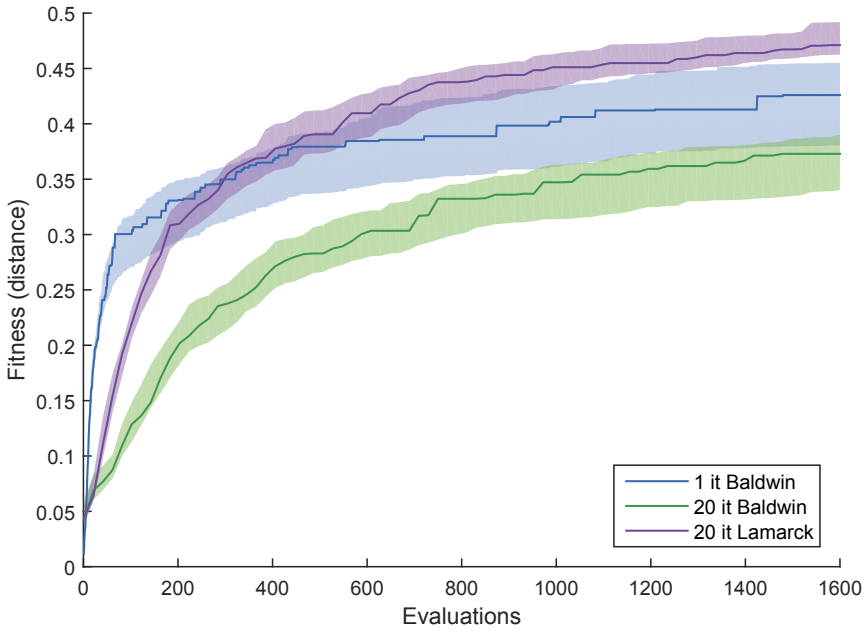


Figure 5.3: Plot with confidence intervals and median, with a 99% confidence level. Only three configurations are plotted to avoid too much overlap.

how much of an effect it has will largely depend on the fitness landscape. In this case, it is not enough to perform better than the baseline, at least on the immediate evolutionary results. There could still be a learning potential stored in each individual at the end of evolution which may be advantageous, a topic which will be addressed further in section 5.3.

When examining the plots in Figures 5.2 and 5.3, it seems that including local search leads to more robustness in the final results. Both Baldwinian and Lamarckian learning lead to lower standard deviations between runs, and it seems that the standard deviation decreases when the number of iterations is increased. This is probably up to some limit which is impossible to see out of these results only, but will be discussed in Subsection 5.1.5. The reason why this is the case could be that evolution with learning more often finds an area of high fitness, possibly the same area over most runs.

5.1.2 Investigatory experiments - *AmpOffPhase*

Local search	**Parameters**
OPL	λ: 2, reevaluation: 100
Control system	**Crossover**
AmpOffPhase	None
Iterations	**Learning**
1, 10, 20	Baldwinian/Lamarckian

Table 5.3: Experiment details for second investigatory experiment

The second investigatory experiment is almost identical to the first, except that it utilizes the *AmpOffPhase* control system instead of *AmpPhase symmetric*. This was done to examine how the memetic algorithm performed on two different systems, to test if the results would be similar and whether the method is robust.

In order to be able to directly compare the results of this investigatory experiment and the previous, the same setup was used in both experiments. See Table 5.3 for the specific details.

Results

Overall, the final fitness values with *AmpOffPhase* were slightly better than with *AmpPhaseSym*, with an increase in mean final fitness of around 0.1 for all configurations, as can be seen in Table 5.4.

Learning	Iterations	Mean fitness	Median fitness	Best fitness
Baldwin	1	0.5211	0.5472	0.6402
	10	0.4241	0.4185	0.6393
	20	0.4256	0.4274	0.5155
Lamarck	1	0.4976	0.5371	0.5915
	10	0.5497	0.558	0.6293
	20	0.5564	0.5581	0.7094

Table 5.4: Results from the final generations in all runs, with *AmpOffPhase* as the control system. The best result in each column is marked with a grey background.

Apart from this, the same tendencies discovered in the previous experiment are also apparent here, though not as strongly. Both with 10 and 20 iterations, Lamarckian learning outperforms no learning, but the Wilcoxon rank-sum test with a significance level of 0.01 shows that the difference is not significant. This can also be seen in the plot in Figure 5.6, where there is significant overlap between the 20 iterations Lamarckian and the no learning confidence intervals. However, the plot in Figure 5.4 shows that there is a definite tendency towards higher performance with Lamarckian learning. As in the previous experiment, the solutions found

with Baldwinian learning are significantly worse than without learning, for the same reasons.

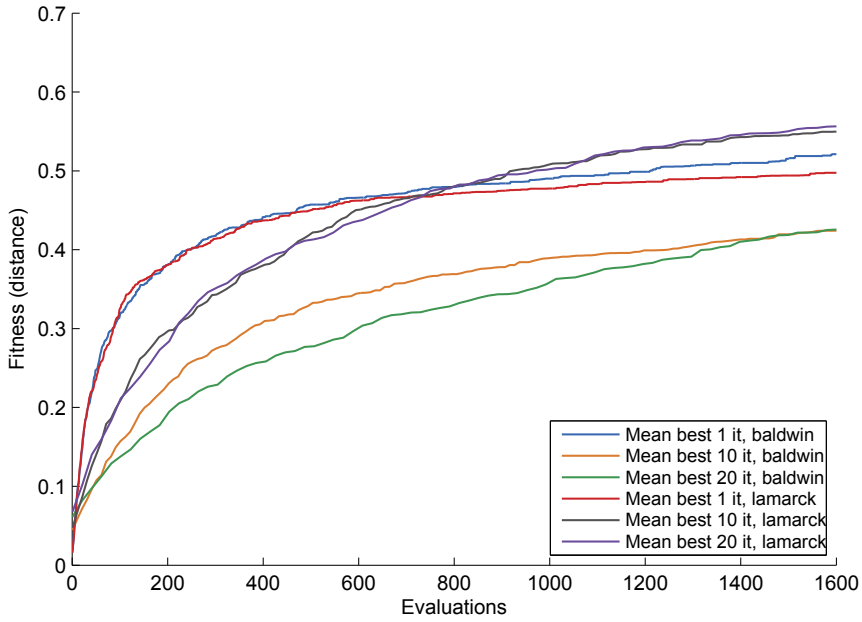


Figure 5.4: Fitness plot showing the mean of the best fitness in each generation, with both Baldwinian and Lamarckian learning.

Analysis

Since this control system is asymmetrical, and also includes an offset parameter, it has a larger total number of evolvable parameters and therefore a larger search space. This allows for a wider range of solutions, some of which will be better than the ones found in the previous experiment, which explains the increase in overall best fitness. However, this also means that the solution deviations will be larger, as is evident in the plot in Figure 5.5, which shows that the standard deviation between runs is larger in all configurations as compared with Figure 5.2. In other words, using *AmpPhaseSym* leads to a more robust system when it comes to final solution deviations, although *AmpOffPhase* opens for better solutions overall in the immediate results after evolution.

Another effect of an increased search space is that the many dimensions make finding the true gradient more difficult for the local search. This could explain why the effects of Lamarckian learning are less clear in this experiment as compared with the previous. It is possible that a different local search algorithm or a change in the OPL parameters would have been better for this configuration. If the lambda parameter was increased, more of the surrounding search space would have been explored in each generation of the OPL algorithm, increasing the chances of finding the true gradient, as explained in Subsection 4.1.2. Whether this would be advantageous on this control system or not is not known, and would require

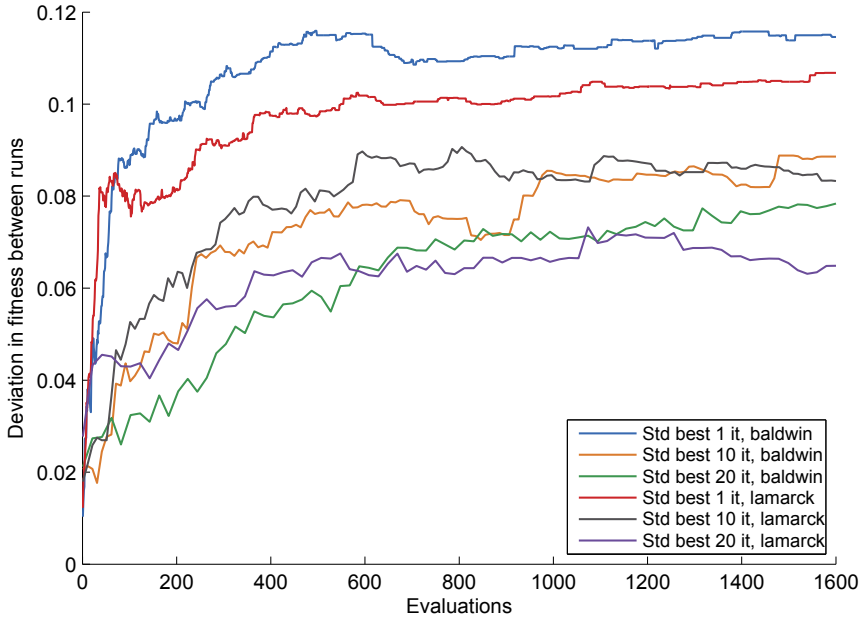


Figure 5.5: Plot of standard deviation of the best fitnesses in each generation.

additional testing.

Because of the increase in dimensions, there is a possibility that the probability of overfitting solutions to the simulator is larger. In addition, it was found that one or more joints often became locked in a fixed position when the offset parameter was close to or exceeded its maximum servo value, see Figure 5.7. This indicates that the *AmpOffPhase* solutions might be less transferable to reality, depending on how the physical servos handle a maximum value offset. This could probably have been avoided if this parameter was constrained, disabling it from exceeding this maximum value. However, because the *AmpPhaseSym* solutions seem to be more transferable to reality, this was not pursued further in these experiments. *AmpPhaseSym* is for this reason used in the remainder of the experiments.

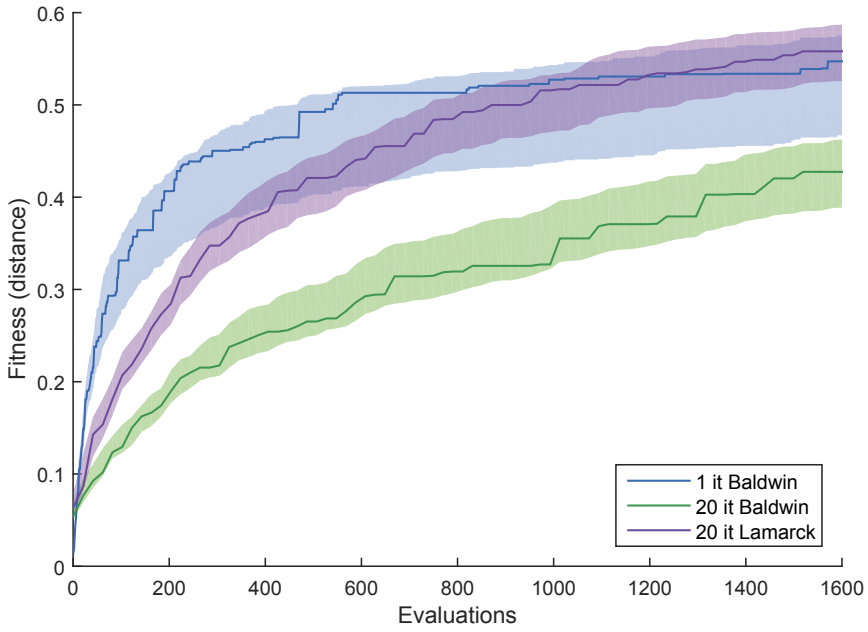


Figure 5.6: Plot with confidence intervals and median, with a 99% confidence level. Only three configurations are plotted to avoid too much overlap.

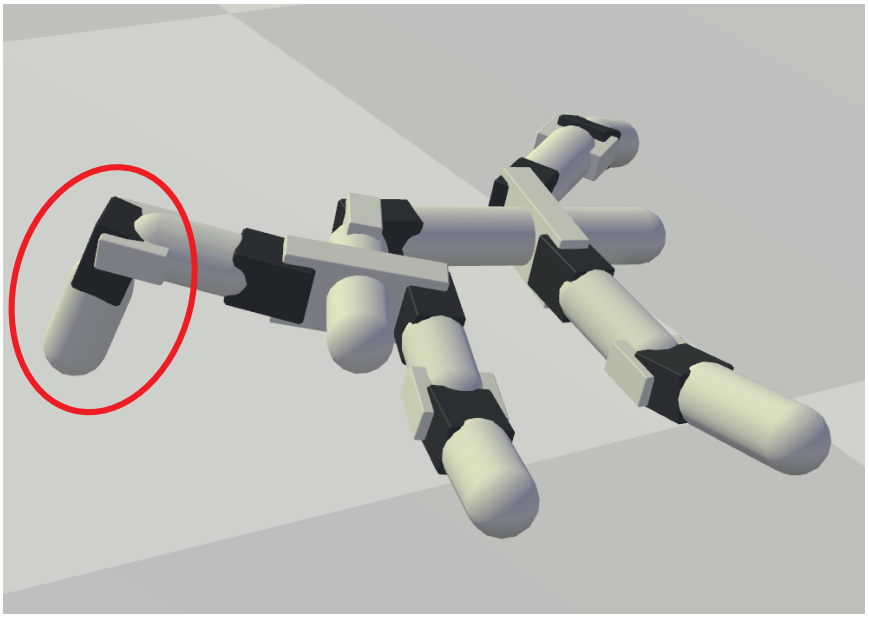


Figure 5.7: Screenshot of the robot with the best fitness from the last generation of the runs with 20 iterations of Lamarckian learning. Because the offset value for joint four, indicated by the red ellipse, is close to or exceeds the maximum value, the joint gets locked in this position.

5.1.3 Investigatory experiments - simulated annealing

Local search	**Parameters**
SA	
Control system	**Crossover**
AmpPhaseSym	None
Iterations	**Learning**
1, 10, 20	Baldwinian/Lamarckian

Table 5.5: Experiment details, with simulated annealing as local search

In order to test the performance of the memetic algorithm with a second local search algorithm, an experiment was set up with *simulated annealing* (SA) as the choice of local search. The hypothesis was that SA is a less appropriate local search algorithm for the problem, as described in Subsection 4.1.2, so it was expected that the performance would decrease compared with the results from when OPL was used. Only 20 runs were done on each configuration because of this, and only one set of evolution without learning was done. See Table 5.5 for the remaining experiment details.

Results

Table 5.6 shows that evolution without learning outperforms evolution with both Baldwinian and Lamarckian learning, with a significant difference with a significance level of 0.05. The configurations with Lamarckian learning achieves slightly better results than the ones with Baldwinian learning, but when looking at the fitness plot in Figure 5.8 it becomes evident that this difference is not present during the whole evolution. The configuration of evolution with 10 iterations of Baldwinian learning is slightly better than the one with 20 iterations of Lamarckian learning over the first part of the evolution, which is never the case in the experiments with OPL.

Learning	Iterations	Mean fitness	Median fitness	Best fitness
None	1	0.4282	0.4248	0.5548
Baldwin	10	0.3522	0.3496	0.4794
	20	0.3283	0.3401	0.4114
Lamarck	10	0.3856	0.3888	0.4857
	20	0.374	0.3734	0.4905

Table 5.6: Final results with SA as local search algorithm. The best result in each column is marked with a grey background.

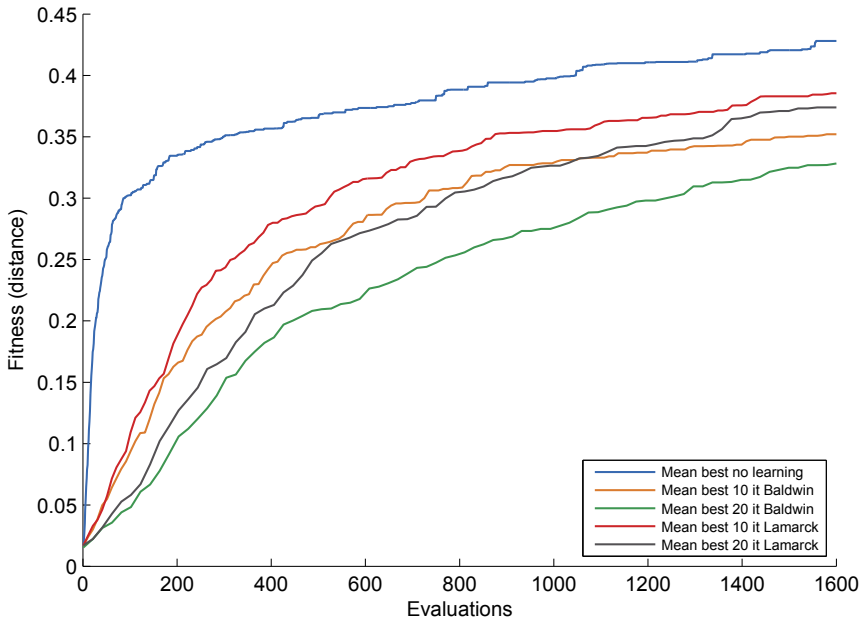


Figure 5.8: Fitness plot showing the mean of the best fitness in each generation, with both Baldwinian and Lamarckian learning.

Analysis

Comparing the results from this experiment with the previous ones where OPL was chosen as the local search algorithm, shows that the choice of local search algorithm in a memetic algorithm is important, and that the local search algorithm must be chosen to fit the problem at hand. In this experiment, the benefits of the local search were not large enough to increase the performance of the genetic algorithm, and instead decreased performance by slowing down the evolutionary search. This is especially clear when studying the plot in Figure 5.8, where not only evolution without learning outperforms evolution with learning in all cases, but where also the configuration with 10 iterations of Baldwinian learning is slightly better than the configuration with 20 iterations of Lamarckian learning in the first part of evolution. More generations of evolution is clearly advantageous to more iterations of local search. SA is a more stochastic search than OPL, and does not exploit the local structures in the fitness landscape to the same extent. Although it has one possible advantage in the fact that it is better at escaping local optima, thus increasing the chance of finding the global optimum, this is already handled by the genetic algorithm. It would therefore be more advantageous to match the genetic algorithm with a greedier local search algorithm, which moves each individual efficiently towards the nearest local optimum. For this reason, OPL is a better choice of local search algorithm than SA for the memetic algorithm used on this problem, which is also evident in the results.

5.1.4 Investigatory experiments - linear crossover

Local search	**Parameters**
OPL	λ: 2, reevaluation: 100
Control system	**Crossover**
AmpPhaseSym	Linear
Iterations	**Learning**
1, 10, 20	Baldwinian/Lamarckian

Table 5.7: Experiment details, linear crossover

In a genetic algorithm, the recombination operator ensures efficient exploration of the search space and creates diversity, by combining the genomes of two or more parents and thereby creating a new genome which is somewhere in between the parents. This results in a fairly large change of the genome, compared with the result of the mutation operator. An experiment was set up in order to examine the performance of the memetic algorithm with an active recombination operator. Linear crossover was used as the recombination operator, as explained in Subsection 4.2.2. Other than this, the same setup as in the first investigatory experiment was used, see Table 5.7 for details.

Results

As Table 5.8 shows, there is no significant difference between the final results of evolution without learning and evolution with Lamarckian learning in this experiment. The relationship between Lamarckian and Baldwinian learning seems to be the same as in the previous experiments, what has changed is the relationship with the baseline evolution without learning. The fitness plot in Figure 5.9 shows that there is overlap between the results from the configurations with Lamarckian learning and the baseline in the last part of the evolution, after about 600 evaluations. As in the investigatory experiment, there is a faster increase in fitness in evolution without learning than in the configurations with Lamarckian learning, however, they seem to converge to about the same fitness values.

Learning	Iterations	Mean fitness	Median fitness	Best fitness
Baldwin	1	0.4132	0.414	0.5338
	10	0.3681	0.3833	0.4826
	20	0.3521	0.3451	0.4826
Lamarck	1	0.3908	0.4044	0.5013
	10	0.4075	0.427	0.5288
	20	0.417	0.4298	0.5023

Table 5.8: Results with linear crossover activated. The best result in each column is marked with a grey background.

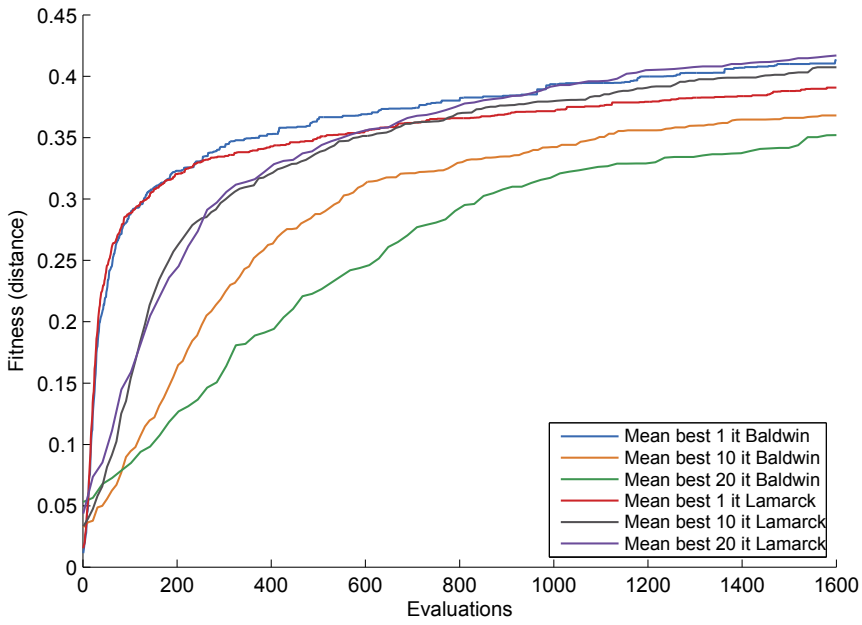


Figure 5.9: Fitness plot showing the mean of the best fitness in each generation, with both Baldwinian and Lamarckian learning.

Analysis

Although an important part of the genetic algorithm, it was not expected that the memetic algorithm would benefit much from including the recombination operator. The results show that this expectation holds up in this case, seeing as the results from the learning configurations are poorer than the ones in the investigatory experiment where crossover was omitted. Performance of evolution without learning seems to be approximately equal in both experiments. Where Lamarckian learning with 20 iterations of local search leads to significantly better final results than the baseline evolution without learning in the investigatory experiment, performance of this configuration has decreased to the same level as the baseline in this experiment. The reason for this can be that the crossover operator causes relatively large changes to the genome of the parents, thereby moving away from the possible local optimum discovered during local search. In other words, the information found during local search is not taken advantage of to the same extent, and the benefits from the local search are blurred out. Since the mutation operator only makes small changes to the genome, it cooperates better with the local search algorithm.

5.1.5 Varying the number of iterations

Local search	**Parameters**
OPL	λ: 2, reevaluation: 100
Control system	**Crossover**
AmpPhaseSym	None
Iterations	**Learning**
1, 20, 50, 100, 200, 400, 1600	Lamarckian

Table 5.9: Experiment details, varying number of iterations

In the first investigatory experiment it was found that Lamarckian learning with 20 iterations of OPL produced significantly better results in the final generation best fitness compared to evolution without learning. With 10 iterations, there was only a tendency towards better performance with the same number of evolutionary runs. It could therefore be interesting to investigate how the number of iterations affects the performance over a larger range of iterations, from the minimum 1 iteration which corresponds to no learning, to the maximum number of iterations, in this case 1600, which corresponds to local search without evolution.

In order to test this, an experiment was set up where Lamarckian learning was tested with seven configurations of different numbers of iterations of local search. Only Lamarckian learning was used, since it performed better than Baldwinian learning, and Baldwinian learning performance also seemed to decrease with increasing number of iterations. Apart from this, the same setup as in the first investigatory experiment was used. See table 5.9 for details. In fact, the results from two of the first sets of evolutionary runs are used in the comparisons, specifically the 1 and 20 iterations configurations, and are restated here in the results table and plots for ease of comparison. The results from local search without evolution are not part of the plots, because they consist of only one set of final fitnesses. This is due to the fact that the evaluations during local search is not stored, so with only one generation, only one final set of fitnesses are obtained. The results from this are given in the results table.

Results

From Table 5.10 and Figure 5.10, it seems that increasing the number of iterations past 20 leads to little change in performance up to a certain limit. The 20, 50 and 100 iteration configurations all converge to about the same average best fitness of just under 0.48, all significantly better than the baseline evolution without learning, using the Wilcoxon rank-sum test with a significance level of 0.01. When the number of iterations is increased to 200, performance decreases slightly, but still seems better than evolution without learning. A Wilcoxon rank-sum test gives a p-value of 0.056 on the final best fitnesses of this set and the baseline, indicating that the difference is not significant with a significance level of 0.01. It is still reasonably

low, so an increased number of runs would probably lead to statistical significance. However, when the number of iterations is increased to 400, the performance decreases considerably, and the final average best fitness between runs is actually significantly closer to the baseline evolution than to the other learning configurations. Table 5.10 also includes the results from the configuration with only local search, that is, 1600 iterations, and shows that this setting has a lower overall performance than all the other configurations. This means that there must be some maximum limit of iterations of local search after which performance will decrease, somewhere between 100 and 400 iterations in this experiment.

Learning	It.	Mean fitness	Median fitness	Best fitness	Final StD
Lamarck	1	0.413	0.4299	0.5432	0.0834
	20	0.4771	0.4711	0.5849	0.0312
	50	0.4767	0.4783	0.5131	0.0206
	100	0.4783	0.4711	0.5583	0.0300
	200	0.4659	0.4671	0.5386	0.0253
	400	0.4382	0.4341	0.4838	0.0293
	1600	0.3564	0.3641	0.4378	0.0540

Table 5.10: Results over a range of different numbers of iterations, from no learning with 1 iteration to only local search with 1600 iterations. Standard deviation between the best fitness from each run in the last generation is included here, since this could not be plotted for the configuration with only local search. The best result in each column is marked with a grey background.

The robustness of learning configurations seems to be about the same when studying the plot over standard deviation in Figure 5.11 and the confidence intervals in Figure 5.12. Standard deviations decrease considerably with these compared to the baseline configuration, which increases more or less steadily. The standard deviation of the configuration with only local search is not plotted, but can be found in Table 5.10.

Analysis

The relatively poor results achieved when only local search was used indicate that OPL alone does not provide enough global exploration to be able to find high fitness areas. With Lamarckian learning of 20 up to 100-200 iterations of OPL, the genetic algorithm provides good exploration, while still benefiting from the local search exploitation of the local structures in the fitness landscape. It seems that at least 20 iterations of OPL with lambda set to two is necessary for the local search to be beneficial, possibly in order to get close to the gradient and move closer to the local optimum. On the other hand, no more than 200 iterations of OPL should be used, since this reduces the global exploration enough to decrease the chance of finding high fitness areas, as can be seen in the results from the configuration with 400 iterations. In other words, there is a balance

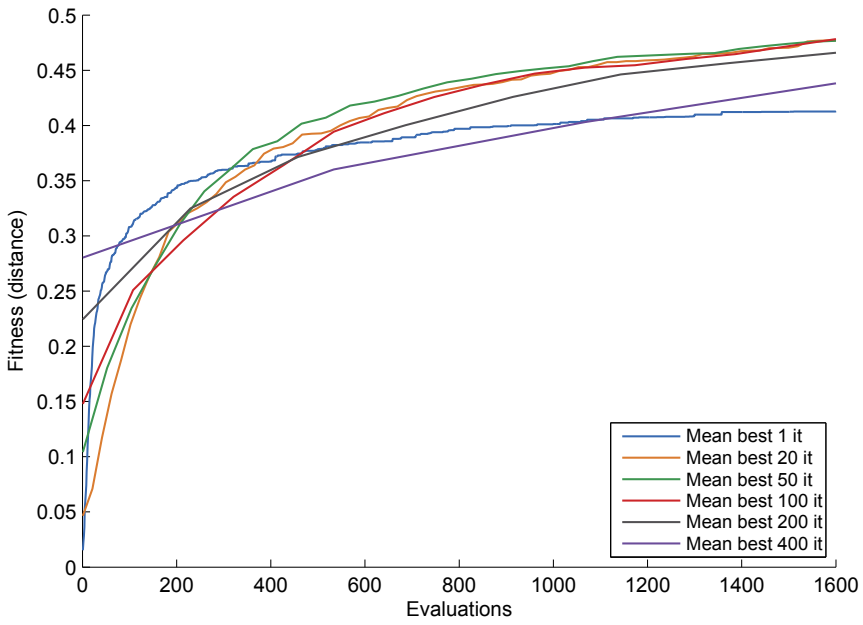


Figure 5.10: Fitness plot showing the mean of the best fitness in each generation. There are 30 runs in each set. Only Lamarckian learning is included. The reason why the initial fitness differs between the different configurations is that when the initial population is evaluated, local search is used as normal, so the initial individuals will all have undergone a local search of the given number of iterations.

between the role of the local search algorithm and the genetic algorithm, that is, a balance between local exploitation and global exploration.

One possible explanation for the decreasing standard deviation in the configurations with 20, 50, 100 and 200 iterations could be that they are all able to find the global optimum or some high quality local optimum. Without learning, there is more randomness involved, and the best solutions found will probably be scattered around high fitness areas, without the ability to move towards an optimum unless a variation operator takes it there by chance. This means that there will be a lot of variation between the best solutions found over different runs, a notorious problem with genetic algorithms. Although the final results of the configuration with 400 iterations of local search were poorer than the other learning configurations, the standard deviation is about the same. This can again be because there is less randomness involved, resulting in individuals moving towards the same local optimum, if not the global optimum.

What was discovered during the experiment process, but not documented here, was that evolution management operators turned out to be more time demanding than the local search management. This meant that the total computation time of evolution decreased with an increasing number of iterations, because of the equivalent decrease in generations. Because the exact time would depend on the hardware and hardware utilization, the numbers are omitted here. Still, it showed that it could be ad-

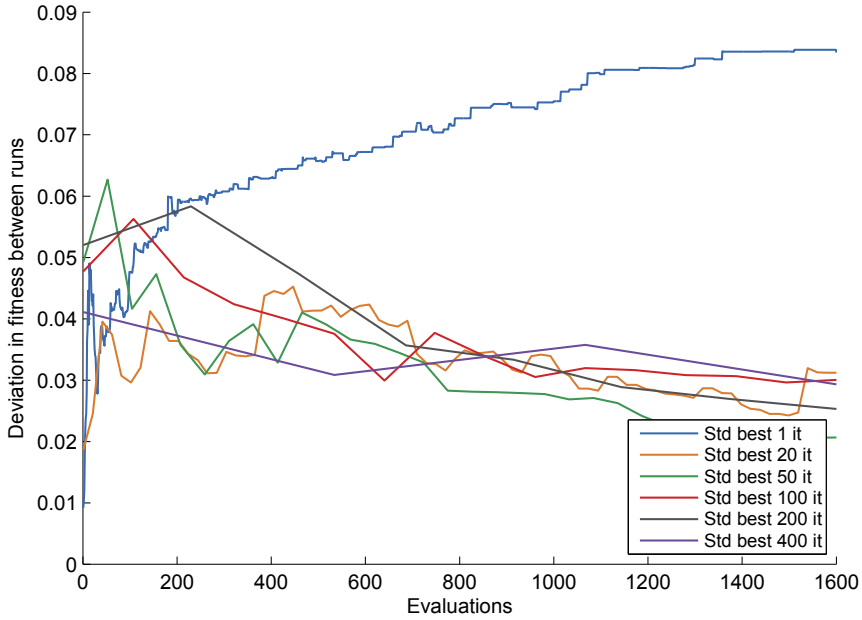


Figure 5.11: Plot of standard deviation of the best fitnesses in each generation.

vantageous to use an as high number of iterations as possible before performance is reduced, in order to keep the total computation time relatively low.

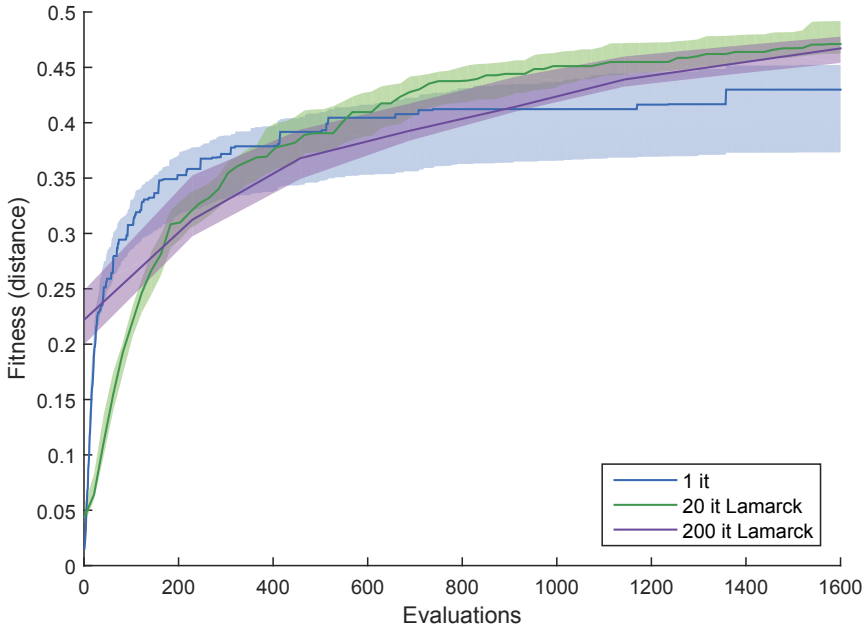


Figure 5.12: Plot with confidence intervals and median, with a 99% confidence level.

5.1.6 Experiment with changing environment

Local search	**Parameters**
OPL	λ: 2, reevaluation: 100
Control system	**Crossover**
AmpPhaseSym	None
Iterations	**Learning**
1, 10, 20	Baldwinian/Lamarckian

Table 5.11: Experiment details, changing environment

This experiment was set up in order to test the robustness to sudden changes in the environment of the memetic algorithm, since there is a possibility that lifetime learning will cause faster adaptation in these situations. See Subsection 4.3.2 for further details on why and how this was implemented. Three different environments were used, the normal one used in the other experiments, one which includes obstacles, and one with a low friction ground. These were introduced during evolution in the order listed here, and at times so that each environment gets the same amount of generations over which it is used. Apart from the changing environment, the same setup as in the first investigatory experiment was used, see Table 5.11.

Results

Since each environment only gets a third of the total evolution time, which is kept the same as the previous experiments, the final fitness values with each environment is not as interesting here. Instead, adaptability to the newly introduced environment is examined. The fitness plot in Figure 5.13 shows that the introduction of obstacles results in a significant drop in fitness, which was to be expected since the obstacles will obstruct the gait of the robots. When the low friction environment then is introduced, the fitness increases again, mainly because there are no obstacles in this environment. It is also evident that Lamarckian learning with 20 iterations of local search immediately adapts better overall to the obstacle environment than the other configurations, with a significantly better performance over the first generations.

After the second switch of environment, that is, when low friction is introduced, there is no significant difference between the baseline and Lamarckian learning, while the Baldwinian learning configurations have relatively low performance compared to the rest. However, there is a tendency towards the baseline performing better than the Lamarckian learning configurations, and also that Lamarckian learning with 10 iterations performs slightly better than the 20 iterations configuration. These differences are not certain to be the same over a larger set of evolutionary runs, as can be seen in Figure 5.15. The confidence intervals in this plot are largely overlapping in the third section where the low friction environment is used.

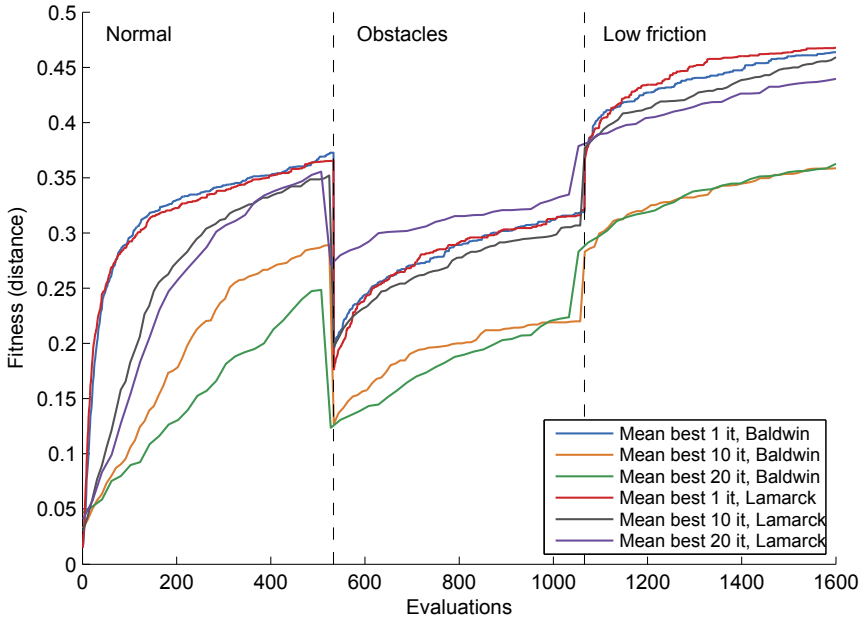


Figure 5.13: Fitness plot showing the mean of the best fitness in each generation. The dashed line shows where a new environment is introduced. Because there are less generations with learning, switching is done at slightly different times with different number of iterations of local search. This is because switching must be done at the end of a generation. The results should not be affected much by this.

Analysis

Although most of the differences in the plot in Figure 5.13 are not significant, there are certain tendencies that are worth commenting. It seems that learning individuals more easily adapt to the sudden change in environment when obstacles are added, however, the baseline evolution without lifetime learning has a faster increase in fitness after this has happened. A possible explanation to this could be that learning individuals quickly discover relatively good solutions in the fitness landscape around each individual, thereby keeping the overall fitness relatively high compared to the baseline, excluding the Baldwinian learning configurations. However, because the entire search space will have changed when the obstacles were added, these local optima may not be optima any more, which means that more global exploration is needed to be able to move towards better solutions. Since evolution without learning is able to do more global exploration over fewer evaluations than with learning, it will move the individuals towards better solutions faster. This explains why the baseline almost catches up with Lamarckian learning with 20 iterations in the obstacle section in Figure 5.13, and that Lamarckian learning with 10 iterations performs at about the same level as the baseline. If the obstacle environment was used over a longer period of time, it is possible that the baseline would have converged before the Lamarckian learning configurations, but this is

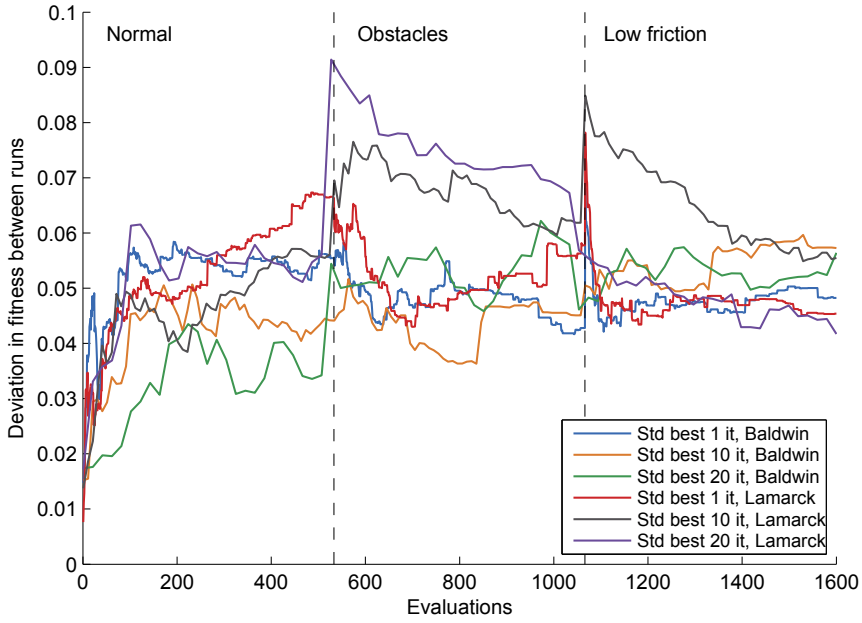


Figure 5.14: Plot of standard deviation of the best fitnesses in each generation.

impossible to say looking at this plot only.

When the low friction environment is introduced, a different situation arises. Firstly, the overall fitness increases suddenly, because the obstacles are removed. Secondly, the baseline configurations both have an almost logarithmic increasing fitness in the low friction period, similar to the two previous periods, and they seem to outperform the learning configurations. The differences here are not significant, as Figure 5.15 indicates, but the tendencies are there. The reason why Lamarckian learning with 20 iterations of OPL suddenly performs worse than the baseline is not clear, but a possible explanation could be that good solutions with an environment filled with obstacles are badly transferred to a low friction environment. In order to avoid the obstacles, it is possible that the individuals develop a more jumping sort of gait, which will exhibit a lot of force towards the ground. When the obstacles are removed and the friction of the ground is reduced, this kind of gait would probably cause a lot more slipping than one which does not jump to the same extent. In other words, good solutions in the obstacle environment might correspond to especially bad solutions in the low friction environment, making global exploration important in order to move away from local optima. This means that this particular change in environment is unfortunate for learning individuals, as individuals might start in low fitness areas which the local search is not powerful enough to get out of. If the evolution was continued for a longer time in the low friction environment, it is possible that there would be a similar trend as in the first investigatory experiment, were the configurations with Lamarckian learning eventually catch up with and surpass the baseline.

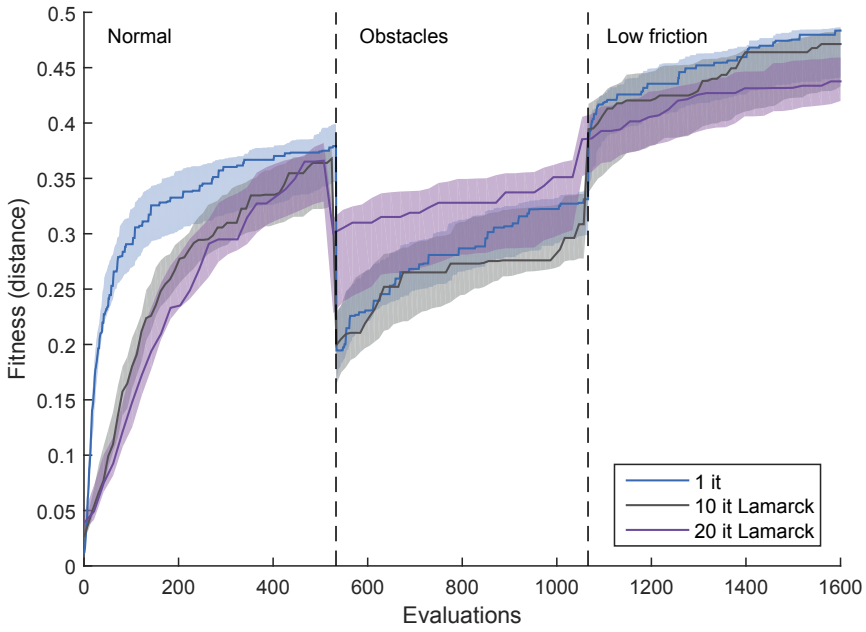


Figure 5.15: Plot with confidence intervals and median, with a 99% confidence level.

How successful the local search is when the environment changes appears to vary a lot between runs, as Figures 5.14 and 5.15 illustrates. The standard deviation between runs in the Lamarckian learning configurations increases significantly when obstacles are introduced. A possible explanation to this could be that when the fitness landscape changes, there is a difference in how close good solutions in the previous environment are to good solutions in the new one. Some individuals will be closer to new local optima than others, creating a large span between the runs and how successful the local search is.

5.2 Evaluating control system performance

Although the results retrieved from evolution in the simulator are interesting in themselves, the final stage of development is the transferral of the evolved control systems to the physical robot. In order to test the performance of a control system on the physical robot, a set of evaluations were done, both in hardware and in simulation for comparison. To get an idea of how a control system performs over time, 30 evaluations were done with each control system, where one evaluation corresponds to the mean displacement over 4 time periods.

Evaluations on the physical robot was done using the motion capture equipment described in section 3.3, while the same simulator as was used in the evolutionary experiments was used for evaluation in simulation. Only the results from the first investigatory experiments were used, see Subsection 5.1.1, since these included both Baldwinian and Lamarckian learning, and the control system used there seemed better transferable to the physical robot than the AmpOffPhase system. The baseline configurations of evolution without learning, corresponding to one iteration of Baldwinian or Lamarckian learning, are the same, and are therefore merged here. Since evolution of control systems most often is a design problem, the individuals with the best final results in each configuration from this experiment were evaluated. To get a better overview of the overall performance, the median individuals in each set were also evaluated.

5.2.1 Results

Tables 5.12 and 5.13 show that there is a difference between the mean evaluations and the original fitness value, both in simulation and in hardware. On the physical robot, the mean evaluation is considerably lower in all cases, while the evaluations in simulation are either lower or about the same.

A reality gap is apparent in all configurations, and is especially clear with the median individuals. This can be seen in Figures 5.16 and 5.17 as well as in the mentioned tables. The configuration with 20 iterations of Baldwinian learning from the best individuals is a special case where the mean evaluations are actually better on the physical robot, but this is probably because of special features in the gait, which will be discussed in the analysis. The size of the reality gap differs between the same configurations in the best and median individuals, so concluding on which configuration gives the largest reality gap is not easy. In the best individuals, the largest difference lies with the baseline configuration without learning, with a 38% performance loss. In the median individuals, the configuration with 20 iterations of Baldwinian learning has the largest performance loss, with 84%.

The box plots in Figures 5.16 and 5.17 show that some individuals have a large spread in the evaluations, such as the baseline configuration in Figure 5.16. The reason for this is that these particular robots tend to get obstructed by their own body, typically by getting one leg stuck under

another. During evaluation, the robot alternates between moving freely and being more or less obstructed by itself, resulting in a wide range of evaluation results. Ideally, all evaluations should be about the same, as this means that the gait is robust. When the spread of evaluations differ between the evaluations done in simulation and hardware, this is part of the reality gap for that robot.

Learning	It.	Fitness	Mean simulation	Mean hardware	Ratio
None	1	0.5432	0.3553	0.2206	0.6209
Baldwin	10	0.4923	0.3531	0.3034	0.8592
	20	0.4515	0.2336	0.2885	1.2350
Lamarck	10	0.5323	0.4981	0.3101	0.6226
	20	0.5849	0.2497	0.2047	0.8198

Table 5.12: Best individuals. Table showing results of evaluation both in simulation and on the physical robot of the best individuals from the investigatory experiment with the AmpPhaseSym control system. All values are in m/s, except the ratio. The original fitness value that resulted from evolution is given in the *fitness* column. The *ratio* column shows the ratio between the mean evaluations in hardware and simulation (mean hardware/mean simulation). The best result in each column is marked with a grey background.

Learning	It.	Fitness	Mean simulation	Mean hardware	Ratio
None	1	0.4299	0.3993	0.1576	0.3947
Baldwin	10	0.3729	0.1004	0.0329	0.3277
	20	0.4105	0.1580	0.0257	0.1627
Lamarck	10	0.451	0.4653	0.1565	0.3363
	20	0.4711	0.1396	0.0893	0.6397

Table 5.13: Median individuals. Table showing results of evaluation both in simulation and on the physical robot of the median individuals from the investigatory experiment with the AmpPhaseSym control system. The column descriptors are the same as in Table 5.12.

5.2.2 Analysis

The differing size of the gap between the final fitness value after evolution and the mean evaluation in simulation can be the result of many factors. One such factor could be that there is a certain level of noise in the local search, which can affect the fitness value. A clear indication that this is the case is that the configurations with 20 iterations of local search all have relatively low mean performance after evaluation in simulation, as compared with the original fitness. Still, most of the configurations with 10 iterations of local search have a fairly high performance after evaluation, indicating that noise is more apparent with a larger number of iterations.

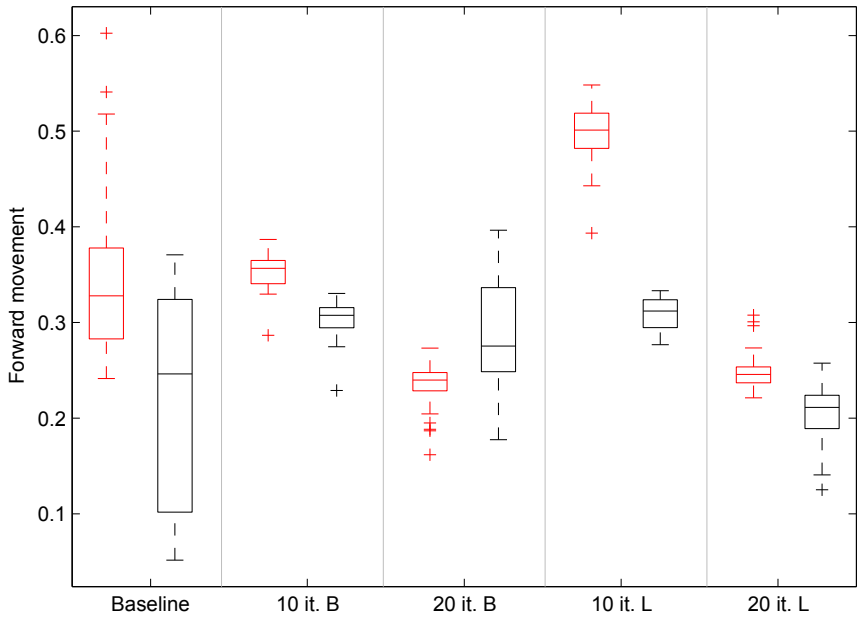


Figure 5.16: Box plot showing the results from evaluation of the best individuals in simulation and on the physical robot. B and L stand for Baldwinian and Lamarckian learning, respectively. The red boxes represent the evaluations done in the simulator, while the black boxes represent the evaluations done on the physical robot.

Because of the low number of samples used here, this situation is not necessarily the case with the rest of the distribution.

The noise in the local search can be the result of inaccuracies in the evaluation function. Measuring forward movement is not trivial; the robot is allowed to turn while moving, and will therefore be changing the direction of what is considered as *forwards* during evaluation. The evaluation function used in these experiments decides the direction in which displacement is measured as the direction the robot is headed at the start of the period. A problem with this method is that if the robot is swaying back and forth while moving, the direction it is headed at sample time is not necessarily the direction of its forward movement. This will lead to inaccuracies in the measurements, the severity of which will vary depending on the particular gait of the robot. When local search is applied during evolution, a large number of evaluations are done during search, possibly increasing the chance of obtaining inaccurate results. This can explain why the evaluation of the individuals evolved with 20 iterations of Lamarckian learning is considerably poorer than the fitness value obtained during evolution in both the median and best individuals. The low evaluated performance of the individuals from this configuration does not necessarily mean that Lamarckian learning fails to find the true good solutions, but that the fitness values stored in the final individuals are inaccurate and do not always correctly represent the performance of the control systems. Thus, in order to reduce noise in the local search and

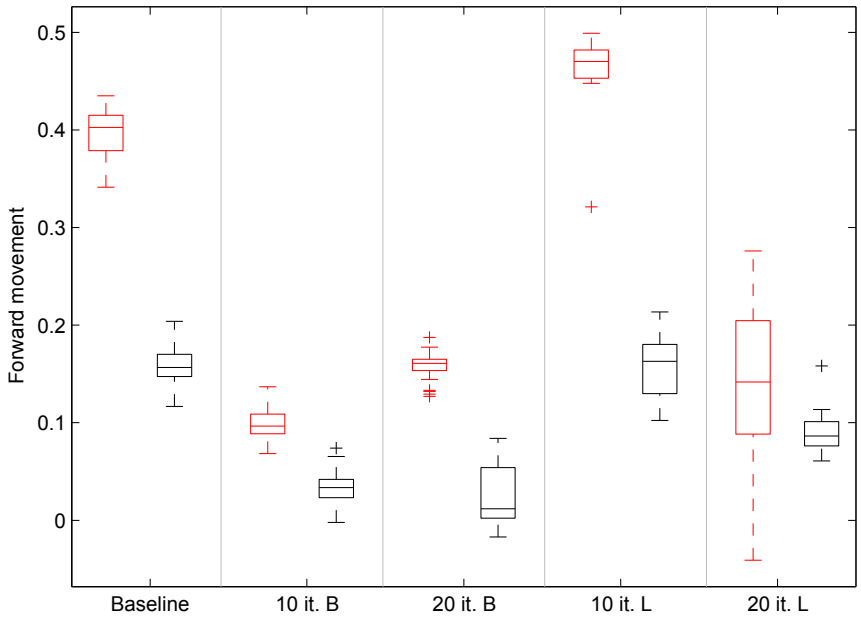


Figure 5.17: Box plot showing the results from evaluation of the median individuals in simulation and on the physical robot. The same layout as in Figure 5.16 is used.

thereby decrease the chance of this happening, a more robust evaluation function should be used.

Another factor could be that evaluation in evolution is only done over 8 time periods, while evaluation of the final solution is done over 120 time periods. Since the performance of a solution will sometimes alter over time, e.g. the robot may for some reason get obstructed by itself after a while, one evaluation of 8 time periods is probably not enough to describe the overall performance of the control system.

When studying the evaluations done in simulation and on the physical robot, it is clear that there is a reality gap of varying size present in most configurations. However, for the configuration with 20 iterations of Baldwinian learning from the best individuals, the mean evaluation in hardware is actually slightly better than in simulation, which is a rare situation in evolutionary robotics. Still, this positive difference is part of the reality gap for this particular robot, because of a specific feature in its gait, this being that the robot moves along a circular path. In hardware, it seems that the robot moves in larger circles than in simulation, and because robot displacement is measured in the direction the robot was headed at the beginning of the evaluation, a larger circular path will lead to a larger measured displacement. The reason why the physical robot moves along a larger circle is probably because there is a difference in friction between the ground in the simulator and the lab floor. When the robot turns, it will therefore slip more in the real world than in the simulator, resulting in a less sharp turn in the real world. The absolute speed of the robot is probably about the same in simulation and hardware. Difference in friction

is the main contributor to the reality gap in these experiments. While leading to larger measured speed for the robot evolved with 20 iterations of Baldwinian learning, it generally leads to poorer performance in reality for the rest of the robots, because they do not get the same benefit in the real world from applying a lot of force on the ground to create large movements. If the floor friction parameter is reduced in the simulator, it is possible that the reality gap is reduced.

As of the difference in reality gap between Baldwinian and Lamarckian learning, the numbers in Tables 5.12 and 5.13 are inconclusive. Within the best individuals, Baldwinian learning has the least loss of performance of the two, while the opposite is the case with the median individuals. However, looking at the box plots in Figures 5.16 and 5.17, there seems to be a tendency towards a larger reality gap in the configuration with 10 iterations of Lamarckian learning, and that these individuals have become overly adapted to the simulator. In order to confirm or reject this hypothesis, a larger amount of gaits from each configuration should be evaluated, thereby increasing the robustness of the results.

5.3 Running local search on the evolved control systems

An interesting method of decreasing the reality gap is to apply learning to the evolved solutions on the physical robot, thereby allowing them to adapt to the real world. Since there is a possibility that solutions evolved with Baldwinian learning has a larger potential to learn, these may have an advantage when this method is used. To test this hypothesis, two experiments were set up to examine the learning abilities of a small selection of solutions from the evolutionary experiments in section 5.1. The same individuals as in the previous section were used, that is, the best and median results from the first investigatory evolutionary experiment, see Subsection 5.1.1. The first experiment focuses on testing learning abilities in simulation, while the second examines learning on the physical robot, and to what degree this reduces the reality gap in each configuration. In both cases, OPL was used as local search algorithm, with lambda set to 2 and no reevaluations. In each learning run, 20 iterations of OPL were performed, followed by 30 evaluations of the best gait obtained during the local search.

It should be noted that the terms *local search* and *learning* are used interchangeably in this section, and that *learning* in this context has the meaning of applying local search to an individual *after* evolution. If learning *during* evolution is mentioned, this is referred to as *lifetime learning* or either Baldwinian or Lamarckian learning. Furthermore, the baseline evolution without learning is again represented by the configurations with one iteration of Baldwinian or Lamarckian learning, which were left separate to keep more of the data.

5.3.1 In simulation

Since the local search algorithm is to some extent stochastic, learning was applied to each individual from every configuration 10 times, in order to reduce noise in differing final results and to test the robustness of the local search. In addition, evaluation time was increased to 8 periods per evaluation, from 4 in the previous section. This was done to reduce the noise in the local search, seeing as taking the mean over more samples reduces the contributions from outliers, which is what is done at the end of an evaluation. Because of this, each initial individual was reevaluated with the same number of evaluation periods, to make sure comparison between lifetime learning and post-learning was fair.

Results

Tables 5.14 and 5.15 contain statistical results from learning on the best and median individuals, respectively. The tables are organized as follows: The *start fitness* row contains the starting fitness of the individual before local search, i.e. the first evaluation in the local search. *Mean final f.* shows the mean of the final fitnesses over all runs of local search, while *improvement* is the difference between the two previous rows. These three rows only use

the direct results obtained during learning, involving only one evaluation, while the next three use the results from after the final gaits have been evaluated over 30 evaluations. *Median impr.* contains the improvement between the mean evaluation of the initial gait, and the median of the mean evaluations of the final gait after local search. The median was chosen because it indicates the average performance of the local search. *Median ratio* shows the improvement ratio instead of the raw improvement value, where a low number indicates that learning has been successful, and a value larger than 1 means that the solution before learning is still better. The *std. dev. ls* row lists the standard deviation between the mean evaluations of the final results after local search.

The results in the two tables deviate from each other with respect to which configuration has the largest improvement. However, on the immediate results from the local search, the configurations with 10 and 20 iterations of Baldwinian learning both have relatively large improvement. Then again, so has the 20 iterations Lamarckian configuration. Only the configurations with 20 iterations of Baldwinian and Lamarckian learning have a positive median improvement on both the best and median individuals.

Figures 5.18 and 5.19 show box plots of the evaluations of the performance of each individual before and after local search was applied. The red boxes are the evaluations of the initial solution, while the black boxes represent the evaluations of the median results after local search. It should be noted that some of the gaits tend to tangle the robot slightly at times, resulting in a range of varying evaluations, which explains why there is a difference in spread.

Learning	Baldwin			Lamarck		
Iterations	1	10	20	1	10	20
Start fitness	0.5113	0.3083	0.1876	0.5432	0.4678	0.2987
Mean final f.	0.5130	0.3873	0.3285	0.5435	0.4777	0.3681
Improvement	0.0018	0.0790	0.1409	0.0002	0.0099	0.0694
Median impr.	-0.0257	-0.0227	0.0397	0.0044	-0.0025	0.0792
Median ratio	1.1216	1.0681	0.8542	0.9868	1.0049	0.7578
Std. dev. ls	0.1572	0.0706	0.1515	0.2264	0.0730	0.2385

Table 5.14: Table showing the results from the local search on the best individuals. The row descriptors are described in the beginning of the current section. The best result in each row is marked with a grey background.

Analysis

When comparing the two Tables 5.14 and 5.15, it becomes clear that the results of learning differ quite a lot between the two individuals within each configuration. Thus, without more samples from each configuration,

Learning	Baldwin			Lamarck		
Iterations	1	10	20	1	10	20
Start fitness	0.3936	0.1218	0.1573	0.4124	0.3862	0.1857
Mean final f.	0.3949	0.3323	0.2986	0.4218	0.4354	0.3452
Improvement	0.0013	0.2105	0.1412	0.0094	0.0492	0.1595
Median impr.	0.0008	0.1961	0.1271	-0.0153	-0.1808	0.1095
Median ratio	0.9979	0.3323	0.5580	1.0398	1.6294	0.5553
Std. dev. ls	0.0018	0.0834	0.1345	0.1272	0.1474	0.1834

Table 5.15: Table showing the results from the local search on the median individuals. The row descriptors are the same as in Table 5.14

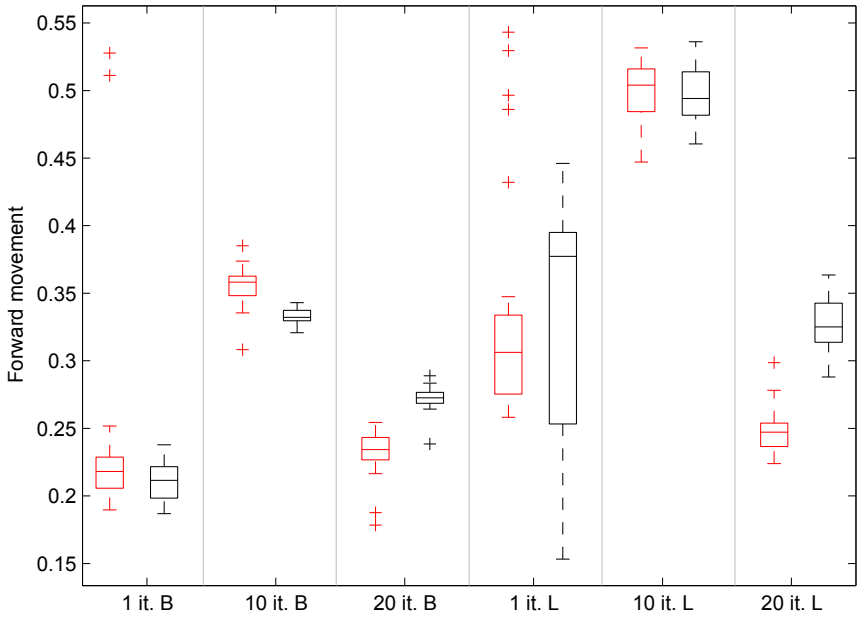


Figure 5.18: Box plot showing the results from evaluation of the best individuals before and after local search, in simulation.

that is, more final solutions from each set of evolutionary runs, it is not easy to conclude on which configuration is best suited for this kind of learning. Every gait has its own properties, which can affect the effects of learning on each particular gait. However, there are a few tendencies that are interesting, and that should be looked further into in future work. The box plot in Figure 5.19 and Table 5.15 show that the largest increase in performance after learning happens with the individuals that were evolved with Baldwinian learning. Since these had a relatively low initial performance, it is possible that they had the largest potential for improvement because of this, which is supported by the results from the configuration with 20 iterations of Lamarckian lifetime learning, which also has a relatively large improvement. This should be investigated further, especially since it is not backed up by the results from the best individuals, were such a tendency is not at all clear.

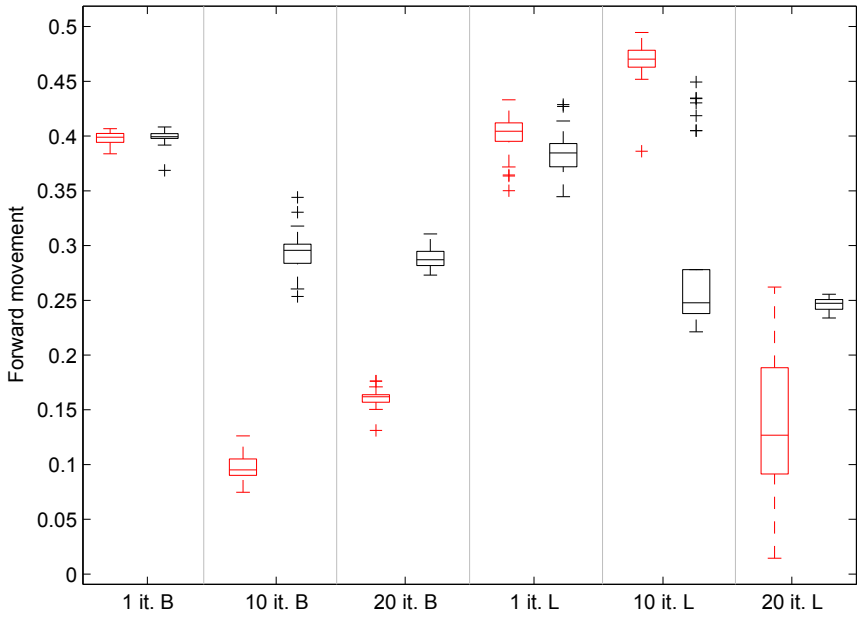


Figure 5.19: Box plot showing the results from evaluation of the median individuals before and after local search, in simulation.

One thing that is consistent in all the results, is that the solutions from the baseline evolution without learning, represented as one iteration of either Baldwinian or Lamarckian learning, all have very little improvement after local search. This is clear both in the direct results obtained during local search, and in the subsequent evaluations. When studying the box plots, and particularly the plot with the median individuals in Figure 5.19, it is clear that both baselines have relatively good performance with the initial solution, so there is a possibility that they are already at or close to a local optimum from which it is difficult to move away.

The standard deviations listed in the tables show no trend towards any configurations being more robust than others, which is probably because of the stochastic properties of OPL. It is possible that this is also affected by instabilities in the sampling of forward movement in the evaluations, which can also explain why the best solution after local search sometimes evaluates to worse performance than before local search. The results of the evaluations of the best solutions found during learning further indicates that there is noise in the local search, since some of the assumed improved solutions have a poorer mean evaluation than the starting solution. Thus, the same problem with the evaluations during local search as was discussed in the analysis in the previous section is present.

5.3.2 On physical robot

In order to test the effects of learning on the physical robot, and then examine the extent to which the reality gap is reduced, the same type of learning experiment as was done in simulation was performed on the physical robot, with a few changes. Firstly, since testing on the physical robot is very time demanding, only one local search was performed on each individual. This is clearly not enough to test the learning abilities of each configuration, but should at least give some indication of the effects. Secondly, the evaluation time was reduced to 4 periods per evaluation. This is safer to do in hardware than in simulation, because the overflow problem is handled to some extent here by the first sample in an evaluation being discarded in hardware.

Results

The box plots in Figures 5.20 and 5.21 show the evaluations of the initial solutions and of the best solution found after local search, pictured in red and black boxes, respectively. In addition, the evaluation in simulation from section 5.2 is included in blue, to better visualize the possible recovery from a reality gap. As in the previous experiment in simulation, there are inconsistencies between the results from the best and the median experiments. Apart from a few exceptions, the same learning tendencies are present in hardware. In the median individuals, the individuals evolved with Baldwinian learning have the larger degree of improvement, while this is not the case in the best individuals, apart from the direct improvement during learning, as can be seen in Table 5.16. Table 5.17 clearly shows that the individual evolved with 20 iterations of Baldwinian learning has the largest improvement, followed by the 10 iterations Baldwinian learning individual.

Learning	Baldwin			Lamarck		
Iterations	1	10	20	1	10	20
Start fitness	0.3394	0.2989	0.2744	0.3075	0.3193	0.1853
Mean final f.	0.4204	0.3012	0.3749	0.3324	0.3741	0.2335
Improvement	0.0810	0.0023	0.1004	0.0250	0.0547	0.0482
Mean impr.	0.1141	-0.0668	-0.0142	0.0191	0.0301	0.0480
Mean ratio	0.7235	1.2824	1.0518	0.9201	0.9115	0.8099

Table 5.16: Table showing the results from the local search on the best individuals. The row descriptors are the same as in Table 5.14, except the last two rows which lists the mean improvement directly, since there is only one instance of learning for each individual. The best result in each row is marked with a grey background.

Learning	Baldwin			Lamarck		
Iterations	1	10	20	1	10	20
Start fitness	0.1987	0.0187	0.0503	0.1556	0.2055	0.1712
Mean final f.	0.2558	0.1668	0.2128	0.2204	0.2263	0.2415
Improvement	0.0572	0.1480	0.1625	0.0649	0.0208	0.0702
Mean impr.	0.0374	0.1235	0.2014	0.0516	0.0150	0.1229
Mean ratio	0.8439	0.2103	0.1131	0.7533	0.9127	0.4207

Table 5.17: Table showing the results from the local search on the median individuals. The row descriptors are the same as in Table 5.16.

Analysis

Although the low amount of data makes it difficult to come to a clear conclusion, there are a few tendencies in the results that are interesting and should be pursued further. The box plot of the evaluations of the median individuals in Figure 5.21 shows that the individuals evolved with Baldwinian learning both eliminate the reality gap after local search is applied, in fact, the evaluations of the new solutions are even better than the evaluations in simulation. This seems very promising for the initial hypothesis that solutions evolved with Baldwinian learning store a potential to learn. In this particular case, also the individual evolved with 20 iterations of Lamarckian learning benefits largely from learning, but it seems this individual is affected by local search noise during evolution, which makes the validity of these results questionable. This can obviously also be a problem with the individuals evolved with Baldwinian learning, and should be investigated further with a more robust local search evaluation function.

The results from the best individuals are inconsistent with the results from the median individuals, and show no indication of a learning potential in the Baldwinian learning individuals. There is a possibility that individuals that fail to achieve a relatively high fitness during evolution benefit more from learning being applied to the final solutions after evolution than the best individuals. Still, this is just a hypothesis, which can only be confirmed or rejected with evaluation of more samples. More gaits from each configuration should be tested, and a larger set of learning runs performed on each of these, thereby increasing the robustness of the results. In addition, a larger number of iterations of local search after evolution should also be tested, as this will probably result in a better recovery and reduction of the reality gap. This is a very time demanding challenge to solve, which is why this has not been done in these experiments.

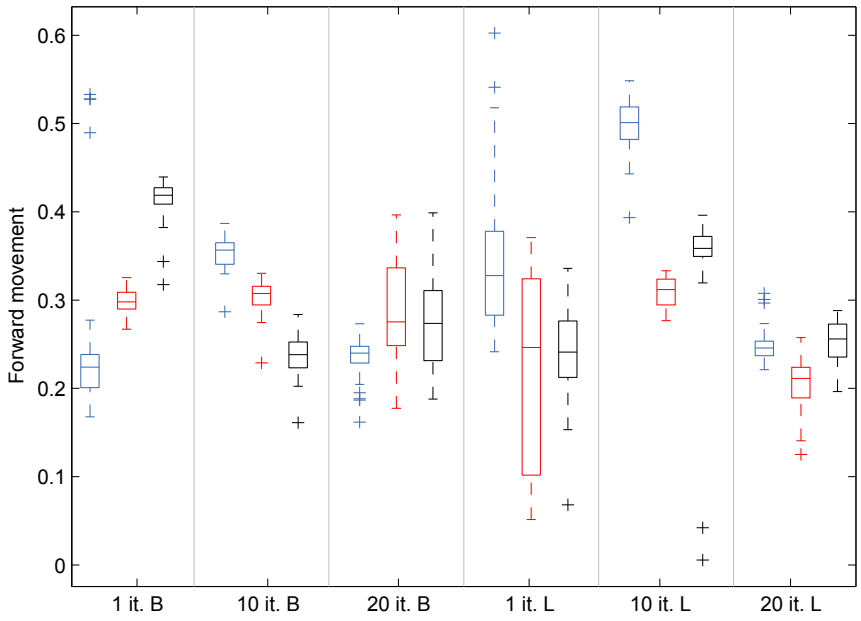


Figure 5.20: Box plot showing the results from evaluation of the best individuals before and after local search, on the physical robot. The red boxes are evaluations in hardware of the gait before learning, while the black boxes are evaluations in hardware of the final gait after learning has been applied. For comparison, the simulated evaluations are included in blue.

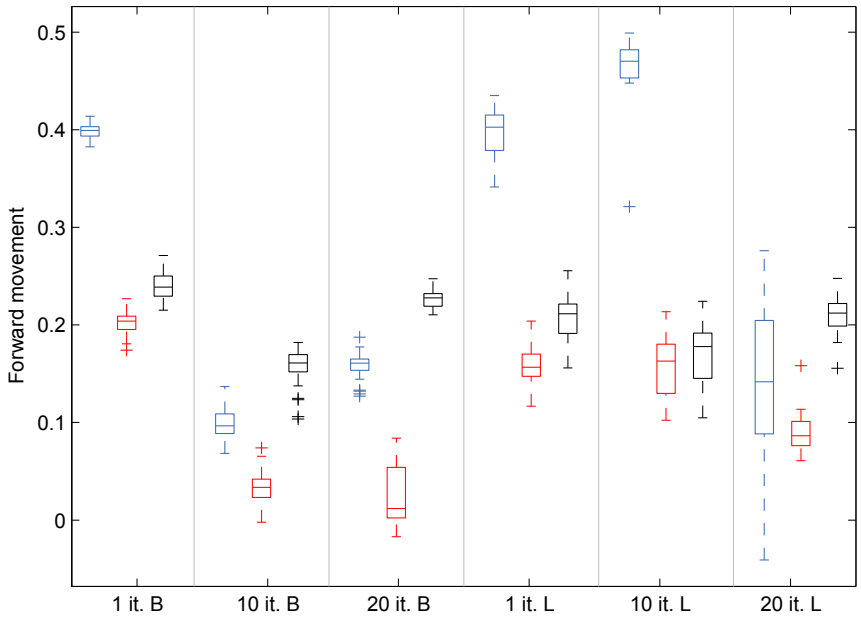


Figure 5.21: Box plot showing the results from evaluation of the median individuals before and after local search, on the physical robot. The box colours are the same as in Figure 5.20

Chapter 6

Discussion

6.1 General discussion

The results from Section 5.1 show that the memetic algorithm with Lamarckian learning with OPL can outperform the standard genetic algorithm (NSGA-II) on the final fitness values obtained from evolution. This is the case when the number of iterations of local search is between 20 and 200, where the increase in final fitness is statistically significant, at least for the control system used in the first investigatory experiment. For the second control system, more testing must be done in order to show significant improvement, but there is a definite trend towards the same situation. From the results of evolution with Baldwinian learning, it seems that the cost of learning is larger than the benefit of a learning potential, at least in the direct results after and during evolution.

All the fitness plots from the evolutionary experiments show that the baseline evolution without learning has a faster increase in fitness at the start of the evolution, as well as a faster convergence. This can be explained by the fact that evolution without learning in these experiments does considerably more global exploration than with learning, thus locating an area of high fitness relatively fast. Learning reduces the number of generations because of the cost of local search, that is, evaluations done over all iterations in the search, thereby slowing down the global exploration. This also explains why evolution with learning has a slower convergence rate with respect to the number of evaluations performed.

When the final solutions were evaluated in Section 5.2, one of the things that became apparent was that the final fitness value was not always consistent with the evaluations of the final solutions in simulation, and especially so for the results from the evolution with 20 iterations of Lamarckian learning. This indicates that there is a certain level of noise in the local search setup, which is possibly increased with increasing number of iterations of local search. One source of noise was known beforehand, i.e. that there is a slight overflow from one iteration of local search to the next, because the robot is not reset between iterations. In other words, the robot will be moving in a certain speed at the beginning of an iteration, excluding the first, depending on the performance of the previous

iteration. However, since the final evaluation of a control system in each iteration is set to the mean displacement over eight time periods, this should not influence the results to a great extent. Another possible source of noise is instabilities in the evaluation function. Measuring forward movement is not a trivial task when the robot is allowed to move freely, since the direction of movement will change continuously with the position of the robot. The evaluation function used in these experiments measured forward movement in the direction the robot is headed at the beginning of the evaluation period, which may not be consistent with the true direction of movement, depending on the specific properties of the gait.

The results from Section 5.2 also show that there is a considerable reality gap in all the configurations tested. Two sets of individuals were tested in each configuration, but because the results were conflicting on which configuration had the largest reality gap, no conclusion can be drawn on whether Lamarckian or Baldwinian learning leads to the smallest gap. The reality gap is mainly the result of difference in friction in the simulator and the real world, both between the robot and the ground and between one robot part and another. It seems that friction between the robot and the ground is larger in the simulator than in the real world, causing the physical robot to slip more than the simulated one. Since the robot is of a morphology which is prone to collide in itself while walking, mainly by a front and hind leg bumping into each other, friction between body parts is also a factor. The robot tends to obstruct itself more in the real world, indicating that the friction between body parts is larger in the real world. This could probably be improved by tuning of the friction parameter in the simulator, by lowering the ground friction and possibly increasing the robot body friction.

In Section 5.3, local search was applied to a small set of solutions from the evolutionary experiments. Although the results were too inconsistent to be able to conclude on which configurations had the greatest learning potential, there was a certain trend towards Baldwinian learning giving the largest potential in some of the evolved individuals. Since the results from evolution were to some extent affected by noise in the local search, and the number of samples from each configuration was relatively low, the hypothesis that individuals that have been evolved with Baldwinian learning can store a learning potential cannot be confirmed or rejected for the memetic algorithm used in these experiments.

6.2 Conclusion

A memetic algorithm for evolution of parametric control system parameters for robots of a fixed morphology was presented in this thesis. The algorithm consists of a merger between the multi-objective genetic algorithm NSGA-II and a local search algorithm, and was implemented with two options of lifetime learning, Lamarckian and Baldwinian. The algorithm was then tested in simulation over a series of different configurations, e.g. with different numbers of iterations of local search, with the two different local

search algorithms SA and OPL, and with two different control systems.

The results show that using a Lamarckian memetic algorithm for evolution of robot control system parameters can be advantageous compared with a normal genetic algorithm. When an adequate number of iterations was used, Lamarckian learning had a positive effect on two different control systems. Because there is a balance between the weight of the local search and the number of generations, a more powerful local search will have less global exploration, which means that there is a limit of how many iterations of local search can be performed before overall fitness quality is reduced. Since only the fitness value and not the parameters are stored in the individual after local search in evolution with Baldwinian learning, this performed significantly worse than both evolution with Lamarckian learning and evolution without learning.

After the base evolutionary experiments were performed, a few control systems were evaluated more thoroughly in the simulator and on the physical robot. Because of inconclusive results, partly due to a low number of samples being evaluated from each generation, no conclusion can be made on whether Lamarckian and Baldwinian learning leads to larger or smaller reality gap compared to standard evolution without learning. To be able to do this, the robustness of the results should be increased by evaluation of a larger selection of solutions from each configuration.

One of the major challenges in this thesis was the presence of noise in the evaluations of the local search, both during evolution in the simulator and when running local search on the physical robot. Noise is a common problem in learning and evaluations of real life robotics, creating a need for further research on methods that take this into account specifically. One way of dealing with noise is to run a large set of equivalent evaluations followed by filtering of the results, but on a real world situated robot this process is in most cases very time demanding, resulting in there being a trade-off between time and the number of evaluations.

6.3 Future work

A few problems were discovered during the thesis work which should be addressed in future work. Firstly, since the local search evaluations seemed to be affected by inaccuracies in the measurements, it might be beneficial to generate more data using a local search setup with less noise, as this would provide more security in the results. Secondly, more testing should be done on learning as a recovery mechanism for reality gap issues, by more rigorous testing on a larger quantity of evolved individuals.

This thesis was only concerned with evolving control systems for a robot with fixed morphology. The logical next step is to extend the algorithm to include coevolution of morphology and control. This is a far more demanding challenge, since this includes encoding the phenotype morphology back into the genotype if Lamarckian learning is to be used, a procedure that is not trivial. When morphology is included in the evolution, the results of lifetime learning may differ from what was found in this

thesis, in the sense of which configuration is most successful.

Another interesting addition could be to introduce local search at a later point in evolution, thus benefiting from fast global exploration in the first part of evolution, followed by exploitation of local structures in the fitness landscape when lifetime learning is included. The fitness plots from the evolutionary experiments show that evolution without learning has a faster increase in fitness at the beginning of evolution, while evolution with Lamarckian learning converges to a higher final fitness. A combination of this could possibly lead to a reduction in the number of evaluations necessary for reaching a good solution. This would be a very artificial evolutionary method, but could possibly lead to faster discovery of high fitness areas.

Bibliography

[1] J Mark Baldwin. 'A new factor in evolution'. In: *American naturalist* (1896), pp. 536–553.

[2] Josh Bongard. 'Evolving modular genetic regulatory networks'. In: *Computational Intelligence, Proceedings of the World on Congress on*. Vol. 2. IEEE. 2002, pp. 1872–1877.

[3] Josh C Bongard. 'Evolutionary robotics'. In: *Communications of the ACM* 56.8 (2013), pp. 74–83.

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

[5] E Borenstein, I Meilijson and E Ruppin. 'The effect of phenotypic plasticity on evolution in multipeaked fitness landscapes'. In: *Journal of evolutionary biology* 19.5 (2006), pp. 1555–1570.

[6] Nick Cheney et al. 'Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding'. In: *Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference*. ACM. 2013, pp. 167–174.

[7] Richard Dawkins. *The selfish gene*. Oxford University Press, 1976.

[8] Kalyanmoy Deb. 'Multi-objective optimization'. In: *Search methodologies*. Springer, 2014, pp. 403–449.

[9] Kalyanmoy Deb et al. 'A fast and elitist multiobjective genetic algorithm: NSGA-II'. In: *Evolutionary Computation, IEEE Transactions on* 6.2 (2002), pp. 182–197.

[10] Stephane Doncieux et al. 'Evolutionary robotics: what, why, and where to'. In: *Frontiers in Robotics and AI* 2 (2015), p. 4.

[11] Stéphane Doncieux et al. 'Evolutionary robotics: Exploring new horizons'. In: *New Horizons in Evolutionary Robotics*. Springer, 2011, pp. 3–25.

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

[13] Matthew Field et al. 'Motion capture in robotics review'. In: *Control and Automation, 2009. ICCA 2009. IEEE International Conference on*. IEEE. 2009, pp. 1697–1702.

[14] Dario Floreano and Francesco Mondada. 'Evolution of plastic neurocontrollers for situated agents'. In: *From Animals to Animats 4, Proceedings of the 4th International Conference on Simulation of Adaptive Behavior (SAB" 1996)*. LIS-CONF-1996-001. MA: MIT Press. 1996, pp. 402–410.

[15] Dominic R Frutiger, Josh C Bongard and Fumiya Iida. 'Iterative product engineering: Evolutionary robot design'. In: *Proceedings of the fifth international conference on climbing and walking robots*. Professional Engineering Publishing. 2002, pp. 619–629.

[16] Salvador García et al. 'A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 special session on real parameter optimization'. In: *Journal of Heuristics* 15.6 (2009), pp. 617–644.

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

[18] Sabine Hauert, J-C Zufferey and Dario Floreano. 'Reverse-engineering of artificially evolved controllers for swarms of robots'. In: *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*. IEEE. 2009, pp. 55–61.

[19] Geoffrey E Hinton and Steven J Nowlan. 'How learning can guide evolution'. In: *Complex systems* 1.3 (1987), pp. 495–502.

[20] Himanshu Jain and Kalyanmoy Deb. 'An improved adaptive approach for elitist nondominated sorting genetic algorithm for many-objective optimization'. In: *Evolutionary Multi-Criterion Optimization*. Springer. 2013, pp. 307–321.

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

[22] Dong-Hwa Kim and Ajith Abraham. 'A hybrid genetic algorithm and bacterial foraging approach for global optimization and robust tuning of PID controller with disturbance rejection'. In: *Hybrid Evolutionary Algorithms*. Springer, 2007, pp. 171–199.

[23] Sylvain Koos, J-B Mouret and Stéphane Doncieux. 'The transferability approach: Crossing the reality gap in evolutionary robotics'. In: *Evolutionary Computation, IEEE Transactions on* 17.1 (2013), pp. 122–145.

[24] Minh Nghia Le et al. 'Lamarckian memetic algorithms: local optimum and connectivity structure analysis'. In: *Memetic Computing* 1.3 (2009), pp. 175–190.

[25] Joel Lehman and Kenneth O Stanley. 'Abandoning objectives: Evolution through the search for novelty alone'. In: *Evolutionary computation* 19.2 (2011), pp. 189–223.

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

[27] Henry B Mann and Donald R Whitney. 'On a test of whether one of two random variables is stochastically larger than the other'. In: *The annals of mathematical statistics* (1947), pp. 50–60.

[28] Peter Merz. 'Advanced fitness landscape analysis and the performance of memetic algorithms'. In: *Evolutionary Computation* 12.3 (2004), pp. 303–325.

[29] Pablo Moscato. 'On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms'. In: *Caltech concurrent computation program, C3P Report* 826 (1989).

[30] Ferrante Neri and Carlos Cotta. 'Memetic algorithms and memetic computing optimization: A literature review'. In: *Swarm and Evolutionary Computation* 2 (2012), pp. 1–14.

[31] Ferrante Neri and Carlos Cotta. 'Memetic algorithms and memetic computing optimization: A literature review'. In: *Swarm and Evolutionary Computation* 2 (2012), pp. 1–14.

[32] Ferrante Neri and Ernesto Mininno. 'Memetic compact differential evolution for cartesian robot control'. In: *Computational Intelligence Magazine, IEEE* 5.2 (2010), pp. 54–65.

[33] Stefano Nolfi and Dario Floreano. 'Learning and evolution'. In: *Autonomous robots* 7.1 (1999), pp. 89–113.

[34] Yew Soon Ong and Andy J Keane. 'Meta-Lamarckian learning in memetic algorithms'. In: *Evolutionary Computation, IEEE Transactions on* 8.2 (2004), pp. 99–110.

[35] Yew-Soon Ong et al. 'Classification of adaptive memetic algorithms: a comparative study'. In: *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on* 36.1 (2006), pp. 141–152.

[36] Ingo Paenke, Yaochu Jin and Jürgen Branke. 'Balancing population- and individual-level adaptation in changing environments'. In: *Adaptive Behavior* 17.2 (2009), pp. 153–174.

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

[38] Matt Quinn et al. 'Evolving controllers for a homogeneous system of physical robots: Structured cooperation with minimal sensors'. In: *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 361.1811 (2003), pp. 2321–2343.

[39] John Rieffel et al. 'Evolving soft robotic locomotion in PhysX'. In: *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*. ACM. 2009, pp. 2499–2504.

[40] Eivind Samuelsen and Kyrre Glette. 'Real-World Reproduction of Evolved Robot Morphologies: Automated Categorization and Evaluation'. In: *Applications of Evolutionary Computation - 18th European Conference, EvoApplications 2015*. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2015.

[41] Eivind Samuelsen, Kyrre Glette and Jim Torresen. 'A hox gene inspired generative approach to evolving robot morphology'. In: *Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference*. ACM. 2013, pp. 751–758.

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

[43] Karl Sims. 'Evolving 3D morphology and behavior by competition'. In: *Artificial life* 1.4 (1994), pp. 353–372.

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

[45] Richard A Watson, SG Ficiei and Jordan B Pollack. 'Embodied evolution: Embodying an evolutionary algorithm in a population of robots'. In: *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*. Vol. 1. IEEE. 1999.

[46] Darrell Whitley, V Scott Gordon and Keith Mathias. 'Lamarckian evolution, the Baldwin effect and function optimization'. In: *Parallel Problem Solving from Nature-PPSN III*. Springer, 1994, pp. 5–15.

[47] Frank Wilcoxon. 'Individual comparisons by ranking methods'. In: *Biometrics bulletin* (1945), pp. 80–83.

[48] Tony D. Williams. *The Penguins*. Oxford University Press, 1995.

[49] David H Wolpert and William G Macready. 'No free lunch theorems for optimization'. In: *Evolutionary Computation, IEEE Transactions on* 1.1 (1997), pp. 67–82.

[50] Juan Cristóbal Zagal and Javier Ruiz-Del-Solar. 'Combining simulation and reality in evolutionary robotics'. In: *Journal of Intelligent and Robotic Systems* 50.1 (2007), pp. 19–39.