# Towards Realising Multimodal Robots

**SHANKER GANESH RADHAKRISHNA PRABHU**

A thesis submitted in partial fulfilment of the
requirements of the University of Greenwich
for the Degree of Doctor of Philosophy

February 2019

# DECLARATION

I certify that the work contained in this thesis, or any part of it, has not been accepted in substance for any previous degree awarded to me or any other person, and is not concurrently being submitted for any other degree other than that of Doctor of Philosophy which has been studied at the University of Greenwich, London, UK.

I also declare that the work contained in this thesis is the result of my own investigations, except where otherwise identified and acknowledged by references. I further declare that no aspects of the contents of this thesis are the outcome of any form of research misconduct.

I declare any personal, sensitive or confidential information/data has been removed or participants have been anonymised. I further declare that where any questionnaires, survey answers or other qualitative responses of participants are recorded/included in the appendices, all personal information has been removed or anonymised. Where University forms (such as those from the Research Ethics Committee) have been included in appendices, all handwritten/scanned signatures have been removed.

Student Name: Shanker Prabhu
Student Signature: ……..
Date: ……..

First Supervisor's Name: ……..
First Supervisor's Signature: ……..
Date: ……..

# ACKNOWLEDGEMENTS

Being able to write this now feels so surreal. I am deeply grateful to all those who stayed around during this bumpy ride.

Firstly, I would like to thank my supervisory team Prof. Peter Kyberd, Dr. Richard Seals, Dr. Jodie Wetherall and Dr. Wim Melis, over the course of this PhD journey. Thank you, Richard, for helping me immensely in building the initial shape of this project. Thanks to Jodie, for my experiments would not have been easily materialised on the HPC system, which massively expedited the results generation. Thanks to Wim for joining the team in the last few months and for providing valuable inputs during our discussions.

My heartfelt thanks to Peter for guiding me and helping steady the ship in uncertain times both as a supervisor and as the head of the department. I can honestly say that this would not have been possible without your wisdom and support throughout the journey. Again, thanks for bearing with me even during those 'demanding' times. I have learnt a lot from you, and I consider myself incredibly fortunate to have been mentored by you.

I also would like to thank the Vice-Chancellor of the University of Greenwich for supporting me with a scholarship during this PhD.

This journey would not have been possible without support from many more individuals. Thank you Prof. Sadanand Gokhale for believing in me over the years. I owe so much to you for my progress since I started my career in academia. Sincere thanks to Ann Dunn for performing the painstaking task of proofreading this document. Thanks to my friends for their continuous support, especially Martin for the consults while coding the massive piece of software. Of course, I would not have been here without the support of my family. My gratitude to my parents for standing by me without necessarily understanding what I was doing and why. And my brother Girish for his encouragement, particularly when things looked bleak.

Last but not least, thanks to the higher power that possibly enabled all of the above to help me in this endeavour.

# ABSTRACT

Evolutionary Algorithms (EAs) have been applied in co-evolutionary robotics over the last quarter century. They simultaneously generate the robot body morphology and controller algorithm through artificial evolution. However, the literature shows that in this area, it is still not possible to generate robots that perform better than conventional manual designs, even for simple tasks. This thesis describes steps undertaken to improve the co-evolution process. The investigation concerns two key areas of co-evolutionary robotics. The first is in fitness function development, which plays an integral part in selecting parents for mating during evolution. The effect of an incremental fitness function based on established algorithmic techniques from specific task domains of robotics is studied. An A-star algorithm-based fitness function for path planning is designed and implemented for co-evolution of robots for navigation and obstacle avoidance. Results show that the trajectories of robots that reach a goal using the A-star algorithm-based fitness function are shorter than the robots evolved with a basic distance-based fitness function. The second area of study is the role of the controller evolution in the co-evolution process. Inspired by natural paradigms of evolution coupled with learning in biological organisms, a Reinforced Co-evolutionary Algorithm (ReCoAl) is proposed. ReCoAl works by allowing a direct policy gradient based Reinforcement Learning algorithm to improve the controller of every evolved robot to better utilise the available morphological resources before the fitness evaluation. The findings indicate that the learning process has both positive and negative effects on the progress of evolution, similar to observations in evolutionary biology.

# CONTENTS

# TABLES

# FIGURES

# ABBREVIATIONS

| | |
|---|---|
| AI: | Artificial Intelligence |
| AMEA: | Adaptive MEA |
| ANN: | Artificial Neural Network |
| BP: | Backpropagation |
| CAD: | Computer-aided Design |
| CCC: | Constraint Compliant Control |
| CMA-ES: | Covariance Matrix Estimation ES |
| CNN: | Convolutional Neural Network |
| CPG: | Central Pattern Generator |
| CPPN: | Compositional Pattern-Producing Networks |
| CTRNN: | Continuous Time RNN |
| DDRRT: | Dynamic Domain RRT |
| DE: | Differential Evolution |
| DOF: | Degrees of Freedom |
| DH: | Denavit-Hartenberg |
| DS: | Direct Policy Search |
| D-Star: | Dynamic-star |
| EA: | Evolutionary Algorithm |
| EC: | Evolutionary Computation |
| EM: | Expectation Maximization |
| ER: | Evolutionary Robotics |
| ERL: | Evolutionary Reinforcement Learning |
| ES: | Evolution Strategy |
| FNN: | Feed-forward Neural Network |
| FPS: | Fitness Proportional Selection |
| FSM: | Finite State Machine |
| GA: | Genetic Algorithm |
| GP: | Genetic Programming |
| GPU: | Graphics Processing Unit |
| IMU: | Inertia Measurement Unit |
| IR: | Infrared |
| JEAF: | Java Evolutionary Algorithm Framework |
| LEO: | Last Elite Opponent |
| LPA-star: | Lifelong Planning A-star |
| LSTM: | Long Short-Term Memory (RNN) |
| MARS: | Machina Arte Robotum Simulans |
| MC: | Monte Carlo |
| MDP: | Markov Decision Process |
| MEA: | Multi-Chromosome EA |
| ML: | Machine Learning |
| MOEA: | Multiple Objective EA |
| NEAT: | NeuroEvolution of Augmenting Topologies |
| NN: | Neural Network |
| NSGA-II: | Non-dominated Sorting GA-II |
| ODE: | Open Dynamics Engine |

| | |
|---|---|
| OLPOMDP: | Online POMDP |
| PD: | Proportional-Derivative |
| PGRD: | Policy Gradient for Reward Design |
| POMDP: | Partially Observable MDP |
| PRM: | Probabilistic Road Map |
| PS: | Policy Search |
| ReCoAl: | Reinforced Co-evolutionary Algorithm |
| RL: | Reinforcement Learning |
| RNN: | Recurrent NN |
| ROS: | Robot Operating System |
| RRG: | Rapidly Exploring Random Graph |
| RRT: | Rapidly Exploring Random Tree |
| SCARA: | Selective Compliance Articulated Robot Arm |
| SGA: | Simple GA |
| SOEA: | Single Objective EA |
| SL: | Supervised Learning |
| TD: | Temporal Difference |
| TPU: | Tensor Processing Unit |
| TGA: | Two-level GA |
| UL: | Unsupervised Learning |

# Chapter 1.

# Introduction

A modern mechanical robot is a mechatronic device built with a combination of sensors, actuators and controller. Each of these fundamental parts is combined to enable the robot to perform a specific and/or range of tasks. In traditional design, engineers visualise a possible solution and build a hardware prototype after cycles of design, modelling and simulation as per application. An alternate approach to achieve the same goal would be to exploit the power of computing technology to optimise or even to automatically generate solutions. However, even with the availability of advanced mechatronic and computing technologies, robot building usually follows a manual approach. There have been notable commercial attempts to automate design such as the recently announced Autodesk *Project Dreamcatcher* [1] or Frustum *Generate* [2], which can automatically optimise design topologies. This renewed commercial interest may be considered as beginning a new era of automated mechatronic design. Furthermore, with ever-increasing cheap and fast computing power and easy access to cloud computing systems, the time has come to exploit these in computer-aided design to develop mature technologies for mechatronic system generation.

In the broad area of mechatronic system generation, this thesis investigates the subtopic of automated design of autonomous mobile robots that can navigate across a simple environment. To ultimately achieve automatic mechatronic system generation, research should ideally move from purely mechanical design optimisation (by modifying an already existing design) to firstly automated mechanical structure generation with manual controller design, followed by automatic controller design on an already available mechanical body and finally to an automatic design of both mechanical body and controller. The literature shows that commercially available mechanical structure optimisation packages are the most advanced yet in the above categories, with research actively being undertaken in other areas of automated mechatronic system design.

Among automated design methods suitable for robot development, unified modelling strategies or Evolutionary Computation (EC) strategies are viable options. Unified modelling of robots involves creating a modular representation of various parts of the robot, and linking them to a framework to generate a robot body or controller [3], while EC strategy uses the application of biological evolutionary principles to evolve the robot body and/or controller [4]. The context of this thesis lies in the latter area that is commonly referred as Evolutionary Robotics (ER).

The algorithms used in EC are called Evolutionary Algorithms (EAs). These are a class of Artificial Intelligence (AI) algorithms that use a population to stochastically search for solutions to a problem. In EAs, a population of randomly generated individuals/solutions improves over generations with the help of feedback regarding the respective individual's performance assessed through a fitness function.

The total range of solutions of a problem can be figuratively referred to as a 'landscape'. If the vertical axis represents the fitness of the solution and the other axes represent various parameters that are being optimised, then moving about the landscape changes the fitness of the solution. For simplicity, if it is assumed that two variables are to be optimised, $x$ and $y$ axes would represent the two parameters and $z$ axis the fitness. The role of EA will be to navigate through this 3D landscape consisting of multiple hills, valleys and depressions. Here, the higher the hill, the better the corresponding fitness. These hills are referred as local optima and the highest point in the landscape is referred to as the global optimum. So, the optimal solution to the problem being solved will be at the global optimum. The core idea behind the use of stochastic algorithms is that it creates a chance for the solver to reach this global optimum. As with other stochastic optimisation methods, EAs also search for available solutions in the landscape. The differentiating feature of EAs is how this exploration of the landscape is conducted during evolution.

An EA which works through variation operations (called mutation or crossover operations, discussed in the next chapter) tries to climb some of the available hills by starting at random points with multiple individuals of a randomly initialised population of solutions. During this exploration, positive changes to fitness means individuals are climbing up-hill. The expectation from the solutions at the end of evolution is to identify the right hill to climb

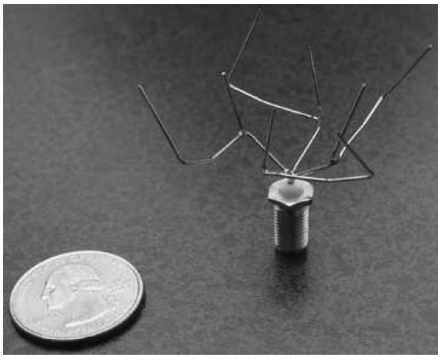


**Figure 1. Evolved antenna placed on Space Technology 5 aircraft [5].**

and reach its peak. In reality, what may be observed is that it depends on the specified evolution parameters, as some of the solutions start climbing the wrong hill and remain stuck there, i.e. solutions try to exploit the local maxima as if it is the global optimum. If on the other hand, evolution parameters are adjusted to avoid the exploitation problem and instead spread the search to much wider unexplored areas of the solutions space, this can lead to an inefficient search. In optimisation theory, this trade-off is referred as the *exploitation versus exploration dilemma*.

Despite these inherent drawbacks of EAs, they do offer a number of advantages. They have often been shown to evolve good solutions to ill-defined problems that current solvers cannot find. It has been observed that, "*an evolutionary algorithm is the second-best solver for any problem*" [4]. This means that while a better solver can be developed, it can come at an unreasonably high cost that outweighs the benefits of such a development. Inverse kinematic control of an articulated robot is a good example that shows the benefit of an EA: given the position of an end effector, computing the joint angles is a complex mathematical problem because it needs to calculate the inverse of several multidimensional matrices; An EA here could iteratively calculate joint angles from robot base to end effector and get away from the complex calculations if the problem is approached from the end effector to base instead.

Furthermore, literature has shown that EAs often develop unimagined solutions when compared to the results from a traditional design process. An example is when an EA was used to evolve an antenna that was placed on NASA's Space Technology 5 aircraft launched in 2006 [5]. The evolved antenna seen in Fig. 1 has an unusual shape which a manual designer cannot easily visualise. This shows how EAs can move away from bias and

limitations of the conventional design process. Furthermore, the antennas evolved in the work demonstrated much better performance than manual designs while taking only just over half the design time.

Another benefit EAs offer in robotics is in their ability to take a holistic approach to problem solving. That is, when a robot is evolved to solve a task, instead of trying to optimise individual elements of the robot, each robot is looked at as a single entity. This allows for easier analysis of the entire robot behaviour and quicker identification of better performing ones.

EAs also have the ability to guarantee a solution (though not optimal) at any point of evolution that is not offered by other problem-solving methods [6]. However, they may not be good solutions as the ability of an EA to generate a good solution is often dependent on the complexity of the problem and time allocated for evolution.

These advantages have helped researchers approach robot design and generation as a single problem as opposed to iterative development with human feedback at the core in the traditional design cycle. This, coupled with the developments in current literature (more details in *Chapter 2*) on how robots have been evolved just with goal specification, acts as the motivation to explore the area of ER for this thesis for simultaneous evolution or generation of mobile robot body and controller.

## 1.1. Objectives and Methodology

The overarching aim of this thesis is to improve the current state of the art of automatic mobile robot body and controller development. For this, ER is chosen as a viable area of exploration and two new approaches are tested:

1. The effect of fitness functions on co-evolution is studied with the help of an A-star based fitness function and Euclidian distance-based fitness function.
2. The effect of learning on co-evolution is tested with the help of a Reinforcement Learning (RL) based algorithm applied to every evolved robot.

Fitness function design is a known hurdle for evolution of satisfactory solutions to problems in EC (mobile robots in this case). But it is not given much attention in the literature for co-evolving robot body and controller. After extensive research on the fitness functions currently used in co-evolution and evolution in general, a new task-specific fitness function was designed, through a novel approach of incorporating established task-based problem-solving algorithms applied in traditional robot design into fitness functions. For this, an A-star algorithm which is commonly used for robot path planning was built into a fitness function while evolving robots for obstacle avoidance and navigation. This new fitness function's performance was then compared with an Euclidian distance-only based fitness function while evolving robots in a virtual 3D arena.

For the second approach, work was inspired from the observations reported in literature and results from experiments with the A-star fitness function. It was observed that a key problem of evolved robots is the inability of the evolved controller to effectively control available robot body parts. Therefore, a reinforcement learning based system was designed to help every evolved controller to better handle the available resources. This learner combined evolution was tested against pure evolution of mobile robot body and controllers. Specific attention was also given to help evolve the functionality of using distance sensors when available on evolved robots. This was attempted with the help of a reward function that encourages the robot to move in the direction its sensor/s are facing. The same tasks were used for testing the A-star fitness function.

Both approaches were implemented in RoboGen [7], an open-source mobile robot evolution platform. Experiments were run on a High-Performance Computing platform that allowed evolution to develop in parallel. More details of implementation are available in *Section 2.3*.

## 1.2. Chapter Outline

The thesis is a step towards improving automatic mobile robot generation while using evolutionary robotics principles. It is organised as follows: *Chapter 2* provides a review of the literature with an emphasis on the use of EC in aiding the design process for robotics. It begins with unified modelling methods used for robot design, followed by the evolutionary-aided design process that uses EAs. In EC, the literature is first approached from a much

broader perspective to cover the general application areas that have benefited from EAs. *Section 2.2* covers efforts to determine body morphology of robots through evolution and body morphology with controller of robots or similar creatures through co-evolution. The works are reviewed from the perspective of how different algorithms are applied and includes a brief explanation of how they are implemented. The section reveals the limited success so far achieved from co-evolution, with evolved robots still only being able to perform primitive tasks despite more than 25 years of research. Eventually, the under focussed area of fitness function design is identified for further exploration to ultimately improve robot co-evolution.

The next section (*2.3*) provides a brief introduction to *RoboGen* [7], the software tool modified and used in the subsequent chapters. The EA theory covered in *Section 2.2.1* along with the functional explanation of *RoboGen* acts as a background to the work explained in the rest of the chapters. In the following sections of *Chapter 2*, a brief review of path planning algorithms, Reinforcement Learning (RL) algorithms and EA-RL hybrid algorithms are discussed as the algorithms proposed in subsequent chapters apply these components.

Selection of evolution parameters is a known challenge in ER. The literature in *Section 2.2* shows that there is no clear consensus on how these parameters are selected for evolving robots for different applications. Therefore in *Chapter 3,* the effect of parameters on the evolution process is studied through multiple experiments. The focus here is to observe the behaviour of evolution while developing robots for performing navigation and obstacle avoidance. The results indicate that the robots are not even performing basic directional changes necessary to reach the goal or evade obstacles despite making variations to the available evolution parameters. This suggest the need for building better fitness functions as the results imply that the simple distance-only fitness function used is insufficient in conveying the goal requirements to the evolver.

As a result, *Chapter 4* describes the design of a unique task-based fitness function, and its performance. In this chapter, it is hypothesised that an incremental fitness function based on established techniques in specific task domains in robotics will aid the co-evolution process. An A-star algorithm-based fitness function for path planning is designed and implemented for evolving the body plans and controllers of robots for navigation and obstacle avoidance

tasks. It is shown here that using this concept, fitter robots have evolved in most cases, when compared to simple distance-only based fitness functions. However, variable performance of an evolver with the A-star fitness function demonstrates that the results are inconclusive. Hence, problems associated with the fitness function are identified and recommendations have been made for designing future fitness functions based on observations from the performed experiments.

The unreliable performance of the A-star fitness function mainly suggests that the fitness function should not just be end result specific, but a combination of multiple functional and non-functional aspects of the robot. It is also noted that the main hindrance to satisfactory robot evolution is poor controller evolution resulting in the simultaneous variation operations performed on the robot morphology and controller. Simply, the controller is a result of pure evolution and is not given a chance to improve before fitness is calculated. Therefore, to improve co-evolution and to bring artificial robot evolution a step closer to the performance of biological evolution, a controller learning phase is introduced for every offspring robot born. The algorithm is referred to as the Reinforced Co-evolution Algorithm (ReCoAl) which is a hybrid of an evolutionary algorithm and a reinforcement learning algorithm. It combines the evolutionary and learning processes in nature to co-evolve robot morphology and controllers. Results from ReCoAl indicate that the conducted learning process can have negative and positive effects on the evolution process.

The final chapter provides a discussion and conclusion of the performed work along with future directions for co-evolution of body and controller of robots.

The original contributions outlined in this thesis that have appeared in peer reviewed publications are as follows:

- A comprehensive list of works in the area of robotics that report the use of EAs for morphology only evolution, simultaneous morphology and controller evolution are reported and published as,

  **S. G. Radhakrishna Prabhu**, R. C. Seals, P. J. Kyberd, and J. C. Wetherall, “A survey on evolutionary-aided design in robotics,” *Robotica*, vol. 36 (12), pp. 1804-1821, Dec. 2018.
  https://doi.org/10.1017/S0263574718000747

- The effect of evolution parameters while co-evolving mobile robots are studied and published as,

  **S. G. Radhakrishna Prabhu,** and R. C. Seals, "Effect of change in evolution parameters on evolutionary robots," *2nd Medway Engineering Conference on System: Efficiency, Sustainability and Modelling*, June 2017.
  http://gala.gre.ac.uk/17141/
- The use of sophisticated fitness functions in evolutionary robotics for co-evolution of mobile robots are tested with the help of an A-star based fitness function and is available as,

  **S. G. Radhakrishna Prabhu,** P. J. Kyberd, and J. C. Wetherall, "Investigating an A-star Algorithm-based Fitness Function for Mobile Robot Evolution," *22nd International Conference on Systems: Theory, Control and Computing (ICSTCC)*, Sinaia, 2018, pp. 771-776.
  10.1109/ICSTCC.2018.8540734
- An EA-RL hybrid algorithm, called ReCoAl, specific to co-evolution of mobile robots is designed and tested to allow the co-evolver to generate better mobile robots. The work is available as,

  **S. G. Radhakrishna Prabhu,** P. J. Kyberd, W. J.C. Melis and J. C. Wetherall, "Does lifelong learning affect mobile robot evolution?" *25th International Conference on Soft Computing (MENDEL)*, 2019 (in press).

# Chapter 2.

# Literature Review

This survey in automatic mechatronic structure generation is grouped into two main categories, the first dealing with unified modelling methods and the second dealing with EA based methods applied for robot morphology and controller generation.

## 2.1. Traditional Methods

Traditional methods commonly referred to as unified modelling of robots work by developing a common framework to link various parts of a robot [3]. In these methods, different mechanical modules are categorised and combined to enhance the functionality of the final system and the definition or specification of these is used to explain the ultimate features. The concept was applied in [8] to link kinematic and dynamic models of various mechanical parts of the robot. Instead of using low-level individual parts, the authors considered a significantly functional part, e.g. a robot leg, as an element. The model was then reused when more than one leg was attached to the robot and it was later converted into a tree type topology for performing kinematic analysis of the overall system or for control software development.

In another work, mathematical models of individual mechanical elements were combined to analyse complex mechanical systems in software [9]. However, this failed to consider the mechatronic features of the robot. A similar method was applied to a modular reconfigurable robot in [10]. In the papers discussed above, the inter-component kinematic relationship was derived, and a layered software architecture was proposed.

A model-based industrial manipulator development is presented in [11]. With given robot specifications, *MATLAB Simulink* blocks automatically created a mathematical model for calculating various forward and inverse kinematic parameters. In a different context, [12] explained methods used by various researchers to automate *LEGO* brick assembly

construction. A *LEGO* brick can be considered as similar to an individual mechanical element in a robot. Algorithms for solving the *LEGO* builder problem utilised mathematical models of individual elements and knowledge of the expected result to arrive at the final shape. Most of these methods initially used a *greedy method* (or similar) to build the object and later optimised the assembly. Recent work in this area is elaborated in [13] where images of the final shape were used to automatically generate hardware building instructions. Unit bricks built the initial shape after which cost and energy functions were applied to optimise and add bricks of multiple configurations. In works that dealt with robots, Burgali *et al.* proposed software abstractions to multiple robot parts to aid in developing software functional components [9]. Similarly, Treanor *et al.* created an autonomous controller development system which could generate a layered controller for various tasks [14]. Kjaergaard [15] created a parameterised model for autonomous mobile robots describing physical configuration, kinematics, dynamics, sensors and actuators. Later these models were embedded into a Smart Parameter Framework in *Python* for configuring a robot's control system just for navigation.

The main criticism for the papers above is that their focus was on the controller development, and therefore the configuration of the robot is known at the beginning of the process. None of the above reported performing automatic structure generation (except with *LEGO* bricks but they lacked any sensory feedback). Various groups designed concepts for defining individual pieces of a robot mathematically at different levels (element or functional levels) and linked them with respective software for the development of complex robots. Despite the publications originating from top-tier universities and being published in influential journals and conferences, their adoption in subsequent papers has been low. This suggests that just a few have been able to make progress in this direction and it casts serious doubts about considering further exploration of this area for automatic robot generation. In contrast, the area of ER has shown multiple attempts at autonomous robot generation from evolving morphologies of articulated [16] to humanoid robots [17] and evolving morphologies and controllers of virtual creatures [18] to actual buildable robots [19].

## 2.2. Evolutionary Robotics (ER)

Evolutionary Computation (EC) stems from Darwin's theory of evolution [20] and Mendel's experiments in hybridising plants [21]. The concept employs various mechanisms of evolutionary theory to stochastically evolve a population of solutions. EAs are a subset of EC introduced in the 1970s by Holland [22] as Genetic Algorithms (GA). GAs use a basic structure that changes over time by minor variations, made at each time-step. At the same time, other researchers developed conceptually similar algorithms like Evolution Strategies (ES) and Genetic Programming (GP). In the early 1990s, all these separately developed algorithms were classed under the single term of Evolutionary Computing.

It took almost two decades for ECs to be successfully applied to robotics. Evolutionary Robotics (ER) is concerned with the generation of autonomous robots using the principles of evolutionary computing [23]. EC strategies have been applied to autonomous robot development in evolving morphologies and controllers separately or co-evolving both simultaneously [23]. In ER, the applications of EAs can spread across various domains from traditional mobile and manipulator robotics to newer areas such as modular, swarm, bio, developmental and soft robotics [24]. This section is concerned with both the evolution of the morphology alone, along with the co-evolution[1] of morphology and controller in the traditional areas of robotics.

Before making a decision on what areas of evolutionary robotics needs to be concentrated on in this thesis, it is necessary to have a broader picture of EC in general to examine if any successful practices can be imported.

Unlike other technologies in robotics that have been actively applied to solve real world problems, literature has shown how EC approaches in robotics are yet to reach that level. However, there are a wide range of areas that have directly benefited from the use of EAs.

[1]The term co-evolution in literature can have different meanings depending on the context. For instance, the process when multiple populations evolve in parallel is also referred as co-evolution. However in this thesis, co-evolution refers to the process of simultaneous evolution of robot morphology and controller.

These algorithms have been primarily developed to be generic solvers for problems that do not have a formal method that computes a solution in polynomial-time. Following the advent of faster computing systems, the developed theory has been applied to a variety of areas, from computer science (generation, optimisation and debugging code [25]), chemistry (molecular structure optimisation [26]) and finance (stock market traders' generation [27]) to economics (demand and supply modelling [28]). The most successful applications of EAs can be seen when optimisation is the main target. Consequently, EAs have been used in the production industry sector for production planning, sequencing, and scheduling optimisation [29].

Moreover, the area of optimisation has a class of problems that are often used to benchmark different algorithms. Literature shows that EAs have also been applied to these problems with great success. A famous example is the *Travelling Salesman Problem* (TSP), where the algorithm has to find the optimal route connecting multiple cities while minimising the total distance travelled [30]. Even though such problems can also be used to benchmark algorithms used in ER, the multidimensionality of ER applications mean that benchmarking using classical problems does not provide an accurate picture.

More examples of optimisation based approach with EAs can be seen in diverse sectors: in traffic, for routing and scheduling of trains [31] and vehicles [32]; in energy, for load management [33]; in telecommunication, for network layout optimisation [34]; in education, for timetabling classes and exams [35]; in the military, for mission planning [36]; in government, for optimising budget rules [37]; in financial service, for credit application scoring [38], risk management [39], insurance risk assessment [40] etc.

In the area of control, EAs have been demonstrated in plant dynamics modelling [41] and controller design for both linear and non-linear systems [42]. Though all these need an offline approach during development, EAs have still been shown to work well in comparison with traditional methods except when designing linear systems [42]. System identification, which is a nearby area of control, has also seen several successful applications of EAs [43].

EAs have also been used in pattern recognition. An example is when a Cellular Neural Network evolved with a GA was able to detect 70 % cases of multiple sclerosis white matter lesions from brain images [44]. In the area of pattern recognition and feature extraction,

Zhang *et al.* explained how disjointed the field is while using EAs for solving problems [45]. This is true in mobile robot co-evolution too.

Moving to areas closer to this thesis, EAs for structural design have received significant attention in the recent past. It has been demonstrated how EAs can also be used for evolving novel designs and to move away from the typical optimisation-oriented application approach commonly observed in the areas discussed till now [46]. The open-ended design representations used in structural design are particularly visible in virtual creature co-evolution discussed later.

Another step closer to co-evolution is the area of controller-only evolution in mobile robots that accounts for close to 95 % of the research in ER [122]. The themes of applications in this field are very similar to the other areas covered above. Optimisation of various control parameters is very common and evolving complete controllers from scratch for different tasks have also been demonstrated. Concepts such as combining evolution and learning, enabling online learning, environment-driven evolution, enabling automatic repair and open-ended evolution have also been tested [4].

The state of the art in co-evolution of mobile robots can only be appreciated after gaining an insight of the underlying concepts in EAs. Therefore, the following section begins with a discussion on the near standard structure of EAs. This is followed by explanations on how different researchers have used EAs in their work.

### 2.2.1. Algorithms for Morphology Evolution

#### *2.2.1.1. The basic structure of evolutionary algorithms.*

As the name suggests, the algorithms are inspired from biological evolution. Consequently, each of the steps involved can be closely mapped with the different stages in biological evolution as natural selection searches for more fit living beings. The topic is explained in detail in [6].

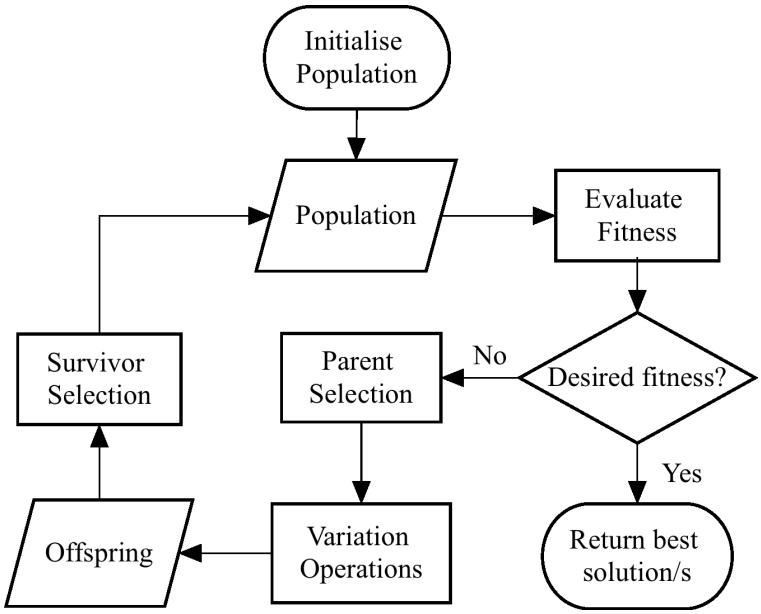



**Figure 2. Evolutionary Algorithm flowchart.** *The randomly initialised solutions population passes through generations of evolution where its suitability for the task is measured through a fitness function that later decides if the robot becomes a parent or not. The parents mate to generate offspring through variation operations and a new population is built with children and old individuals. The cycle continues until a satisfactory solution is generated by the system or the maximum number of generations is reached or the maximum computation time is exceeded.*

The structure of a typical EA as shown in Fig. 2 is as follows:

1. Initialise a population of candidate solutions.
2. Evaluate each candidate.
3. Select parent/s.
4. Recombine parents.
5. Mutate offspring.
6. Replace generation.

Steps 2 to 6 are repeated until the results are sufficiently close to the desired objective or until the computation time is exceeded. Before discussing the above procedure, it is necessary to explain how the candidate solutions are represented.

*Genotype and phenotype.* In biology, a genome carries the features of the organism, through a set of genes, and *genotype* is the equivalent term in ER. The genotype that is converted into the physical or observable entity of the candidate solution or robot is referred to as *phenotype. Phenotype space* is where the observable properties of a robot are found. When

**Table I. Different types of genotype representations.**

| Genotype representation | Citations |
|---|---|
| Binary* | [17], [47], [48] |
| Integer* | [49] |
| Real number* | [50]-[52] |
| String* | [53] |
| Matrix | [54] |
| Vector | [16], [55]-[57] |
| Graph | [18], [58]-[62] |
| Tree | [7], [19], [63]-[68] |
| *Hox* gene | [57], [69] |
| Compositional Pattern-Producing Networks (CPPN) encoding | [49], [70] |
| L-system | [71] |
| Parametric | [17], [47], [56], [60], [66], [67], [72]-[75] |
| Open-ended | [19], [50], [65], [76], [70], [77] |

*Basic genome types used in other representations.

the genotype encodes a description of the phenotype (or robot), a genotype-phenotype mapping transfers the information between genotype space and phenotype space. Evaluation of each individual is performed on the phenotype while all other steps during evolution are conducted in the genotype space.

The genotype holds morphological and control parameters or control algorithms according to the application. Various morphological parameters convey details pertaining to body size, weight, wheel diameters, spatial information about position (coordinates or angle), and any other relevant information. The control side may include various neural network parameters, neural network wiring or other parameters unique to the specific controller type under use. The main types of genotype representations that have been applied in ER are listed in Table I.

*Genotype-phenotype mapping.* Each candidate's fitness (or suitability for the task) is estimated through analysing the performance of the corresponding phenotype in the phenotype space. However, before this can be evaluated, each candidate needs to be

converted from its genotype to phenotype. A genotype to phenotype mapping is used during this conversion.

*Population.* The initial population acts as a seed for solutions which are later evolved into solution(s) with the desired performance. Despite the speed of successful evolution depending on the size and quality of an evolving population, population design in ER is an under reported area. Population initialisation has always been random to ensure unbiased evolution except in [57] and [79]. Regarding population size, the area of co-evolution shows that it varies from 15 to 1000 candidate solutions when the fitness evaluation is performed in computer simulation. But in a unique case, actual robots were built for evaluation from 1 to 3 elements to avoid problems due to simulator accuracy [78]. A single population evolved solutions except in [80], where the best individual was transferred to a second population when the average fitness of the first population reached a fixed value (or after 50 generations). Once twenty such transfers were completed, the solutions were further optimised with the second population. This helped to build a niche second population that was further evolved. But the authors did not demonstrate how the new process benefited evolution.

*Fitness function (for candidate evaluation).* Each candidate solution or robot in the population is tested using a fitness function applied to its phenotype's behaviour. Fitness functions observe the robot that is placed in a simulated environment and allocate a fitness value to each of them depending on a number of factors. They are task specific through individual or a combination of penalty functions, cost functions, reward functions or based on the extent of objective attainment. Different fitness functions applied in co-evolutionary robotics are listed in Table II. Fitness functions applied are further discussed in the applications section as these functions are always application oriented.

*Parent selection.* The probability of creating a better offspring depends strongly on parent selection and hence this plays a major role in determining the time taken by EAs to converge to a satisfactory solution. Normally, parents with higher fitness are allotted a higher priority to be selected for reproduction. However, to avoid solutions being trapped in local optima, parents with lower fitness are also selected, through a selection probability which is set at a

**Table II. Different types of fitness functions.**

| Fitness function based on | Citations |
|---|---|
| Movement (e.g. distance, area, speed) | [73], [76], [78], [81], [82]*, [57], [61], [83]* [47], [50], [84]*, [53], [60], [64], [65], [79], [85]*, [86]*, [80]*, [56]*, [87], [88]*, [48]*, [75]*, [89]*, [62]*, [69]*, [90]*, [66], [67]*, [91]*, [92]* |
| Energy or cost | [50], [84]*, [55]*, [90]*, [17] |
| Action (e.g. collision, touch, fall, damage inflicted or suffered, obstacle avoidance) | [76]*, [73]*, [60]*, [66]* |
| Mechanical features (e.g. centre of gravity, torque generated, load, weight, size, dexterity, mobility, tension) | [70]*, [80]*, [56]*, [88]*, [48]*, [89]*, [75]*, [55]*, [90]*, [91], [93]* |
| Design or feature (e.g. novelty, diversity) | [83]*, [63] |
| Goal specific (e.g. time to contact with prey, lifting, grasping, time on line, food eaten, competition winner, climbing ability) | [16], [51], [59], [62]*, [69]*, [55]*, [54], [92]* |
| Human feedback | [76]*, [72] |

*Combination of more than one category.

low value [6]. In most cases, two parents are mated, except when applying asexual reproduction, e.g. [70]. There are several ways to select the parents:

a. *Fitness Proportional Selection (FPS).* In FPS, the individual's selection probability depends on its absolute fitness relative to the remaining population. However, consistent selection of best candidates often leads to premature convergence or solutions getting stuck in local minima. Methods like windowing or sigma scaling are commonly used to avoid this problem [6].

b. *Ranking selection.* FPS suffers from a lack of constant selection pressure. This can be avoided with *ranking selection*. At first individuals are ranked based on their fitness. From there, a probability is allocated according to the rank and not the fitness value itself with a higher ranked individual being allocated a higher selection

**Table III. Methods of parent selection.**

| Parent selection methods | Citations |
|---|---|
| Random | [61], [62], [73], [80]* |
| Tournament selection (binary/deterministic/triple/multiple population) | [7], [47], [57], [59], [66], [69], [74]*, [51], [94]*, [75]*, [95]*, [17] |
| Fitness proportionate (FPS/rank/cost/*x* best/neighbouring pair) | [49], [58], [80]*, [51], [63], [87], [94]* |
| Non-dominated sorting | [57]*, [86]* |
| Crowd distance sorting | [86]* |
| Diversity measure | [57]* |
| Stochastic universal sampling | [89] |
| Roulette wheel selection | [48], [55], [75]*, [95]*, [16] |

*Combination of more than one category.

probability. For selecting probabilities, commonly used techniques are *roulette wheel* and *stochastic universal sampling*. *Roulette wheel selection* conceptually involves randomly selecting parents by repeatedly spinning a hypothetical roulette wheel. Each slot on the wheel represents a selection probability (allocated from a *cumulative probability distribution function*), and a parent is selected after every spin of the wheel. Among its disadvantages, a uniform selection of parents from the probability distribution is not possible as the wheel is spun more than once. To overcome this, *stochastic universal sampling* is applied where all parents are selected simultaneously from the distribution [6]. This is accomplished by using the concept similar to a *wheel of fortune,* where instead of one selecting pointer directing to a single slot, multiple uniformly spaced pointers (equal to the number of parents) select multiple slots (with their size proportional to corresponding probability) in a single spin.

*c. Tournament selection.* The parent selection methods explained so far require the probability distribution of the entire population in every generation to apply the necessary algorithm. In contrast, tournament selection works by dividing the population into groups and selecting the best for each candidate's relative fitness in that group. The prominent parent selection methods applied in the area are listed in Table III.

*Variation operations (mutation and recombination).* Variation operators are applied to the selected parents to create new offspring. Usually, both recombination and mutation after mating are implemented in that order. When a child is generated from a single parent, the process is called mutation, and when multiple parents are used, the process is referred to as recombination. Mutation involves stochastic modification of parent's genotype first through cloning and then modifying. It must be noted that the genotype should be altered at random positions and should be unbiased. Hence, slightly targeted tweaking of the genotype is not considered as mutation [6].

In the second method, recombination or crossover operation is applied to the two parents to generate offspring. In this stochastic method, random parts of the mating parents are joined to generate offspring with features from both parents. Moving away from the purely Darwinian model, theoretical studies and demonstrations have shown that using more than two parents in recombination can produce more fit offspring. Despite this, the method is seldom used [6]. There are several variations to the mutation and crossover operations when applied to solve problems in ER, and some of the significant variations are listed in Tables IV and V, respectively.

*Survivor selection (for population update).* The population size is fixed by finite resources. So, the proportion of the offspring that are added to the population and which individuals are replaced needs to be decided. Age-based and fitness-based replacement are the two main types of selection strategies. Fitness-based selection involves several methods, and the most prominent ones are discussed below.

A fixed number of least fit members are replaced from the population and the rest are moved to the next generation in a *replace the worst* strategy. It results in a fast increase of average fitness, but this may also lead to premature convergence by the population getting stuck in local minima. As a result, it is only commonly applied to large populations. *Elitism* is implemented in addition to age-based and stochastic replacement methods to ensure that the fittest member of the population is retained and not replaced by other methods. In cases where fittest is among the parent population and not among the children, the latter are discarded. The round-robin tournament is another method, in which competitions are conducted between an offspring and a randomly selected opponent from a merged parent-

**Table IV. Mutation methods.**

| Mutation operations | Citations |
|---|---|
| Fixed non-adaptive | [73] |
| Gaussian | [7], [51], [64] |
| Gaussian random | [54] |
| Standard | [16], [47], [50], [55], [60], [63], [67], [69], [70], [75], [84], [85], [90], [93] |
| Through deletion, duplication, modification | [57], [62], [65], [79], [96], [18] |
| Variable probabilities | [87] |
| Binary/float/integer | [91], [92] |
| Vector differential mutation | [55] |

**Table V. Types of crossover.**

| Crossover operations | Citations |
|---|---|
| Random | [49], [55], [96], [65], [67], [18], [91] |
| Single point | [47], [74], [88] |
| Two point | [17], [66], [81] |
| N point | [64] |
| Grafting, Copying | [59] |
| Classic | [16], [50], [60], [63], [75], [84], [90], [93], [61], [92] |
| BLX-$\propto$ | [85] |
| Segregation | [92] |

child population. Each child competes for a fixed number of times, and once all of them finish competing, the offspring with highest number of wins replace the weakest of the population [6].

Literature shows survivor selection has most time been performed using ad hoc methods. This is true especially when multiple standard methods discussed before are combined. Such examples are: deletion of a certain percentage of the population [67], [75]; variable replacement strategy [7]; elitism with tournament selection [74]; replacing worst in population with children [59]; and children replacing the entire population with the periodic injection of random individuals [87].

**Table VI. Different EAs.**

| EA | Citations |
|---|---|
| Standard GA | [7], [59], [69], [72], [80], [88], [90] |
| Last Elite Opponent Algorithm [98] | [95] |
| GA with extremal and pareto optimisation | [80] |
| Non-dominated Sorting Genetic Algorithm II (NSGA-II) [99] | [57], [86] |
| Differential Evolution (DE) | [55] |
| Multiple Objective Evolutionary Algorithm (MOEA) | [50], [54], [73] |
| Multi-Chromosome Evolutionary Algorithm (MEA) [100] | [91] |
| Genetic Programming (GP) | [63], [64], [84] |
| NeuroEvolution of Augmenting Topologies (NEAT) [101] | [70] |
| Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [102] | [56], [82] |
| Java Evolutionary Algorithm Framework (JEAF) [103] for DE, GA or CMA-ES | [19] |

*2.2.1.2. Different EAs.*

EAs normally applied have been realised using variations of the basic algorithm. Its prominent variations when applied to solve problems in ER are listed in Table VI. By far the most commonly applied EA is the Genetic Algorithm (GA), Simple GA (SGA) or canonical GA that uses a binary representation for its genotypes, FPS mechanism for parent selection, low mutation rates, and one-point crossover set as the standard recombination mechanism. The entire population is refreshed in every generation as everyone is selected for crossover and is replaced by mutated children. However, depending on the probabilities set on variation operators, there can be copies of parents in the new population. SGAs possess flaws, they converge slowly. They have been modified to include *elitism* and *tournament selection* or other selection methods to achieve faster convergence [6]. Such variations to GAs can be seen in Table VI.

The algorithms listed above were most of the time developed for solving problems outside robotics. Consequently, their testing process only involved applying them to standard

benchmarking problems. Later in ER, these have been directly applied to specific problems as no benchmarking problems exist here. This results in an inability to make conclusions on the benefits of each algorithm type. Furthermore, parameter tuning also becomes extremely difficult while applying these algorithms in ER due to the same reason.

### 2.2.2. Controller Algorithms

During co-evolution, the body and controller are evolved simultaneously with EAs as the generations progress. The controller types selected for such evolutions have been either artificial intelligence type (mainly Artificial Neural Networks (ANNs)) or traditional control technique based. Over two-thirds of works in co-evolution used an NN based controller and the next nearest are oscillatory controllers.

In an ANN-based control scheme, artificial neurons are internally wired to connect sensors (receptors), actuators (effectors) or other neurons. Output signals are generated based on the input value, corresponding internal weights, biases and other internal operations that take place in the neural network. Various arithmetic operations or oscillating signals act at different neural nodes to manipulate the input signal. As the genotype representation carries information about the control system info of each part, it gets carried to offspring and gets modified during the variation operations. Examples of such or similar systems are included in [7], [59]-[62], [79], [81], [94], [96], [98], [104].

There are controllers that can use internal feedback connections to generate output even in the absence of input signals. These are therefore useful in systems without sensors. Such examples are: a neural oscillator with PD (Proportional-Derivative) control [97]; a neural network with a Central Pattern Generator (CPG) [57]; CPGs only controller [50]; Continuous Time Recurrent Neural Networks (CTRNNs) [54], [65], [83] and; Elman's recurrent network [88]. Among these, while a PD type controller is suitable for applications that require precise control signals, CPGs are suitable to induce time dependent oscillatory signals. Furthermore, recurrent networks allow oscillatory output signal generation by using output generated in previous steps. This makes them suitable for complex problems with a large number of inputs or outputs.

Examples of fully non-ANN type controllers can be seen in: [106] where a reactive controller was evolved with a GA; a simple inverse kinematic control evolved by an EA [56]; and a simple periodic open loop control [76], [92]. Among these, reactive and open loop control have been demonstrated in mobile robot evolution due to the relative ease of implementation while also being able to reasonably solve the task.

### 2.2.3. Application Areas

The areas discussed are classified into works that deal purely with the evolution of mechanical design and works where control and morphology are co-evolved with the help of EAs. A comprehensive chronological list of works that report morphology only evolution, and co-evolution of morphology and control scheme can be found in Table VII and Table VIII, respectively.

#### *2.2.3.1. Morphology only evolution.*

In robot morphology evolution, EAs have been used for extremely constrained applications to ensure their effective use. As a result of such targeted approaches, much more positive results have been generated when compared with EA applications in co-evolution. This can be seen in articulated robot morphology designs [16], [48], [55], [63].

On the other hand, there are not many works that report the use of EAs in mobile robot body-only evolution. The main reason for this is the difficulty in designing a controller separately after the evolution is completed. Unless a mobile robot uses a reactive type or oscillatory controller, it almost always needs a uniquely designed controller for proper functioning. Therefore, in mobile robot body-only evolution, only certain mechanical parameters are evolved as this enables the reuse of the already available controller with minute variations. Examples of such works are: Inspection robots for space constrained areas (like a duct) designed with a set of modules for power, control, joint and foot [90] (Fig. 3 (a)); *LEGO* robot designs searched by a GA primarily for maximising distance travelled with wheels (Figs. 3 (b) and (c)) [89]; Where the kinematic parameters of a passive humanoid robot were evolved [17].

Virtual creature related evolutions are almost always performed on both body and controller. However, a unique case of morphology-only evolution in virtual creatures is documented in

**Table VII. List of works with morphology only evolution.**

| Author/s | Robot application type | Algorithm | Year |
|---|---|---|---|
| Chedmail *et al.* [16] | Manipulator robot design | GA | 1996 |
| Chung *et al.* [93] | Manipulator robot design | GA | 1996 |
| Chocron *et al.* [48] | Manipulator robot design | GA | 1997 |
| Farritor *et al.* [90] | Inspection robot design for constrained areas | GA | 2001 |
| Shiakolas *et al.* [55] | Manipulator robot design | GA, DE | 2002 |
| Parker *et al.* [89] | LEGO robot for locomotion | GA | 2007 |
| Lipson [63] | 2D robot mechanism | GP | 2008 |
| Smith *et al.* [80] | Legged robot design | GA with Extremal & Pareto Optimisation | 2010 |
| Clark *et al.* [74] | Robot fin design | GA | 2012 |
| Lim *et al.* [73] | Six-legged robot design | SOEA, MOEA | 2015 |
| De Beir *et al.* [72] | Social robot design | GA | 2016 |
| Cruz *et al.* [17] | Mechanical design of humanoid | GA | 2016 |
| Naranjo *et al.* [107] | Design of spherical parallel manipulator | GA | 2018 |
| Simon *et al.* [108] | Mobile robot design | GA | 2018 |

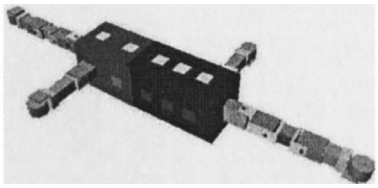
[57]. The authors preferred morphology-only evolution as the goal of the project was to test a new genotype representation. Creatures that moved towards a goal used an innovative *hox* gene inspired strategy for tracking robot ancestors (Fig. 3 (d)). A *hox* gene is the gene type responsible for the body plan along the head-to-tail axis in certain biological organisms.

*2.2.3.2. Co-evolution of morphology and control.*

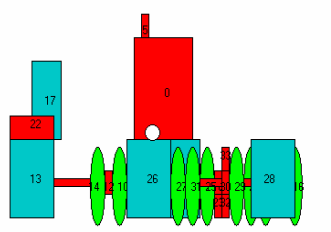
In this section, the research can be primarily classified into the evolution of purely virtual creatures and virtual but physically realisable robots.

*a. Virtual creatures or robots.*

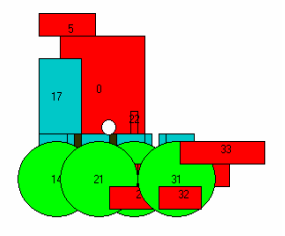
The pivotal paper that reported the evolution of virtual creatures was by Sims in 1994 [18], [98], where the body and brain of creatures for swimming, walking and jumping were evolved. 3D blocks with imaginary joints were allowed to freely develop in simulation as shown in Fig. 3 (e). Subsequent works have been inspired from Sims' virtual creatures e.g. [60], [94], [109].



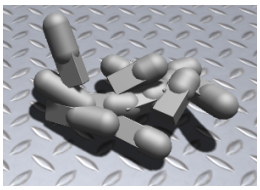
(a) [90]



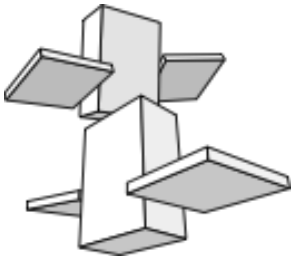
(b) [89]



(c) [89]



(d) [57]



(e)[18]

**Figure 3. Examples of evolved designs.**

Even though the concept of using virtual creatures was introduced in the 1990s, it is still favoured during evolution for several reasons. Considering the computational complexity involved in testing concepts in co-evolution, the use of virtual creatures helps in balancing the hardware and time constraints during evolution. It also allows a quicker software development due to the relatively easier representation of virtual creatures. In recent papers, virtual creatures have been mainly used for testing a proof of concept, for example the concept of morphological plasticity was used in [113] to modify creature morphologies on-the-go. The idea was that, even after the co-evolution process was completed, the virtual creatures could modify their morphology in an online fashion over their lifetime in response to their environment.

*b. Physically realisable robots.*

The papers in the previous section covered creatures or robots with virtually co-evolved body and controller, and they all suffered from a major drawback of not being able to be built physically. As this thesis deals with automatic generation of mobile robots, the literature available on physically realisable robots needs to be closely looked at. Therefore, this section is dedicated to the research that reports buildable robots.

**Table VIII. List of co-evolving creatures or robots.**

| Author/s | Application | Type of algorithm | | Year |
|---|---|---|---|---|
| | | Evolution | Controller | |
| Sims[18] | Creatures for swimming, walking and jumping | GA | ANN | 1994 |
| Lee *et al*. [66] | Mobile robot with obstacle avoidance | GA, GP | Boolean | 1996 |
| Komosiński *et al*. [81] | Creatures for walking and swimming | EA | ANN | 1999 |
| Mautner *et al*. [53] | Mobile robot with obstacle avoidance | GA | ANN | 2000 |
| Pollack *et al.* [79] | Locomotion with linear elements | EA | ANN | 2000 |
| Lee [67] | Mobile robot with obstacle avoidance | GA, GP | ANN | 2000 |
| Endo *et al.* [84] | 2D robot for locomotion and hill climbing | GP | ANN | 2001 |
| Taylor *et al.* [109] | Creatures for swimming, walking and jumping | GA | ANN | 2001 |
| Lund [69] | Line follower robot with *LEGO* | GA, GP | ANN | 2003 |
| Pollack *et al.* [96] | *Genobots* for locomotion | EA | ANN | 2003 |
| Lee [110] | Straight locomotion with obstacle avoidance | GA, GP | ANN | 2003 |
| Endo *et al.* [97] | Humanoid body design | GA | ANN | 2003 |
| Shim *et al.* [75] | Wing structure design | EA | ANN | 2004 |
| O'Kelly *et al.* [62] | Creatures for combat | GA | ANN | 2004 |
| Macinnes *et al.* [65] | Locomotion with *LEGO* | EA | CTRNN | 2004 |
| Miconi *et al.* [94] | Creatures with multiple locomotion modes | GA | ANN | 2005 |
| Lassabe *et al.* [104] | Creatures for multi-surface locomotion | GA | ANN | 2007 |
| Chaumont *et al.* [61] | Creatures for walking/block throwing | GA | ANN | 2007 |
| Parker *et al.* [106] | Sensor position and gait design of mobile robot | GA | Reactive controller | 2007 |
| Chocron [91] | Serial manipulator design | TGA, M/AMEA | Inverse kinematic | 2007 |
| Chocron [92] | Mobile robot design | EA | Open-loop | 2007 |
| Miconi *et al.* [95] | Fighting creatures | GA with LEO | ANN | 2008 |
| Heinen *et al.* [88] | Four-legged walking robot | GA | FSM & ANN | 2009 |

**Table VIII. Continued.**

| Author/s | Application | Type of algorithm | | Year |
|---|---|---|---|---|
| | | Evolution | Controller | |
| Mazzapoida *et al.* [87] | Creatures for irregular surface locomotion | EA | ANN | 2009 |
| Rommerman *et al.* [56] | Walking legged robots | CMA-ES | Oscillatory | 2009 |
| Azarbadegan *et al.* [60] | Biped walking creatures | GA | ANN | 2011 |
| Rubrecht *et al*. [86] | Serial manipulator | GA | CCC | 2011 |
| Gregor *et al.* [64] | Creatures for locomotion | GP | ANN | 2012 |
| Larpin *et al.* [50] | Quadrupedal robot design | MOEA | CPG | 2012 |
| Moore *et al.* [111], [47] | Amphibious robot design | EA | Sinusoidal controller | 2012 |
| Pilat *et al.* [59] | Food consuming creatures | Steady State GA | Recurrent ANN | 2012 |
| Auerbach *et al.* [83] | Creatures for locomotion | CPPN-NEAT | CTRNN | 2013 |
| Samuelsen *et al.* [57] | Robots for locomotion | NSGA-II | CPG | 2013 |
| Risi *et al.* [70] | Robots for walking | NEAT | CPPN | 2013 |
| Lessin *et al.* [58] | Light following/fighting creatures | EA | ESP-ANN | 2014 |
| Digumarti *et al.* [82] | Legged robot design | CMA-ES | Inverted pendulum | 2014 |
| Auerbach *et al.* [7] | Racing/chasing robot design | GA | Recurrent ANN/Hyper-NEAT | 2014 |
| Corucci *et al.* [76] | Underwater robot design | GA | Open-loop control | 2015 |
| Faiña *et al*. [19] | Mobile robot design | Multiple EAs | Sinusoidal control | 2015 |
| Brodbeck *et al*. [78] | Mobile robot design | EA | Amplitude & phase shift based | 2015 |
| Nygaard *et al.* [112] | Mobile robot design | NSGA-II | Amplitude & phase shift based | 2017 |
| Krcah [113] | Mobile robots with dynamic morphologies | Hierarchical NEAT | ANN | 2017 |
| Georgiev *et al.* [114] | Nanorobots for predator-prey pursuit modelling | GA | Reactive controller | 2018 |
| Nygaard *et al.* [115] | Real world evolution of mobile robot | NSGA-II | Inverse kinematic controller | 2018 |

In addition to the area specific discussion of buildable robots, work in this section is further classified in Table IX. The papers are grouped based on: whether they used a fixed robot body design during evolution; if the body was allowed to change, was a fixed library of parts used; if that was the case, were parts sizes allowed to vary. Furthermore, the works are also categorised by: the type of locomotion (wheeled or legged); if sensor feedback was available at the evolved controller; if a full controller evolution was performed or if only controller parameters were evolved; if there was an option for the robots to be evolved for more than one task e.g. multiple waypoint locomotion. The table is particularly important for this thesis as it gives a snapshot of the current status of buildable co-evolved robots that this work seeks to improve.

*Manipulator robots.* Chocron [91] applied a two level GA similar to [48] for evolving the end effector pose and orientation with other physical parameters of a serial manipulator with multiple chromosome genotypes. The Multi-Chromosome Evolutionary Algorithm concentrated some of the robot's features on a single chromosome of floating-point numbers. The variation operations were performed on each chromosome and not globally on the genotype. The paper also tested an Adaptive Multi-Chromosome Evolutionary Algorithm (AMEA) with variation operator parameters modified as per an adaptive selection pressure function. A hard selection pressure was applied when the fitness of solutions exhibited a higher spread or greater standard deviation. The focused and adaptive variation process certainly helped to improve the speed of evolution. However, these methods are seldom applied directly in other works. Nevertheless, some of the concepts like the graph or tree genotype representations can be considered to be similar to, but better than, the multiple chromosome genotype implemented in this work.

Another serial manipulator, for highly constrained space like the inside of a tunnel boring machine is elaborated in [86]. Each robot was made of segments which comprised of links with revolute or prismatic joints with a maximum of one-DOF (degree of freedom). Like almost all the works in this section, robots in these papers did not have any sensory feedback, which ultimately led to an open-loop control of the available actuators. A minimising fitness function was applied that consisted of aspects for comparing the actual end effector trajectory with expected trajectory, the

number of DOFs, the robot length and the number of collisions per link in every episode. Nonetheless, how each of these were factored in the actual calculation was not explained and the methods used for calculating the error in trajectories were also not stated.

*Mobile robots with wheels.* In one of the earliest works that reported buildable robots, Lee *et al.* [66] maintained a fixed robot shape while evolving the structural parameters of a mobile robot for locomotion with obstacle avoidance. The robots used 8 IR (infrared) sensors to maintain a safe distance from obstacles. The evolver used AND, OR and comparison operators (>, <) to build a suitable controller. Though this type of a controller can be considered as an oversimplified version of an ANN, jerk-free or smooth robot movements were not possible due to the Boolean control of the two motors. Additionally, as the only evolved structural parameters were the body size, wheel-base and wheel radii, the main task of the evolver was only to generate a suitable controller. The advantage of such an approach is that the mechanical size constraints applied will always ensure that a physically buildable robot is available without a controller. Furthermore, the authors justified their approach by discussing the importance of a suitable body-controller coupling. Despite their efforts, they could not clearly justify the benefits of co-evolution. From a different perspective, it can be safely stated that, as the robot mechanical parameters change, a controller modification becomes inevitable. So rather than co-evolving body size and controller, a manual controller-only design should have been sufficient in this case.

For fitness evaluation, Lee *et al.* used two types of penalties, one for maintaining a safe distance from obstacles and the other on the number of times the robot hit an obstacle. Both were accumulated over the simulation period. Consequently, navigation was an indirect result of the obstacle avoidance behaviour. Using the same system, Lee [67], [110] measured a cumulative fitness at each time-step with a final fitness calculated at the end of simulation while evolving robots for obstacle avoidance and box pushing operations. Fitness at each time-step involved a weighted addition of IR sensor reading, forward speed and rotation speed. Though the concepts applied were repeated in the three works, they did not discuss the intuition behind how and why each of those factors were considered or how the weighted

addition of time-step fitness was performed. Ultimately, these later papers added very little to the conclusions from the former.

Sensor feedback use in the controller of co-evolved robots was first demonstrated when Lund [69] evolved a line follower robot using *LEGO Mindstorms.* The author fixed the morphological shape and allowed the evolution of all other parameters. But again, consistent with most other works discussed in this section, just the parameters (like the wheel size) were evolved in the body. However, the key difference here was the use of sensors. This was forced by permanently fixing two sensors on the robot and using a fixed linear perceptron type NN whose weights were evolved. To evolve these robots, the fitness function was comprised of factors that checked if the robot was on the line and whether it was moving in the right direction. Though the authors said that the fitness calculation equation was decided based on empirical tests, how each factor affected evolution was not demonstrated or discussed.

*Mobile robots with legs.* The morphology and walking pattern of a six-legged robot for space missions were simultaneously optimised by Rommerman *et al.* [56] in the MARS (Machina Arte Robotum Simulans) simulation platform. The robot was simulated in three scenarios during its evaluation. The fixed topology of the legged robot used is shown in Fig. 4 (a). Similar to Lee *et al.* [66], the mechanical parameters evolved included different body part lengths and the centre of mass. But instead of evolving a full controller, just the walking pattern parameters of an oscillatory controller were evolved. This meant both body and controller were optimised instead of co-evolving from scratch as in true automatic robot generation. Furthermore, sensor feedback used in evolution was only for fitness evaluation and not for actual robot control, effectively resulting in open-loop control. For the walking robot, the fitness function was made of four factors: on energy efficiency; load on joints; to check if any part other than the legs was touching the ground; and the percentage of time the robot's centre of mass was outside the stable region. Another difference when compared to other papers was how a negative fitness was applied with the maximum fitness tending to zero.

Similar to the six-legged robot [56], Chocron evolved a quadrupedal robot using modular components for rough terrain exploration [92]. The legged robot was

controlled with an open-loop sinusoidal controller. So, only the oscillator's parameters were evolved in the controller side. While a matrix-based representation of the genotype was used, its dimensions increased very quickly as the number of joints on the robots increased. Though the author used it for a compact representation of the robot, it is felt that such a representation would have created an unnecessary computational burden during genotype-phenotype mapping. Such problems could have been avoided by using other representations like tree, or generative encoding. Furthermore, the author also believed that fitness function design could only be addressed by experimentation. Though the paper reported how the fitness calculation was performed after comparing the candidate performance with a manually designed solution, any further details about this process were missing.

Another walking robot evolution is explored in [57] which used a *distance to goal* based fitness function. To incorporate multiple goals, the robot was evaluated based on whether it reached the first goal and second goal after that. However, it was evident from the results that the best robots were not even able to reach the first goal. Therefore, new fitness calculations for the second goal were never applied.

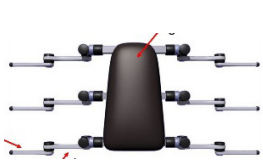
The concepts of open-loop control and simple distance-based fitness calculation appeared again during the mobile robot evolution in [97]. A number of variables of a fixed topology biped humanoid robot along with its controller for gait design were evolved. The controller evolution involved parameters of a fixed NN containing a sinusoidal oscillator. A notable innovation in this work was in the way servo actuators were controlled. Instead of the NN directly controlling the actuator, the output of the NN was the desired trajectory corresponding to each robot joint. This trajectory was then fed to a PD controller to regulate motor torque. Though this two-level control approach has its advantages, it is not commonly applied. A probable reason could be the limited range of applications reported in co-evolutionary robotics. In evolution, it suits applications where precise control of actuators is necessary. An example is when using an inverse pendulum type controller for humanoid locomotion. Here, subtle variations to the control signals will be required to compensate for the dynamic changes occurring during locomotion.

Another example of the two-level control approach is [82], where the body dimensions and control parameters of a four-legged robot for optimal speed and predefined gait were evolved. Multiple factors were considered during the fitness calculation for gait optimisation. One part was a penalty function while the other encouraged the robot to accelerate to a set speed and maintain it after a certain number of strides. Penalties were imposed to ensure smooth torque at joints, to avoid limb collisions and to ensure power efficiency. Similar to several works in this section, the details on how each of these factors were calculated were missing.

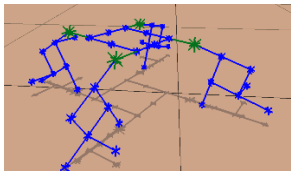
Among recent works, Nygaard *et al*. evolved the structural parameters of a six-legged mobile robot along with corresponding control parameters [112]. An amplitude and phase shift controller connected to servo motors in the legs generated an oscillatory gait. The uniqueness of this work was how a two-stage evolution process was used to increase performance by 10% when compared to a single stage evolver. Co-evolution occurred in the first stage where 256 individuals co-evolved for 1024 generations after which three robots were chosen for controller-only evolution. As the evolution was done in two stages, a chosen few were physically built, and a controller-only evolution was performed for generating robots that could walk as fast as possible. Further details on how the evaluation was performed were unavailable.

The concept of real-world evolution was again used in [115] to arrive at satisfactory leg lengths of a mobile robot. A physical mechanism allowed multiple leg lengths to be set automatically directed by the evolver. As with the previous case, precise details of how the gait evaluation was performed seemed to be missing.

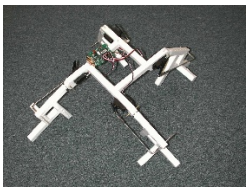
*Flying robots.* A unique application for the two-level control approach can be seen in the design of wing structure (Fig. 4 (g)) and the low-level controllers of a robot for flying along a straight line [75]. An evolved sinusoidal NN signal was connected to a PD controller for flapping wings. The duty of the evolver was to add new joints to the wings of the robot and make subsequent changes to the controller. While the authors believed that the combination of evolution with flying robot design would



(a) [56]

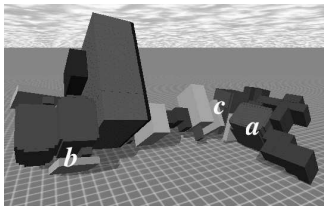


(b) [71], [96], [116]

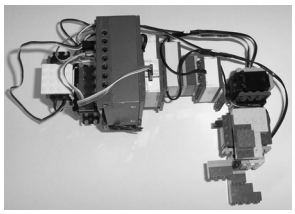


(c) [71], [96], [116]

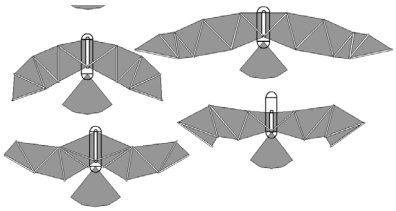
(d) [79]



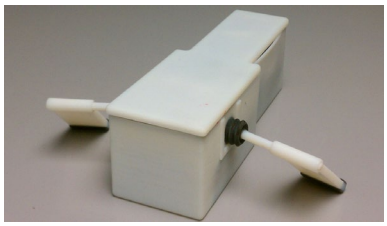

(e) [65]



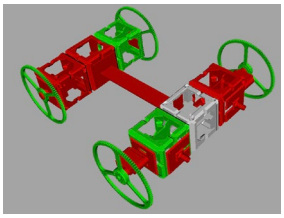
(f) [65]



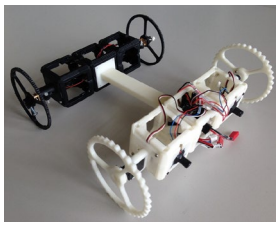
(g) [75]



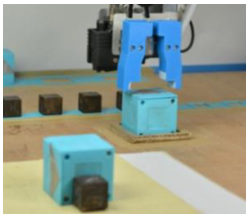
(h) [47]



(i) [7]



(j) [7]



(k) [78]

**Figure 4. Examples of co-evolved designs.**

take the area to new frontiers, even after 15 years since it was published, progress is still stagnant. This is perhaps due to the lack of proper sensor feedback use which is yet to be demonstrated in evolved robots and which would be of absolute importance in flying robots. Furthermore, an uncommon approach to fitness calculation was applied in this paper. The overall fitness of each robot depended on the amount of time it was airborne. Additionally, the fitness calculation function was modified over the evolution process depending on the performance of the entire population. This was accomplished by a variable that only allowed changes to fitness after a set airborne time was reached. Therefore, as generations progressed, it became tougher

for robots to attain a better fitness value. The problem here could be that there was a chance for the fitness to reduce over the process and negatively influence the evolver. However, this could not be verified due to the unavailability of fitness performance curves.

*Underwater robots.* Authors of [47], [111] evolved mechanical parameters of two sweeping arms along with the sinusoidal controller parameters of an amphibious robot. The final model was created with rapid prototyping to confirm the simulation (Fig. 4 (h)). However, poor modelling of underwater conditions resulted in significant differences between the behaviours of real and simulated robots. The papers reported evolution with two different types of fitness functions. One was just the distance travelled in 10 s. The other was essentially the square of the sum of distance travelled on land and in water. The paper claimed that the second fitness limit was always between 0 and 16 [47]. This would only have been possible if the distances travelled were limited to less than 2 m in each case.

Along similar lines as [47], existing design parameters of an underwater robot *PoseiDRONE* were modified using a *novelty search-based* GA by Corucci *et al.* [76]. They showed how an objective based fitness function was better for evolving faster robots while the novelty search based GA was better at inducing new behaviours in evolved robots even when these robots never reached the goal. Additionally, a human designer manually selected *interesting* individuals during the run in evolving successful designs, although the criteria used for selection were not explained. Eventually it meant a strong bias was introduced to the evolver.

Corucci *et al.* also implemented a two-part fitness calculation but with a penalty function and a negatively allocated fitness based on distance travelled. The distance-based fitness awarded was the ratio of distance travelled along the x axis to the total length of the robot. The penalty function involved a fixed punishment for falling robots and a variable penalty that depended on the period the robot was unable to balance itself. However, the reasoning behind each of those decisions was not explained. Particularly, why only the movement along the x axis was considered in fitness computation is unclear.

*Mobile robots with complex modes of locomotion.* An efficient genotype representation when compared to [92], especially for robot morphology is generative encoding. It was applied to create *Genobots* with bars and joints (actuated and non-actuated) for locomotion [71], [96], [116] (Figs. 4 (b) and (c)). The same research group later evolved robots with linear bars and linear actuators through the GOLEM project, and with the help of rapid prototyping tested the evolved prototypes (Fig. 4 (d)) [79]. These papers applied a minimal fitness calculation where fitness was the distance travelled at the end of simulation.

Almost 10 years after the first reported co-evolving virtual creatures, the work from Macinnes *et al.* [65] can be considered as the next landmark in co-evolutionary robotics. Their main contribution was how robots were built for locomotion from a library of pre-existing parts comprised of *LEGO* bricks, instead of co-evolving robots with fixed mechanical structure with sensors. Servo motors and position feedback sensors were used in the assembly of the final robot after evolution (example robot is shown in Figs. 4 (e) and (f)). The fitness evaluation involved measuring the least distance travelled in a fixed time by a single block from all of the blocks that constituted the robot. Though the paper claimed the use of feedback sensors in their robots, how that feedback made a difference in the control of the robots was not demonstrated. Furthermore, the oscillatory locomotion of the robot could be the result of the Recurrent Neural Network (RNN) type of controller used without sensor use. In addition to these drawbacks, not a lot of progress has been made in co-evolutionary robotics since this paper.

The next notable work is [7], which presented a new software called *RoboGen* where the concept used by [65] was applied for co-evolution. *RoboGen* is an open-source platform which generates robots for racing or chasing activities with simple distance-based fitness functions from a set of standard parts including servo motor actuator, IR, light and IMU (Inertia Measurement Unit) sensors and an *Arduino* controller. The package is capable of evolving morphology and controller, and generates 3D printable models which can be physically assembled. A virtual evolved mobile robot and corresponding 3D printed model is shown in Figs. 4 (i) and (j), respectively. A detailed study of this platform is in *Section 2.3*.

Another similar work was by Faiña *et al.* [19] where a system for applications involving a list of subtasks such as painting, carrying and cleaning executed through a variety of locomotion modes like climbing, walking, rolling and crawling was developed. The package called *EDHMoR* [117] has a core module containing encoder and accelerometer. There are four possible options for actuators or special sensors that can be connected on empty slots of the core module. A two-part type fitness calculation was applied for the two demonstrated tasks. One part was based on goal attainment which was simply the distance travelled and the other was a reward awarded based on a minimum threshold attainment. Again, as in many of the papers above, the authors did not provide any more details on how exactly the fitness evaluation function was mathematically computed.

In most of the works previously described, the evolution was performed in a software platform, and the best designs were implemented in reality. However, in the paper by Brodbeck *et al.* [78], a 6-DOF serial mother robot built the solution population and tested their speed of locomotion and created better solutions with the help of an EA. The population was constructed from active and passive modules. Servo motor, *Bluetooth* module and *Arduino* controller were used in the module while the evolution process was offloaded to *MATLAB* running on a PC which was interfaced with the mother robot. The end-effector of the mother robot while building robots is shown in Fig. 4 (k). Moving away from directly using distance in the fitness calculation, Brodbeck *et al.* calculated the average speed of each robot with the help of a camera.

In a step further, Weel *et al.* [49] explained a futuristic concept of online co-evolution as a *proof-of-concept* in simulation with no central evolution process involved. The hypothesis was that robots could self-reproduce, and grow bodies and brains online with no human intervention. Several works reported the use of this concept in co-evolving modular robots with an emphasis on evolving a controller for a new-born robot e.g. [118], [119]. The authors felt this was a very promising area to study with a number of open questions.

**Table IX. Buildable co-evolved robots.**

| Author/s | Fixed design | Fixed parts library | Fixed part shape with variable size | Locomotion (Wheeled(W)/ Legged(L)) | Sensor feedback used during evolution | Controller parameter only evolution | Subtasks |
|---|---|---|---|---|---|---|---|
| Lee *et al.* [66] | ✓ | | | W | ✓ | | ✓ |
| Pollack *et al.* [79] | | ✓ | ✓ | L | | | |
| Lee [67] | ✓ | | | W | ✓ | | ✓ |
| Lund [69] | ✓ | | | W | ✓ | | ✓ |
| Pollack *et al.* [71], [96], [116] | | ✓ | ✓ | L | | | |
| Lee [110] | ✓ | | | W | ✓ | | ✓ |
| Chocron [92] | | ✓ | | W&/L | | ✓ | ✓ |
| Endo *et al.* [97] | ✓ | | | L | ✓ | ✓ | |
| Shim *et al.* [75] | | ✓ | ✓ | Winged | ✓ | ✓ | |
| Macinnes *et al*. [65] | | ✓ | | L | ✓ | | |
| Rommerman *et al.* [56] | ✓ | | | L | | ✓ | |
| Rubrecht *et al*. [86] | | ✓ | ✓ | ^ | ✓ | | |
| Moore *et al*. [111], [47] | ✓ | | | L[$] | | ✓ | |
| Samuelsen *et al.* [57] | | ✓ | ✓ | L | | ✓ | |
| Digumarti *et al.* [82] | ✓ | | | L | | ✓ | |
| Auerbach *et al.* [7] | | ✓ | ✓ | W&/L | ✓ | | ✓ |
| Corucci *et al.* [76] | ✓ | | | L[$] | | ✓ | |
| Faiña *et al.* [19] | | ✓ | | L | | ✓ | ✓ |
| Brodbeck *et al.* [78] | | ✓ | | L | ✓ | ✓ | |
| Nygaard *et al*. [112] | ✓ | | | L | | ✓ | |
| Nygaard *et al.* [115] | ✓ | | | L | | ✓ | |

[$] Underwater fin based.

^ Not a mobile robot.

For successful evolution, the literature has shown that the number of variables evolved needs to be kept to a minimum. Fixing the mechanical structural shape and only varying certain parameters is such an option. This means the genotype to phenotype mapping need not alter with the evolver as only a few numbers are modified. Over several generations, these targeted changes can lead to faster convergence to a satisfactory solution. Additionally, this would allow for the faster computation of results as the rendering of virtual simulations would also be faster as very few parameters are changed in every step. This can be seen in the majority of works in Table IX that have chosen to fix the mechanical shape during evolution.

The next step for further improvement, that also increases the computational complexity, could be to use a library of parts and allow the evolver to mix and match them which could potentially make every evolved robot visibly different. This additional step would also mean that genotype-phenotype mappings can keep changing over the entire evolution epoch. On the other hand, from the evolver's perspective, variation operations will only be required on a few variables, as before.

On the controller side, as previously stated, successful evolutions are often demonstrated when the evolver is in charge of modifying as little as possible. In this regard, controller parameters only change due to variations in morphology. This can also be observed in the majority of the works. When sensor use was available during evolution, it was mainly demonstrated while using fixed robot designs, with a few exceptions.

For truly enabling the use of co-evolved robots in real applications, both morphology and controller have to freely evolve while simultaneously integrating sensor feedback. However, none of the works in Table IX demonstrate such capabilities. The most advanced from the list is [7], which promises the evolution of a full controller with sensors while using a library of parts, with the option of modifying the size of one part type during variation operations. Therefore, [7] was chosen as the starting point for improving ER in this thesis.

During the process of robot co-evolution, each of the created robots is tested with a fitness function that decides how well it is able to accomplish the task and decides whether it is worthy to be moved to the next phase of evolution. Consequently, if the fitness is not an effective indicator of task attainment, the evolution process can deviate from the shortest

path to the solution. However, another key observation from the works above is how fitness function design is not being given sufficient weight as the focus is on the rest of the aspects of evolution. Most of the works used very simplified methods of fitness evaluation, even when the success of the entire evolution process depended largely on how task requirements are conveyed to the evolver. Furthermore, ad hoc methods employed with limited details reported meant it was not possible to compare the performance of each of those fitness functions.

Moving away from this methodology, literature on controller-only evolution in robots has shown how multiple approaches have been implemented to develop fitness functions including aggregate, competitive, environmental, behavioural, incremental, tailored and training data-based fitness functions [123]. Hence, the work in *Chapter 4* covers fitness function design in an effort to address some of the drawbacks encountered while co-evolving mobile robots.

### 2.2.4. The Reality Gap

The discussion of ER cannot be completed without touching upon one of the main factors that is preventing the area reaching its full potential. Referred to as the *reality gap* [120], it is the difference in performance of physically built and corresponding virtual evolved robots. This performance difference often makes physical robots unsuitable for the intended tasks. The main culprit contributing to the reality gap is the simulator used in fitness evaluation.

During fitness evaluation, the individually designed fitness function is normally applied to the behaviour of candidates in a simulated environment. A poorly designed surface could mean real conditions are not mimicked. The task specific simulation environments for mobile creatures were flat surfaces, except in a few cases where they were curved [56], uneven or stepped [104]. Furthermore, though not absolutely necessary, it is notable that none of the reported works attempted to use a dynamically changing environment. The only reported partial dynamic aspect of a simulation was an automatic removal of consumed food [59].

Quality of the simulation setup plays an important role in deciding if the evolved individual can perform the same task in reality as in a simulation. This means software package

**Table X. Evaluation platforms.**

| Platform | Citations |
|---|---|
| Webots | [50], [73] |
| MARS | [56] |
| Ella | [64] |
| ODE | [47], [61], [62], [65], [70], [74], [75], [82], [87] |
| CimStation | [16] |
| Gazebo | [19] |
| MATLAB | [55], [76], [78] |
| PhysX | [57], [58] |
| GOLEM | [79], [96] |
| ORCOS | [86] |
| Custom six modules | [89] |
| MathEngine | [109] |
| LeGen* | [88] |
| Breve* | [104] |
| RoboGen* | [7] |
| YAKS | [51] |
| MASS^ | [59] |
| EDHMoR | [19] |
| FEM based | [55], [81] |
| Custom C++ based | [92] |
| ParadisEO with custom framework | [112] |

*Based on ODE

^Based on PhysX

limitations also contribute to the reality gap, although the exact effect of how each package affects the reality gap is unclear from reported papers. The software packages developed have been individually built from scratch or using easily available packages such as *MATLAB* or *PhysX* or *Open Dynamics Engine (ODE)* (Table X).

Furthermore, even though the reality gap is a well-known concern in ER, only limited papers in the area of co-evolution shed light on the problems faced during physically building and testing robots. Moore *et al.* [47] experienced difficulties in the physical validation of robots owing to poor modelling of mechanical elements such as servo motor joints, and an inability

to simulate physical conditions. The difference in on-board controller timing and simulator timing resulted in a drastic difference of robot speeds. The speeds recorded were 55 cm per minute and 14 cm per minute in virtual and real systems, respectively [65]. The inconsistency of 3D printing added towards a notable reality gap in [74]. Further, *numerical explosion* due to inaccuracy of the simulator environment added errors into the numerical calculations which accumulated and resulted in evolving solutions with unreasonable fitness [61].

To counter the reality gap and to accommodate the uncertainty of physical systems, a common method is to introduce random noise into the measurements [7], [65]. For testing controller robustness, Bongard [54] performed damage testing by disabling sensors on the robot. Lee *et al.* [66] proposed using a training set with multiple starting positions and incorporating cumulative fitness to ensure robustness. A simulator tool called *BROOKS* was recently released, specifically for ER simulations, that claims to produce almost identical results when compared to reality [121]. However, due to lack of data for comparison, the claim could not be verified. In a distinctive approach to building simulation platforms for countering the reality gap, [69] and [106] converted the behaviour of a robot in reality to a virtual environment. To almost completely get rid of the problem, an online driven evaluation process was tested in [78].

### 2.2.5. Discussion

In this section, the general principle of EAs along with the various methods applied for evolving body morphology and controller were explored. A brief explanation of the algorithms for controller development and how they are applied in different scenarios was also covered in the above sections. Along with this, EA applications in areas outside robotics, a discussion on buildable robots, and the problem of the reality gap were also included.

The above review demonstrated where the area of co-evolution has reached with over 25 years of research. A common feature in all of the above domains is how the same ideas have been applied in different applications. The differences among them lie in how each of the individual components of an EA (discussed in *Section 2.2.1*) are fixed. For instance, a surface mount device assembly machine brought forward by Philips Industrial Electronics

Corporation, that was much better than the state of the art used a GA with binary coded genotype [29]. The same type of representation can be seen in the evolution of the gait of a humanoid robot [17]. The difference between them is in what each of the binary numbers actually represent. Therefore, from a functional perspective, low level details will look very similar. This enables porting of core EC ideas between fields.

In the quarter century that EC techniques having found applications in robotics, more than 80% of the work reported is post-2000, which is probably due to the advent of high-speed computing systems. This is because optimisation methods are computationally expensive due to the stochastic exploration methods applied. The core topology of evolved robots remained fixed probably also due to the same reason.

While many researchers applied EAs in robotics as a proof-of-concept, there were only a handful who tested the evolved robots in reality. Usually, when the design was allowed to evolve freely, the ability to physically realise it appeared to be compromised. When the aim was to build physical robots, the evolution process was confined to the selection of parts from a predefined set, except in a few cases. However, in such situations, any sensory feedback in the system seemed to be missing. Additionally, even when sensors were present, their utilisation could not be verified due to the NN type controllers in evolved robots.

In buildable systems, the general trend of an application area was towards locomotion and serial manipulation. The cause for such constrained applications might be due to:

- the long time required for evolution on the current general-purpose computing systems, even for slightly more complex tasks,
- the reality gap between simulation and real systems,
- difficulty in designing effective fitness functions, and
- methodological problems such as biasing and premature convergence.

The effect of the reality gap cannot be stressed enough in its contribution to discouraging physical realisation. Even though there have been several reported works addressing the reality gap in ER, they mainly focussed on controller-only evolution. This suggests the need for further research specific to the co-evolution process.

The review also suggests the need for software packages integrating multiple areas covering a physics simulator, CAD modelling, controller development and an evolver. Currently only an interdisciplinary team can take full advantage of ER. This highlights an immediate need for developing a stable high-level ER environment. There is also a need to consolidate the generated knowledge and develop standards including benchmarking methods to aid transferability and to move away from ad hoc practices. Moreover, formal conventional methods for morphology and controller designs guarantee convergence and discovery of suitable solutions. All these reasons collectively could be contributing to the unpopularity of ER in robotics or discouraging roboticists from delving into the area. However, it must not be forgotten that humans are a result of evolution by natural selection and current technology is yet to replicate from the traditional approaches the effectiveness and complexity exhibited by biology.

Through the survey reported above, it is evident that the field of ER is yet to generate robots capable of performing complex tasks. Despite 25 years of development, the evolved robots are only capable of simple locomotion tasks with an oversimplified controller where the robot morphology and controller are co-evolved and there has been only a handful of systems that can generate real physical robots. Nichele estimated that more than 95% of the literature focussed on the evolution of robot controllers. In the rest, less than 1% of the reported works physically tested the generated morphology and controller and the remaining 4% only simulated the co-evolution process [122]. Furthermore, from the viewpoint of co-evolution of mobile robots, there are still a number of concepts that are yet to be tested which are based on different methods implemented in the above areas. Consequently, this thesis examines two new approaches in co-evolution that could be considered to be inspired from research in other areas. The decision-making process involved while designing each of these is also explained in respective sections.

## 2.3. Introduction to *RoboGen*

There is a demonstrable need for a stable multidisciplinary software package for evolution that uses mechatronics concepts of mechanical design with feedback control, including 3D dynamic simulations while applying computing concepts. However, the development of this was not attempted as it is a program of study in itself. The approach chosen was to use an

available package and build on it. When the project began, only two fully-fledged co-evolution platforms existed, *RoboGen* [7] and *EDHMOR* [117].

Developed in 2014, *RoboGen* is an evolution platform that can evolve mobile robots for primitive locomotion tasks. It is a package capable of handling the co-evolution process of generating complete virtual robots. It runs an evolution engine and simulation engine side-by-side with data transferred between them multiple times during the evolution process. The evolution engine performs the primary steps involved in the evolution process and the simulation engine estimates the performance of each evolved individual through a fitness function.

*EDHMOR* is a reconfigurable modular robot evolver that uses a sinusoidal controller in its robots. For evolution, it employs Java Evolutionary Algorithm Framework (JEAF) which allows the use of multiple EAs with tree-based genotype representations, and a simple distance-based fitness function to evaluate its robots in a *Gazebo*-based physics simulator. Since the scope of this project is not in reconfigurable robot evolution, it was not chosen. *RoboGen* was preferred as its source code was open source, it used ANN in controllers (which is shown to have more potential than pure sinusoidal controllers), it offered features like the availability of parametric joints for evolution, the option of customised fitness function design, and the possibility of web browser-based evolution and as its simulator was based on the commonly used *ODE*. This section provides a brief introduction to *RoboGen* with an emphasis on its underlying functional operations to help in understanding the enhancements performed on it as presented in subsequent chapters.

*RoboGen* evolves robots from a list of available parts, namely: a core component brick that houses an IMU, controller and battery, shown in Fig. 5 (a); a fixed brick that looks the same as the core component (Fig. 5 (a)); a parametric bar joint with variables to configure the arm length and tilt angle (Fig. 5 (b)); an active servo motor driven joint (Fig. 5 (c)); a passive hinge (Fig. 5 (d)); a light sensor (Fig. 5 (e)) and; an IR based distance sensor (Fig. 5 (f)). The core and fixed brick, which are about 5 cm in size, can connect to up to four parts on their vertical faces while all other parts only allow connections on two sides.

As per input parameters, *RoboGen* evolves robots and each robot is evaluated individually with the help of a physics-based simulator. 3D printable robot part files and controller code


(a) Core brick/Fixed brick (b) Parametric joint

(c) Servo motor joint (d) Passive joint

(e) Light sensor (f) Infrared sensor

**Figure 5. Parts available in *RoboGen*[7].**

are also generated to allow physical testing of robots. An example robot evolved in *RoboGen* is shown in Fig. 6. The robot in the figure has 2 active joints that oscillate to generate a resultant motion.

The EA used to evolve morphology is a GA with a tree-based representation of the phenotype. The GA works by randomly initialising a fixed population of parents (μ) and evaluating them through a fitness function specific to the application. After the fitness evaluation process is completed, the population is randomly divided into λ groups of two and the best individual in each group is chosen as a parent using a deterministic tournament strategy. Later, various mutation operators are applied on these selected parents according to the set probabilities and parameters in the evolution configuration file. As part of mutation, operations such as addition and deletion of parts, modification of parametric

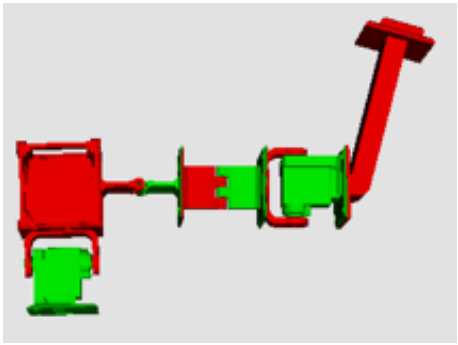


**Figure 6. An example robot from *RoboGen*.** *It has a core component (red block to the left) with a servo motor joint (in green) on one side. The adjacent side of the core component is connected to a passive joint. This is followed by another passive joint, a servo motor joint, a parametric joint and an IR sensor on the extreme right-hand end. Robot locomotion is accomplished by two servo motor joint oscillations.*

variables, duplication, swapping and removal of sub-trees are performed on the robot tree. The mutated children are then added to the population and the entire population is ranked according to their fitness. The best μ individuals are then retained while the rest are deleted from the population. This method is a (μ+λ) evolution strategy where μ is the parent size and λ is the number of children [6].

A similar process is also performed on the neural network controller to evolve a controller for each evolved body. The kind of neural network used is similar to a Recurrent Neural Network (RNN). Here the ANN composed of three layers is made of several input, hidden and output neurons. Every neuron in the network is connected to every other neuron and there are also feedback connections for all except the input neurons as seen in Fig. 7. This is unlike a typical RNN where there is only self-feedback at every neuron and not feedback to every other neuron.

The effectiveness of the ANN used is further improved by the availability of oscillatory neurons. These have previously been demonstrated to be better than a standard ANN controller in the co-evolution process to evolve controllers of robots [124]. These oscillatory neurons can be added to any of the available layers to induce an oscillatory signal at the output. These oscillators acting as signal generators along with inputs from sensors combine while generating motor control signals.

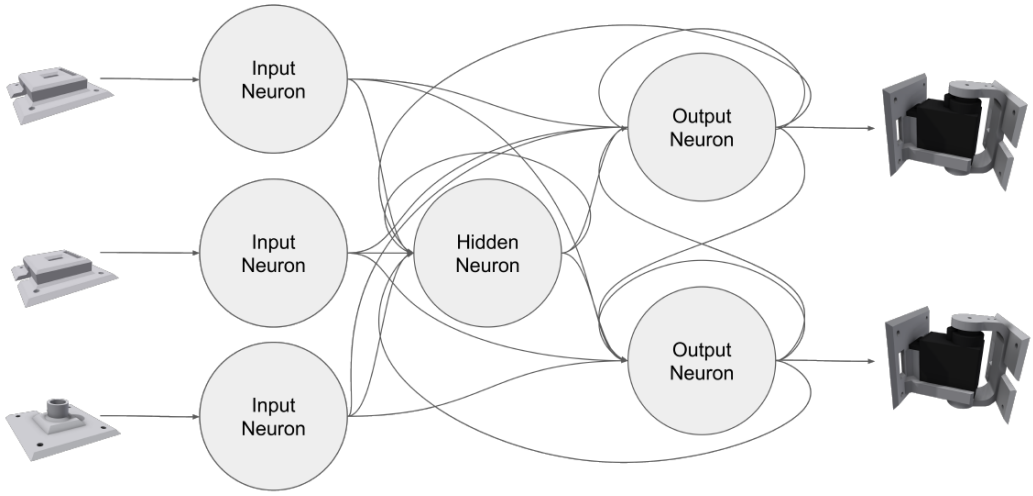



**Figure 7. An example of ANN controller in *RoboGen*[7].** *It can have up to three layers based on evolution parameters. Except for input and oscillatory neurons, every neuron is connected to every other neuron.*

During the evolution process, multiple constraints are also applied on each of the robots to confirm their physical buildability. Robots whose parts intersect with each other are discarded, including the only core part, as there can be only a single controller. The maximum I/O port requirements of the controller board allow up to three sensors and eight motors during the evolution process.

After the evolution process ends, the package can also generate CAD designs for 3D printing and controller code compatible with an *Arduino Leonardo* microcontroller board. The assembled robots can then be tested for various tasks as set during evolution.

A number of parameters are also set at the beginning of evolution. They are mainly for the evolver and simulator. The main evolution parameters set are population size, number of children generated, population update strategy, various probabilities and bounds for mutation and crossover, bounds of period, phase and amplitude of oscillator, neuron bias and weights. The simulation parameters mainly involve details regarding the fitness evaluation time, fitness function, and other parameters needed for building of the virtual 3D simulator.

See Fig. 8 for the complete *RoboGen* architecture. While this architecture is applied in *Chapters 3* and *4*, a part of *Chapter 5* implements a different feedforward NN architecture

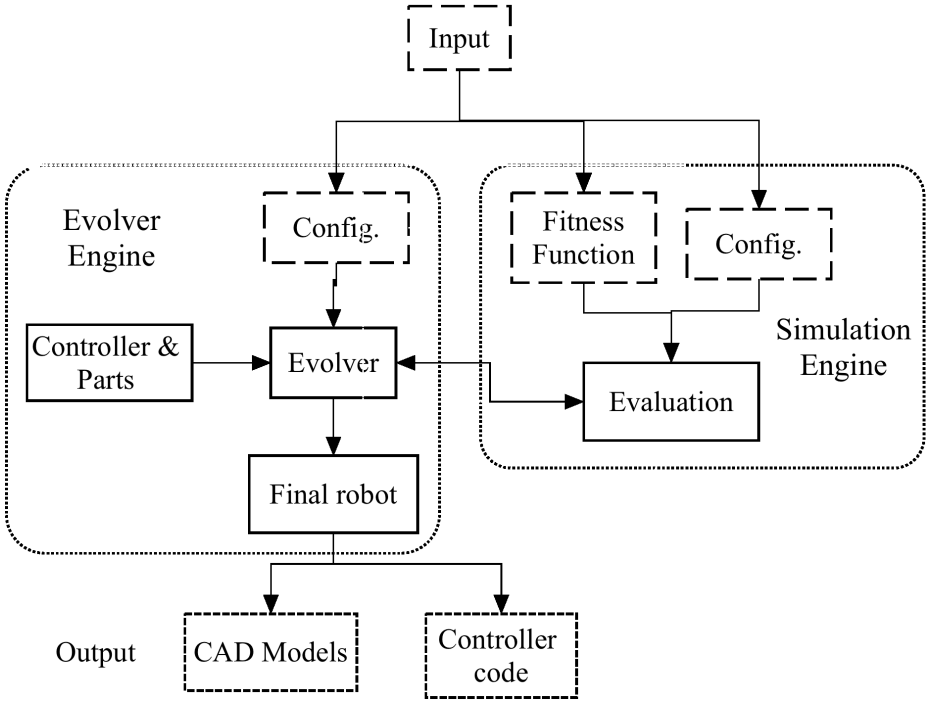



**Figure 8. *RoboGen* architecture.** *Evolver engine and Simulation engine run in parallel with data transfers between both processes through Google Protocol Buffer [125] protocol. The inputs are the evolver configuration and simulation configuration with environment parameters, obstacle data and fitness function. The output contains evolved robots with details on CAD design and ANN. A separate script can convert these into 3D printer compatible models and Arduino compatible controller code.*

as explained in *Section 5.1*. Except in *Chapter 3*, all experiments are performed on a High Performance Computing (HPC) platform comprising 50 nodes and 20 cores per node. This allows for parallelising the fitness evaluation of every robot in *Chapter 4* and learning and evaluation of every robot in *Chapter 5*. Each robot is allocated a core, and so evaluation of an entire population of robots happens simultaneously in parallel on a number of cores on the HPC.

## 2.4. Path Planning Algorithms

The main focus of this thesis is to improve the co-evolution process. To assess the concepts, mobile robots are evolved for reaching a goal while performing obstacle avoidance. This desired robot behaviour can be classed under path planning in mobile robotics. Moreover, robots in *Chapter 4* are mainly built using an established path planning algorithm. Therefore, this section is devoted to discussing an overview of path planning algorithms used in mobile robotics.

Path planning is an important step performed by autonomous mobile robots to navigate from a starting position to an end position. Normally, based on sensory feedback and sometimes with stored map data, the robots perceive their location in real-time and make necessary calculations to control the actuators for reaching the goal. Path planning algorithms can be classified in five groups namely: sampling-based; node-based optimal; mathematical model-based; bioinspired; and multi-fusion based algorithms [126]. Each of them is explained below.

### 2.4.1. Sampling-Based Algorithms

These algorithms require a mathematical model of the environment. The sampling process takes place by dividing the map into nodes or cells to find a suitable route. The class is further divided into active and passive algorithms where the former are capable of finding the optimal route from a number of routes and the latter cannot independently do the same. Examples of active algorithms are Rapidly Exploring Random Trees (RRTs), Dynamic Domain RRT (DDRRT), RRT-star and Artificial Potential Methods. Voronoi, Probabilistic Road Map (PRM), k-PRM, s-PRM, k-s-PRM and Rapidly Exploring Random Graph (RRG) are examples of passive sampling-based algorithms.

#### *2.4.1.1. RRT*

Introduced by LaValle [127], RRT operates by first randomly choosing a node which is within a predefined distance from the initial node. The process continues until the node falls in the obstacle space or when no new node is available within the predefined reachable distance. From there, based on a control factor and a cost function, a new node is found, and this cycle runs until the goal is reached. Due to the random nature of exploration, optimal convergence is not guaranteed. DDRRT is a variation to address the shortcomings of the basic RRT [128]. Instead of randomly choosing a node in RRT, DDRRT works by selecting nodes that are closest in the reachable region defined by an imaginary sphere. While it converges faster than RRT, it is seldom optimal [126]. For ensuring asymptotic optimality, RRG [129] connects to all possible nodes in the reachable region simultaneously. A much faster algorithm is RRT-star which also factors in the cost incurred while taking every simulation step and eliminates routes taken that are costly during node selection [129].

#### *2.4.1.2. PRM*

PRM performs navigation in two major steps. At first, a road map is built by connecting *k* neighbours (in k-PRM), all neighbours in a pre-set distance (in s-PRM) or by a combination of both (k-s-PRM). After the road map is built, a node-based search algorithm such as Dijkstra or A-star (discussed in next section) is used to find the optimal path [130].

#### *2.4.1.3. Voronoi*

The entire map is converted into a Voronoi diagram where each point on the line segments (referred to as a Voronoi channel) is equidistant from two neighbouring obstacles [131]. Then a search algorithm such as A-star is implemented to find the shortest path to the goal. The main problem with this method is the risk of non-convergence.

#### *2.4.1.4. Artificial potential*

Here an attractive potential function is allocated to the goal and a repulsive potential to the obstacles in the first step. The robot is then guided, based on the resultant gradient made by the different potentials acting at a point in space. Even though this method is considered to have low computational complexity, it is seldom used by itself due to the problem of local minima [132].

### 2.4.2. Node-Based Optimal Algorithms

These algorithms divide the environment into nodes or grids and calculate the optimal distance from start to goal. The examples for such algorithms are Dijkstra's algorithm, A-star, Lifelong planning A-star (LPA-star) and D-star.

#### *2.4.2.1. Dijkstra's algorithm*

Here, the nodes are first classified into two lists based on whether they have been visited or not. Each node is then allocated a distance value which is infinity for all except the first which will be 0. Initially, all the nodes will be in the unvisited list. The distances from the first node to the neighbouring nodes are calculated and the distance value set in each node initially is updated to the lowest measured distance. That is, the shortest distance from each node to the initial node is marked at the end of the cost calculation. The visited node is then transferred to the corresponding list and this process continues for all nodes until the goal

node is visited. In the last step, the cost of each node from the goal is used to trace the route back to the origin [133].

*2.4.2.2. A-star algorithm*

A-star is a commonly applied algorithm for performing path planning in mobile robotics. The algorithm decomposes the environment into nodes and searches for a path to the goal with the least cost. This is accomplished through an iterative calculation of a cost function at every node as in Dijkstra's algorithm. The difference between the two is in how the cost function is calculated. While Dijkstra's algorithm uses only the cost incurred to reach the current node, A-star combines that with the cost to goal and thereby ensures faster convergence [134]. The cost function in an A-star algorithm is given by,

$$f(n) = g(n) + h(n) \tag{2.1}$$

Where for the $n^{th}$ node, $g(n)$ is the cost from initial node and $h(n)$ is the estimation of cost to the goal. Dijkstra's algorithm can be considered as a special case of the A-star algorithm with $h(n)$ always equal to 0.

*2.4.2.3. D-star algorithm*

D-star stands for Dynamic-star algorithm and is another variation of A-star with the key difference being its ability to handle dynamic obstacles. It works like A-star, except that when a new obstacle is encountered, it is added on to the map and a new A-star calculation is performed to reach the destination [135].

### 2.4.3. Mathematical Model-Based

The path planning algorithms discussed above, do not consider the environmental system dynamics and thereby do not use all available information. This gap is addressed by mathematical model-based algorithms where the environment is modelled as a time variant system with multiple kinematic and dynamic constraints. Examples of the approach are linear algorithm-based [136] and optimal control-based [137]. The main criticism of these methods is the high computational cost that often makes them inefficient for online use [126].

### 2.4.4. Bio-Inspired Algorithms

As the name suggests, these algorithms attempt to imitate nature's methods of problem solving. The major types here are EAs and NN with their mechanisms discussed in *Section 2.2*. For path planning, EAs evolve a population of solutions or, in this case, the chronological order of nodes in a map until the goal is reached with desired performance [138]. In NN-based path planning, an example is [139]. There, a NN grid was built with each neuron responsible for a node or a particular area in the map. The activation function of neurons corresponding to each obstacle was set to 0 and goal set to 1. The sensory input triggered changes to the network and the neural activity gradient decided how the motors were controlled.

### 2.4.5. Multi-Fusion Algorithms

This class of algorithms is a result of combinations of the path planning algorithms explained above to mitigate the shortcomings of each. Examples of such algorithms are VVP (a combination of Voronoi and potential field) [140], a hybrid of mathematical model and EA [141], PRM and node-based [142], or sampling-based on EA [143].

In the above section, major path planning algorithms were described and discussed due to their planned application in *Chapter 4*. However, the main contribution in this thesis is in the design of a new evolutionary algorithm (in *Chapter 5*) that incorporates a learning mechanism for improving controllers evolved during co-evolution. Therefore, a discussion on the concepts used in learning algorithms is presented below.

## 2.5. Reinforcement Learning

Control algorithms in robotics can be broadly classified into algorithms that are based on a mathematical model of the robot and surroundings, which use a set of rules (e.g. a simple rule could be, if the sensor detects an obstacle, turn right) and Artificial Intelligence (AI) and Machine Learning (ML) types. In the AI/ML approach, the developed controller is much more robust as it can (or tries to) behave rationally even when it encounters unknown scenarios. Despite mathematical model-based approaches guaranteeing a good solution, it is not preferred for complex systems because of the inability to create accurate models, the

amount of time taken or lack of proper methods available to solve higher order differential equations that represent the system, and limitations imposed in online use due to the computational complexity involved.

The AI/ML approach works through a gradual modification of the controller over the design process or even during the life of the robot. The main benefit here is the ability of the approach to develop a suitable controller, often from scratch, with limited manual intervention. In other methods when the engineer builds the entire controller, only a learning platform for the controller needs to be built. This can be through setting up a controller that is based on a fixed NN architecture, generating necessary training data (when available) and building a learning algorithm to modify the NN [144].

The conventional learning process is referred to as either supervised or unsupervised. In Supervised Learning (SL), the initial controller is modified depending on the extent the output is away from the expected or target output signal. This adjustment is repeated with multiple input-output combinations from a database of training data. Evidently, the more versatile the training data is, the better the controller. This learning is the time-consuming part in the controller design process. While the training data in SL has clearly labelled outputs that work as a feedback for the AI controller to develop, in Unsupervised Learning (UL), the dataset does not contain any labels and the learning process here involves finding patterns in the data.

While SL offers a good approach to designing robust controllers, its success mainly hinges on the quality of the training data and it is not often practical to generate a large dataset that covers all possible combinations of inputs and outputs. To overcome this problem, Reinforcement Learning (RL) has been introduced [144].

RL is inspired by methods humans normally use to learn a new skill. An easily relatable example is how one learns to ride a bike. It is neither through supervised learning that says when to do what (through a series of steps), nor through unsupervised learning that does not say anything on what to do or what is expected. Instead it is through trial-and-error of trying to ride and then falling, which inflicts pain that acts as feedback, or reinforcement from the environment. Similarly in RL, an agent acts in a stochastic environment based on a control

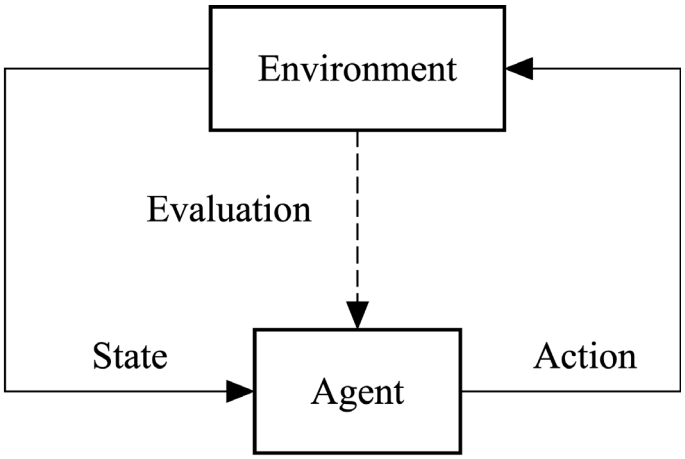



**Figure 9. A typical RL framework.** *The agent receives rewards for the actions it performs in an environment. The rewards and state information are then used to decide the next action.*

policy and receives a reward or return. The objective of the agent or robot is to maximise the long-term reward throughout its lifetime.

A typical RL framework is shown in Fig. 9. The actions that are allowed are $a \in A$, state $s \in S$ is information based on which the robot makes a decision and policy $\pi$ maps $s$ to $a$ ($a = \pi(s)$), to maximise the cumulative scalar reward over each episode received through a return function. In robot navigation, $a$ can be motor control signals, $s$ can be sensor readings and $\pi$ can be through a neural network that accepts input $s$ and generates $a$. The agent via its actions in the environment must therefore learn the optimal state-action relationship through exploration. This is embedded in the policy itself or performed separately only during the learning phase.

Traditional RL methods are modelled as a Markov Decision Process (MDP) with the tuple $[S, A, \varepsilon, P(s'|s,a), R(s'|s,a), \gamma]$. Where $P(s'|s,a)$ is the transition probability to a new state $s'$ from state $s$ and action $a$, $R(s'|s,a)$ is the reward expected from the transition, $\varepsilon$ is the random variable the policy function depends on, and $\gamma$ is the discount factor that increases or decreases the reward memory [144]. In robotics, MDPs are modelled as Partially Observable MDPs (POMDPs) due to the high dimensional nature of the entire system and since not every dimension is observable at all times [145].

Applications of RL approaches in robotics are classed as either model-based (indirect) or model-free (direct). In the model-based approach, along with a policy, the system dynamics are also built into the model in a similar way to traditional model-based control. However, this approach is seldom used. On the other hand, in a model-free approach, the policy is updated directly instead of both policy and model in the previous case [144]. The following sections discuss the model-free learning due to its relevance to the project.

The two most prominent methods to solve a model-free RL problem are through value function and policy search.

### 2.5.1. Value Function-Based

In a value function approach, the Bellman equations for a value-state function and an action-value function can be written respectively as:

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s\prime} P(s'|s,a)[R(s,s',a) + \gamma V^{\pi}(s')] \quad 2.2$$

$$Q^{\pi}(s,a) = \sum_{s\prime} P(s'|s,a)[R(s,s',a) + \gamma Q^{\pi}(s',a')] \quad 2.3$$

It can be shown that there exists an optimal policy that maximises the value in *Equation 2.2* and provides an optimal state-value function for *Equation 2.3*. The approaches focus on maximising the equations for each of the cases ($V^{\pi}(s)$ and $Q^{\pi}(s,a)$) by using Dynamic Programming, Temporal Difference (TD) methods and Monte Carlo (MC) methods [145].

Dynamic Programming consists of either policy-iteration or value-iteration types. Policy-iteration performs two steps. In the first step, after a random initialisation, the policy is evaluated with the corresponding value function, current and successor states. After this reaches a desired point, the policy is improved with the updated best action for every state. Both steps are iterated until a desired performance level is reached. While the policy is only updated after the convergence of a value function when using policy-iteration, in the value-iteration method both policy evaluation and update occur in the same step. MC methods rely on sampling for value function estimation and temporal difference methods calculate temporal errors to estimate the value function during each episode whereas the other methods wait for the end of each episode before performing a value function computation [145].

### 2.5.2. Policy Search-Based

Different to value function-based methods, the Policy Search (PS) based methods offer the option of integrating expert knowledge via policy structure or during policy initialisation. At the same time they have fewer parameters than value functions hence they are favoured in robotics [145]. Normally, an existing policy $\pi$ parameterised through $\theta_i$ is updated by $\Delta\theta_i$ as,

$$\theta_{i+1} = \theta_i + \Delta\theta_i \tag{2.4}$$

Literature shows that the key difference in various policy search algorithms lies in how $\Delta\theta_i$ is computed. They are categorised on how the policy is explored, evaluated and updated during learning [146]. The policy exploration strategy is responsible for generating new trajectories for the robot to follow which in turn creates rewards that are finally used to update the policy. Over exploration should be avoided as this can cause excessive wear and tear on physical robots. In a step-based strategy, the exploration is performed at every step while in an episodic exploration, the process only happens once at the beginning of an episode. In policy evaluation, depending on whether it is step-based or episodic, either the quality of each state-action pair is tested at each step, or the entire parameter vector is tested once at the end of each episode, respectively.

The last step of a policy update is prominently through policy gradient, expectation maximisation and information-theoretic approaches [146]. The policy update can either be performed as part of the long search process or just by itself, referred to as Direct Policy Search (DS).

#### *2.5.2.1. Policy gradient methods*

These use gradient-ascend to maximise the expected return $J_\theta$. The gradient $\nabla_\theta J_\theta$ is evaluated to update the policy as,

$$\theta_{i+1} = \theta_i + \propto \nabla_\theta J_\theta \tag{2.5}$$

where $\propto$ is the learning rate. Further variations of $\nabla_\theta J_\theta$ or the gradient computation differentiate available policy gradient methods. In an episode-based setting, the finite difference policy gradients proposed by Peters *et al.* used a first order Taylor-expansion of

$J_\theta$ for gradient calculation [147]. The REINFORCE algorithm by Williams is one of the earliest that applies the *likelihood-ratio trick* for calculating $\nabla_\theta J_\theta$ [148].

The REINFORCE algorithm suffers from high gradient variance that results in slow convergence. To reduce this, several approaches are reported in the literature. Specific to POMDPs that often require details about the internal state or have memory of previous states, the Gradient POMDP (GPOMDP) algorithm developed by Baxter *et al.* used just one parameter $\beta \in [0,1)$ without the need of any underlying state information to manage the bias-variance ratio or the tradeoff between the biased gradient of the average reward and variance [149]. The authors also proposed Online POMDP (OLPOMDP) which is another direct policy gradient based algorithm which can be implemented online. In GPOMDP, the algorithm updates the policy after accumulating the rewards at every time-step and finally modifying the policy. But the policy is updated in every time-step in OLPOMDP. The convergence of OLPOMDP is proven in [150]. An enhanced version of OLPOMDP is Policy Gradient for Reward Design (PGRD) which was designed exclusively to improve the reward function outcome over an agent's lifetime [151].

Other variations of policy gradient methods are now discussed. In [152], the GPOMDP was further modified into multiple algorithms to add internal memory. The tasks were split into subtasks with individual state-action spaces before calculating the gradient in [153]. The policy gradient computation with parameter-based exploration (PGPE) involved the policy parameters being defined as a distribution that was sampled at the beginning of each episode [154]. The advantage of this was that it did not induce noise into the gradient at every time-step thereby reducing variance. PGPE was applied in [155] to develop a controller for the *Alpha-Go* board game. A value function was designed for variance reduction through an actor-critic architecture for policy learning in [156]. The complete learning of quadrupedal locomotion was performed on a real robot by Kohl *et al.* [157]. They split the rewards obtained in each time-step into three categories and used an additional parameter to decide the calculation of a weighted average reward. This averaged reward pointed to the most favorable direction for updating the policy.

Peters *et al.* presented an episodic natural actor-critic algorithm that performed better than the REINFORCE and GPOMDP algorithms [147]. In an actor-critic setup, the critic

computes the value function and actor updates the policy based on the value function. The random exploratory nature at every time-step that contributed to high variance was fixed by the addition of a State-Dependent Exploration function (SDE) that selected a fixed action for every state [158]. In a different approach, to compensate for limited memory in policies that needed long term memory, a LSTM (Long Short-Term Memory) RNN based architecture was proposed. The network weights in a LSTM RNN were updated by approximating the policy gradient and backpropagating it through time [159], [160]. Contrary to the standard method of gradient estimation in both action and state spaces in a stochastic policy, Silver *et al.* developed a deterministic policy gradient that only worked in the state space and thereby reduced sampling time [161]. A similar algorithm with a policy gradient based actor-critic to enable biped locomotion was later applied on a humanoid robot with a CPG-based controller [162]. The actor generated CPG signals while a critic estimated value function using a TD error.

In PS approaches, a relatively new development lies in the use of a limited number of trails to generate useful results unlike in typical RL methods [163]. This is in contrast to the recent paper that generated renewed interest in RL [164]. There, Silver *et al.* taught an agent to play *Alpha-Go* from scratch while taking 29 million games for the learning process to complete. Despite performing the experiments on a state of the art Tensor Processing Unit (TPU) designed specifically for AI/ML, this task required days to complete. Fast PS methods either rely on the availability of prior knowledge of the system or build a surrogate model of the system through the lifetime of the agent. Even though this is particularly suited for model-based RL, there are a few exceptions such as the transferability approach that assumes the accuracy of physics simulators for certain tasks while training with limited trials [105].

### 2.5.3. Other Model-free RL Methods

Expectation maximisation algorithms have been designed to avoid convergence problems of policy gradient methods due to poor learning rate selection [146]. Examples of such algorithms are the Expectation Maximization (EM) algorithm, MC-EM [165], and Variational Inference for policy search [166]. Another category in the model-free methods is the Information-Theoretic approaches that combine the advantages of gradient-based and

EM algorithms. Relative Entropy Policy Search [167] is such an algorithm that performs policy search through the optimisation of trajectories and state-action pairs.

### 2.5.4. Return Functions

Return functions are an integral part of the RL process as they act as primary feedback to inform the agent on its performance. Therefore their improper design can lead to significant impact on the outcome of the learner as the policy change depends on them [168]. They aggregate the positive or negative rewards over the lifetime of the agent. In robotics, finite horizon return functions are used, which can be achieved using discounted or averaged return functions. Each can be shown respectively as,

$$R = \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \tag{2.6}$$

$$R = \frac{1}{T} \sum_{t=0}^{T-1} r(s_t, a_t) \tag{2.7}$$

The discount factor $\gamma$ ensures that the initial rewards are valued more in the discounted return function while the rewards are given equal weights in an averaged return function. $\gamma$ is selected to balance between convergence time and quality of learning.

Even though the importance of a proper reward function is well documented, currently there is no theoretical underpinning on how to effectively design them. Kobel *et al.* in their survey paper [145] referred to it as the *curse of goal specification* while explaining how difficult it is to develop suitable reward functions. Normally rewards are a simplified representation of goal achievement. Examples are binary rewards for holding a ball in a cup [145], three possible integer rewards for moving in the correct direction [169] or a simple distance-based reward based on closeness to goal [170]. Among other approaches, the inverse optimal control proposed by Russel [171] has been used by several researchers to perform reward function design. Here, instead of manual reward function design, the system develops a reward function automatically through a series of expert task demonstrations. There have also been works that specifically design reward functions e.g. [151], [172]. In the recent paper [172], the authors emphasised how hard it is to design a reward function that captures the task at hand. Consequently they proposed Inverse Reward Design (IRD) that works with

the help of a proxy reward function and a set of possible reward functions while training the MDP.

## 2.6. EA-RL Hybrid Algorithms

*Sections 2.2* and *2.5* showed how EA and RL algorithms are closely related to natural selection methods for problem solving. The learning principles mimic survival strategies organisms use over their lifetime while the evolutionary principles handle long-term goals of the entire species. Even though it may seem like they work on two different scales, a closer examination shows that optimisation is a key target in both. This can also be observed in their applications for solving engineering problems. There exists a class of algorithms that fuse these ideas, and this is the focus of *Chapter 5*. Therefore, this section covers a discussion around those algorithms that combine evolutionary and reinforcement learning principles.

These algorithms can be classed into three types: RL in EC, EC in RL and general purpose optimisers [173]. The RL in EC and EC in RL categories respectively handle algorithms that use RL to enhance the EC process and vice versa. On the other side, general purpose optimisers are neither purely EC or RL, but combine some concepts from both, e.g. a predator-prey model that used RL for survival and EA for evolution [174]. EC in RL algorithms: In multi-agent systems, the RL agents used an EA to help each other exchange and acquire the best skills learnt [175]. Other examples are [176] where a GA was used to perform a policy search, a GP used for RL state feature discovery [177], evolutionary function approximation in RL [178] and CMA-ES used for policy search [179].

### 2.6.1. RL in EC

There are several papers that reported the use of RL principles to improve the evolution process. Buzdalova *et al.* proposed a combination of EA with a Multi-Objective RL (EA-MORL) [180]. Here, the fitness function for the evolver was controlled by the RL algorithm, effectively resulting in a multi-objective selection through the evolution process. The reinforced genetic programming by Downing [181] improved the efficiency of the GP evolved program by performing modifications at a node level with the help of an RL algorithm. Proper parameter tuning is a well recognised hurdle during the use of EAs and,

in an attempt to fix the defect, Karafotias *et al.* [182] and Thierens [183] applied RL to modify EA parameters during evolution. Online variation operator probabilities were calculated by the multi-arm bandits algorithm in [184].

The GA-RL algorithm in [185] evolved with states where actions were represented as variation operations. Selected actions depended on the state-action probability and rewards that were used to update the probability depended on the relative parent-offspring fitness. A similar method was presented in [186] where a population of RL agents participated in a GA driven evolution. In a graph-based genetic network, the EA took charge of a diversified search by connecting nodes in the graph based on a fitness function while the RL handled an intensified search and selected the best possible node transition depending on rewards [187]. Learning Classifier Systems in [188] that generated *if-then* rules used a GA for rule discovery and RL for reinforcing good rules. The most recent hybrid algorithm is Evolutionary Reinforcement Learning (ERL) developed by Khadka *et al.* [189]. A population of actors involved in the fitness evaluation in the environment, and fitness allocated depended on the rewards received at every time-step. While the typical EA progressed, an RL-actor simultaneously observed the experiences in the fitness evaluation simulator and updated itself through an actor-critic RL algorithm. Then, at random times, the RL-actor was injected into the population to pass on the newly learnt experiences to subsequent generations.

## 2.7. Summary

This chapter discussed literature pertaining to various topics relevant to this thesis. A wide range of areas covering published articles on automatic robot generation, mobile robot path planning algorithms, Reinforcement Learning and Evolutionary Reinforcement Learning hybrid algorithms were discussed. Explanations of fundamental concepts and relevant tools exploited in this thesis were also included. Building on those, the next chapter reports experimental attempts conducted to answer some of the questions while simultaneously evolving mobile robot bodies and controllers.

# Chapter 3.

# Effect of Evolution Parameters on Co-Evolution

The literature presented on co-evolution showed different ways to evolve robots for different applications. *Section 2.2* explained methods and parameters that are part of the evolution process. Multiple factors, such as costly computational requirements, the large number of adjustable variables and the random elements of the process, forced the application of EAs to be constrained to highly targeted design and optimisation problems. Furthermore, an application-oriented presentation of findings in literature made it unclear how and why each of the available options were chosen during evolution. Many times, the research only reported results with a certain set of evolution parameters, selected on an ad hoc basis. There is also a lack of consensus on how the evolution process should be approached to solve design problems in robotics. Therefore, to improve the process of evolving buildable robots, it is necessary to have a better understanding of the process itself. Hence, this chapter describes a study and analysis of the behaviour of evolving robots under various conditions. The experiments demonstrate how the variation of each parameter affects the course of evolution.

The adopted methodology, (implemented in *RoboGen*) was the following: Evolved robots were expected to perform navigation and obstacle avoidance. Here, navigation refers to the action of moving as far as possible from the centre of the virtual arena in a fixed amount of time. This meant moving along a straight line unless deviating around obstacles. Fitness evaluation for all of the experiments in this chapter was performed with the default fitness function in *RoboGen*. This fitness was allocated solely based on how far (Euclidian distance in metre) the robot had reached from the centre of the arena at the end of fitness evaluation simulation. Therefore, a robot remaining stationary or a robot returning to the centre at the end of simulation would have zero fitness. This calculation was performed on all individuals of the solutions population. The majority of the experiments discussed in this chapter performed fitness evaluation in an open arena. When obstacles were used, their design

**Table XI. Evolution parameters.**

| Parameter | Value |
|---|---|
| Population size | 20 |
| Number of evolved children | 20 |
| Probability of brain mutation | 0.3 |
| Number of generations | 100 |
| Sigma value of brain | 0.7 |
| Brain bounds | 3:3 |
| Minimum and maximum number of initial parts | 2:10 |
| Probability of node insertion | 0.1 |
| Probability of sub-tree removal | 0.1 |
| Probability of duplicating sub-tree | 0.1 |
| Probability of swapping sub-tree | 0.1 |
| Probability of node removal | 0.1 |
| Probability of modifying parameters | 0.1 |

process is explained in corresponding sections. Sample evolution configuration parameters, which were used as default values during experiments, are listed in Table XI. These were also the default values of evolution parameters set in *RoboGen*. The evolution was performed on a Linux PC with an Intel i7 dual-core 2.50 GHz processor and 8 GB of RAM.

Literature indicated that the effectiveness of the evolver was only as good as its fitness evaluation simulation platform, suggesting that the simulator plays an important role in the entire process. To avoid unexpected simulation inconsistencies, *RoboGen's* tested and recommended settings for the virtual simulation platform were used. Unless otherwise mentioned, each robot was run for 8 s in the virtual environment, over a flat surface (the only option in *RoboGen*), and readings were recorded every 0.005 s to avoid any loss of data. Eight seconds was chosen as initial experiments designed for simple obstacle less navigation demonstrated robots had achieved a stable fixed trajectory by this time. The step resolution was chosen to obtain usable results while keeping simulation times to a manageable level. Increasing the time-step would increase the time taken to run the simulations (it is the resolution recommended by *RoboGen*).

In this designed setup, various experiments, described below, were conducted while varying the parameters such as number of evolved generations, number of parts used for evolution, population size and number of children generated. It has to be noted that the main focus of

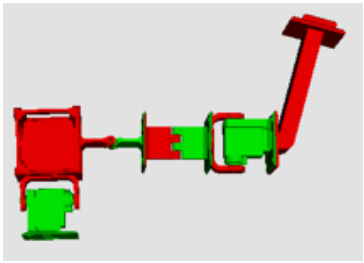
the experiments was to observe the parameters that contributed to the most computational burden during evolution. The most computationally expensive part of evolution is the fitness evaluation of each evolved individual. This evaluation time can very quickly multiply depending on the size of the evolving population or the number of generations evolved. Therefore, better understanding of how such parameters affect evolution can help save computational resources. Furthermore, the aim of the experiments was to deduce useful information for selecting various parameter values used in the remainder of this thesis, or for identifying viable research areas for improving co-evolution.

## 3.1. Experiments and Results

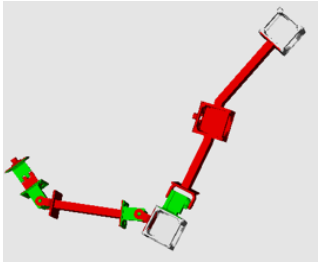
To understand the effect of variation of individual evolution parameters (listed in Table XI) in fitness of the robot, multiple experiments were designed to specifically vary each of the elements in Table XI separately. As discussed in *Section 2.3*, these parameters mainly relate to the evolver engine of *RoboGen*. In all these experiments random robots are initialised based on set evolution parameters. The number of initiated robots depend on the population size and each of these are then send to the simulator for fitness evaluation. The calculated fitness is then fed back to the evolver for progressing evolution via parent selection, variation operations and population update. Unless specified, all experiments below are conducted once with the seed to the random number generator set at 1.

### 3.1.1. Effect of Number of Generations

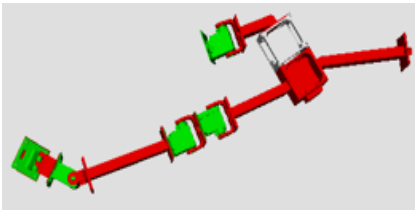
To study the effect of generations on fitness value, the population size was fixed at 20. With a maximum of 20 initial parts, robots were then allowed to evolve on a flat surface for over 37,000 generations. One can observe that most morphology changes were noted in the early phase of evolution, with the final robot shape remaining identical in the last 32,000 generations of evolution, while its fitness increased from 1.8 to 2.7 (Fig. 10 (i)). During these last 32,000 generations, the number of parts increased from 7 to 16. Even though robots evolved for up to 37,570 generations, morphology changes were observed only 9 times. Individual parts of the evolved robot and their positions within the final robot (Fig. 10 (i)) are listed in Table XII. This robot in Fig. 10 (i) did not have any IR sensors. The improvements in fitness of these robots as the generations progressed can also be observed from Fig. 10.



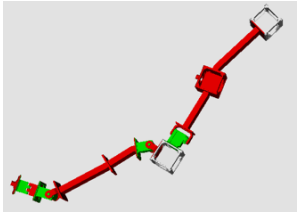
(a) Generation- 1, Fitness- 0.0001, No. of parts- 7



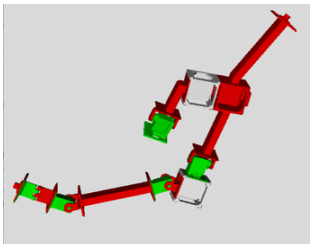
(e) Generation- 61, Fitness- 0.91, No. of parts- 12



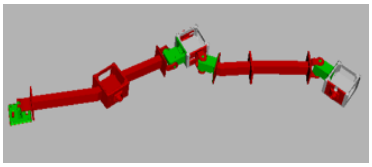
(b) Generation- 2, Fitness- 0.13, No. of parts- 13



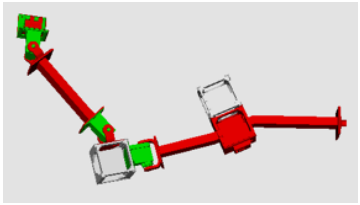
(f) Generation- 79, Fitness- 1.06, No. of parts- 13



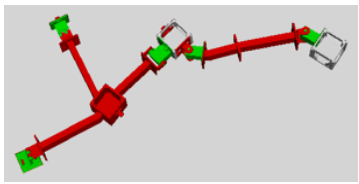
(c) Generation- 7, Fitness- 0.39, No. of parts- 15



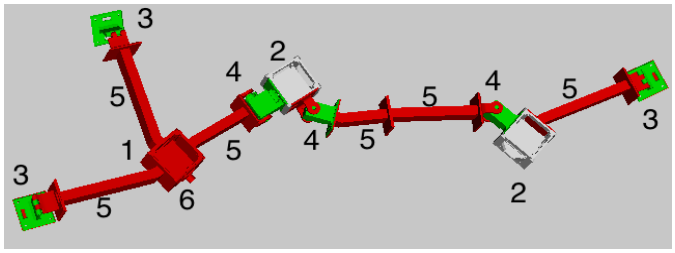
(g) Generation- 3098, Fitness- 1.80, No. of parts- 14



(d) Generation- 19, Fitness- 0.61, No. of parts- 12



(h) Generation- 3203, Fitness- 1.84, No. of parts- 15




(i) Generation- 37570, Fitness- 2.72, No. of parts- 16

**Figure 10. Progress of morphology evolution.** *Limited changes to morphology over 37,570 generations shows the controller is modified more than the morphology.*

**Table XII. Part details for the robot in Fig. 10 (i).**

| Sl. no. | Part type | No. of occurrences | Colour |
|---|---|---|---|
| 1 | Core brick | 1 | Red |
| 2 | Fixed brick | 2 | Grey |
| 3 | Passive joint | 3 | Red-Green |
| 4 | Active joint | 3 | Red-Green |
| 5 | Parametric bar | 6 | Red |
| 6 | Light sensor | 1 | Red |
| 7 | IR distance sensor | 0 | N/A |

Initially the average fitness of all the members of the population showed an expected deviation from the best individuals as seen in Fig. 11. In these early stages of evolution, when the evolver first created a better robot (than the rest of the population), its fitness was considerably different to the other robots in the population. Over time, multiple copies of the best robot were made and/or new offspring from the best robots that had a better fitness appeared and this resulted in an improvement of average fitness of the population. Furthermore, as the fitness of the best individual settled, the average fitness moved towards the best fitness.

This is further supported by the fact that on multiple occasions during the experiments, the standard deviation of the population converged to zero, which meant that all the individuals in a population were identical and even a slightly less fit robot was discarded. This pointed to an evolver bias that resulted in a greedy selection of best robots to fill the population without considering the need for diversity, unlike the natural paradigm where diversity is a key factor for the success of a species. This problem is further discussed in *Chapter 6.*

Each evolution was initiated with a seed number for the random number generator used by the evolver. In all the experiments above, the seed was set at 1. However, it was also found that changing the seed meant loss of repeatability of obtaining the same set of results. That is, if experiments began with the same seed and evolution parameters, this led to the same

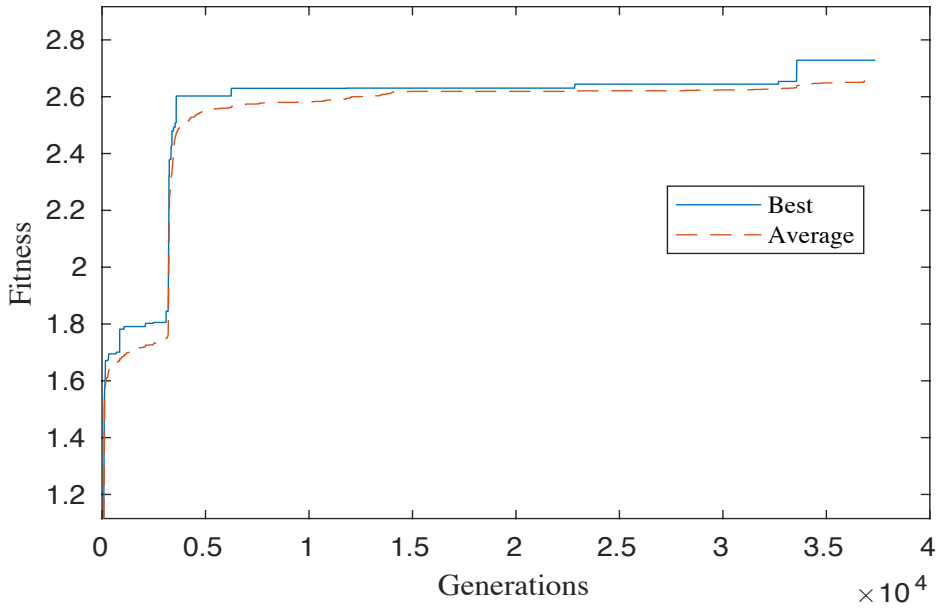



**Figure 11[2]. Fitness of best robot and average fitness of population versus generations.** The x axis shows the evolution time through a number of generations. The y-axis is the increasing fitness of the best robot in an entire population or the average fitness of the entire population. *The sudden jumps in fitness suggest either a morphology change or a major change in the controller. The average fitness of the population closely matches the best robot showing how relatively unfit robots are removed from the evolution process through displacement by fitter robots and in most cases with multiple copies of the best robot.*

robots. On the other hand, it can be considered as an advantage as this allows the testing of robustness of the set evolutionary scenario despite the stochastic nature of EAs. This repeatability was a result of the way *RoboGen* is designed to operate. Furthermore, it can indeed be considered as a useful feature as this helped to guarantee consistent results while transferring the software across multiple machines. Though methods can be devised to look at how different initial seeding affected evolution, these would have to deal more with how seeds affect the output of pseudorandom number generators in *RoboGen* which is beyond the purview of this thesis.

In a next set of experiments, the number of evolved generations was varied while keeping all other parameters fixed. The maximum evolved generations were doubled in every

[2] Figure title is in **bold**, description is in regular and commentary is in *italic* font.

experiment in a range from 100 to 35,000. The results demonstrated repeatability for all experiments up to 10,000 generations. After this, the evolver branched off in different directions by building different robots which suggested that *RoboGen* could only generate repeatable results until the 10,000$^{th}$ generation. This limitation was probably due to the way *RoboGen* was constructed.

### 3.1.2. Effect of Number of Initial Parts

The initial number of parts used or, available for use during evolution also played a role in the robot behaviour. In this case, the range of the maximum number of initial parts was varied from 10 to 100. Even though they seemed to have a clear impact on fitness progression, the results did not exhibit any discernible patterns. In Fig. 12, an experiment with 80 initial parts shows the least increase in fitness rate over the period of the experiment. In this particular case, this was a result of the sudden increase of number of robot parts from 5 in the first generation to 70 around 300 generations. Additionally, the hardware building constrains allowed the robot to only have a maximum of 8 motors which would normally not be sufficient to generate enough torque to move 70 parts.

In the initial 50 generations of all of the experiments, the rate of increase of fitness exhibited a different pattern when compared to the behaviour in other evolution runs (Fig. 12). The experiment evolving from 70 initial parts showed the fastest speed of fitness increase followed by experiments with 60 and 90 initial parts. The lowest rate of change was exhibited by the experiment with 10 initial parts followed by 80 initial parts. The experiment with 60 initial parts was the fastest to settle down in ±5 % of its final fitness followed by experiments with 90 and 100 initial parts. Despite beginning with a specific number of initial parts, the experiments performed showed variations in the number of parts in the best robot from the first generation onwards. This was caused by the effects of the pseudorandom number generator in *RoboGen* that was used to select various probabilities during evolution. The variable number of parts in the best robots continued as the generations progressed (can be seen in Table XIII).

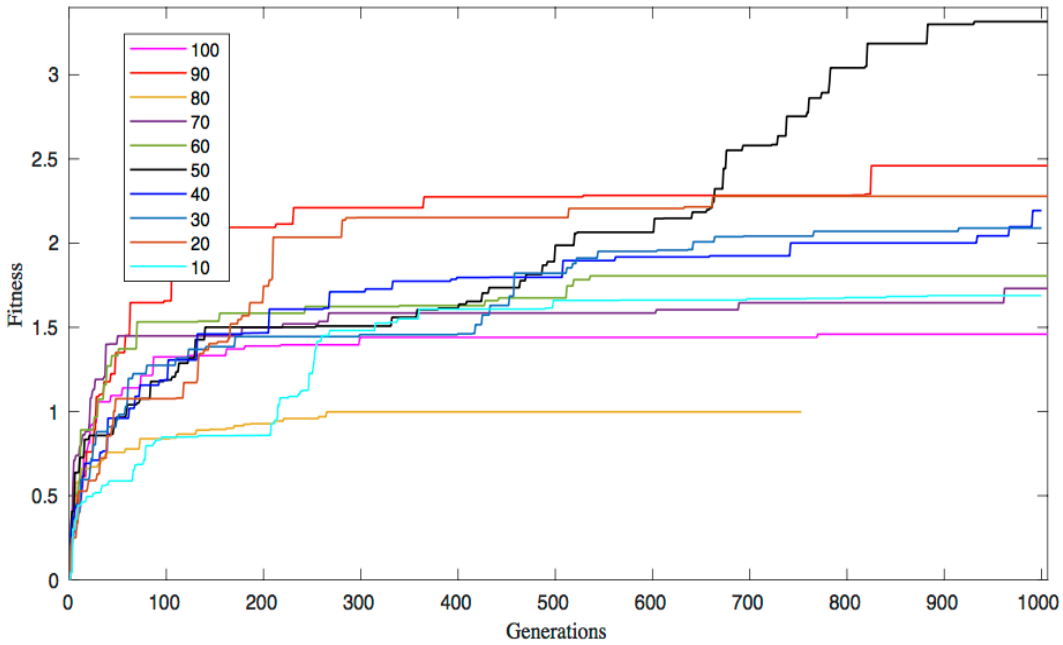



**Figure 12. Average fitness change to initial number of parts.** *Experiment with 50 initial parts shows fastest growth over the first 1000 generations while the experiment with 80 initial parts is the slowest. However, over the first 100 generations, the experiment with 90 initial parts improves fastest while the experiments with 10 and 80 initial parts closely follow each other. This behaviour is a result of the availability of sufficient parts on evolved robot in the first generation in case of former and similar number of parts in latter. An incomplete curve in the graph indicates that the evolution did not complete in the set timeframe due to the time-consuming mathematical calculations involved.*

**Table XIII. Parts numbers and fitness in different experiments.**

| Initial number of parts on evolved robot (max. available parts initially) | No. of parts at 1000 generations | Best fitness at 1000 generations |
|---|---|---|
| 5 (80) | 70 | 1 |
| 7 (20) | 21 | 2.3 |
| 10 (10) | 7 | 1.6 |
| 17 (30) | 22 | 2.1 |
| 17 (40) | 16 | 2.1 |
| 20 (70) | 19 | 1.6 |
| 20 (90) | 22 | 2.4 |
| 23 (60) | 14 | 1.7 |
| 37 (50) | 32 | 3.3 |
| 74 (100) | 19 | 1.4 |

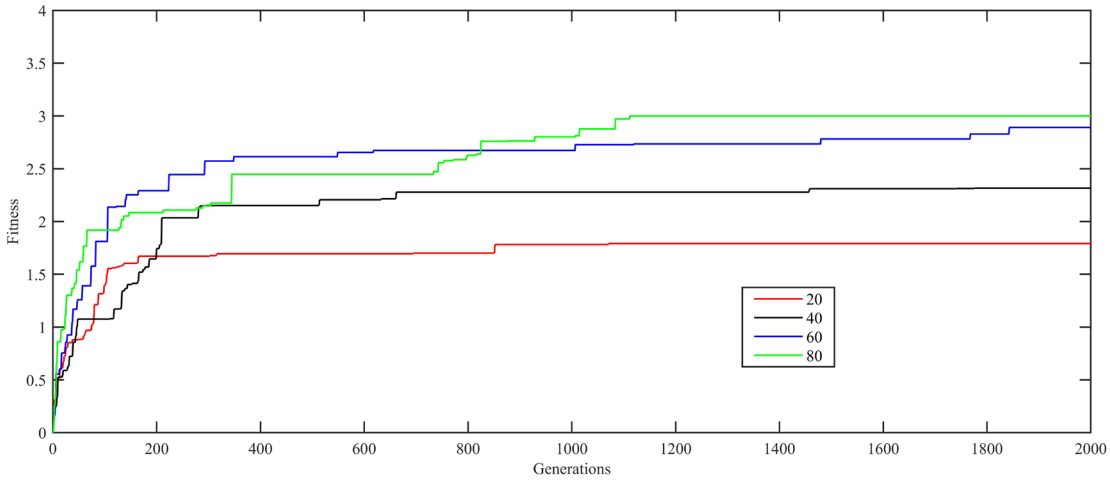



**Figure 13. Best fitness variation to population size.** *In general, higher population size is associated with faster growth. However, until steady state condition is reached, experiments show variable growth rates.*

The experiment that had the best fit robots was the one with 50 initial parts and the worst performing individuals were found in the experiment with 80 initial parts. Except for when there were 70 parts in the robot, bigger robots normally corresponded to fitter robots. The main factors that contributed to such behaviours were the availability and position of actuators, suitability of controller and effect of randomness. These were in addition to the effect of evolution and simulation parameters.

### 3.1.3. Effect of Change in Population Size

The population size was varied from 20 to 80 members and experiments were run for 2000 generations. It was expected that a population of 80 individuals would stabilise earlier than the smaller populations as more individuals would lead to more chances for the creation of better offspring. This trend can be seen in Fig. 13. The experiment with 80 individuals in a population stabilised first to a fitness value of 3 in about 2000 generations while the 20-member sized population needed the maximum amount of time to approach 1.7.

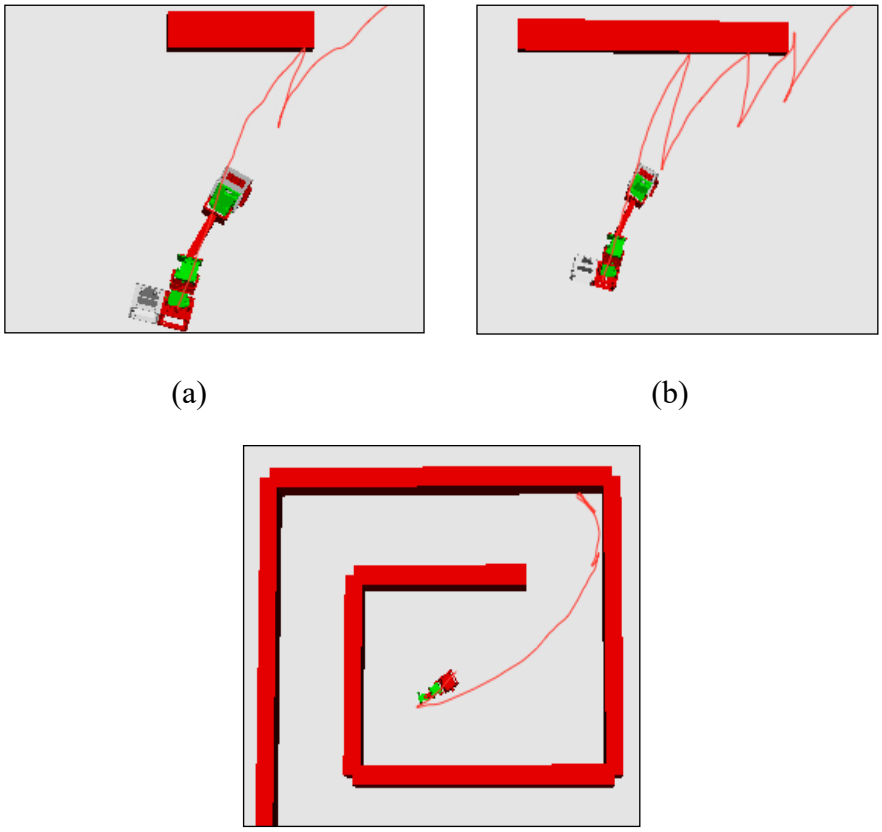



**Figure 14. Obstacle avoidance trajectories of evolved robots.** In (a) and (b), the same robot is placed in a virtual arena with different obstacles (in red). The robot in (c) is evolved in an attempt to move out of the arena. *The tendency of the robot to make trajectory changes mainly due to collision with obstacle and not sensing the obstacle is visible. The repetitive nature of the controller to perform the same set of movements can also be seen ((a) and (b)).*

### 3.1.4. Effect of Obstacles

To understand how robots behave when they are placed in a different environment, they were originally evolved with just one obstacle as shown in Fig. 14 (a). The size and position of the obstacle was determined based on experience from previous experiments (though not reported here). Here evolved robots previously showed a higher chance of moving towards the top of the shown virtual arena under the set evolution parameters. So an obstacle with a suitable size was placed along the vertical axis as seen in Fig. 14 (a). It was expected that the availability of obstacles would cause robots to evolve obstacle sensors. The best robot's travel route was recorded (red lines in Fig. 14 (a)) and in the next experiment, the same robot was placed in a different setup with a different sized obstacle. The observations from both

cases showed that the robot was performed course corrections due to collision and not with the use of available sensors. The new course of movement along with new obstacle positions are shown in Fig. 14 (b). It needs to be also noted that due to the use of RNN based controller, the robot seemed to follow a cyclic trajectory evident from the back-and-forth movements in Figs. 14 (a) and (b).

In the next test, an experiment was designed to evolve robots in a maze arena. Here, the robots were allowed to evolve for 3000 generations with a 100 s window for the robots to leave the arena. The best fit individual showed a fitness of 0.64 and could just move away from the central area. The route taken by the best robot that attempted to come out of the maze is shown in Fig. 14 (c). An unexpected aspect of this robot was that it too evolved without any distance sensors. When sensors were added, they were deleted during the evolution. This suggested that obstacle sensors were not prioritised during evolution possibly due to their ineffectiveness in making a meaningful difference to the controller.

### 3.1.5. Effect of Number of Children Generated

The number of children evolved at the end of each generation will dictate the potential pool of diversity in the population. Its proportion within the population has the potential to influence evolution. So the proportion of offspring was varied to observe its effect on fitness. Here, the child population ($\lambda$) was incremented from 20 to 100 with the population size ($\mu$) varied from 20 to 100 individuals. The best fitness and average fitness of the population for each of the combinations is marked in Fig. 15. When $\mu$ was 20, the slowest to increase the best fitness value was the population generating 20 children. In all of the experiments, the populations that generated 100 children were the best or the second-best performers during evolution. This suggested that wherever possible, a population should attempt to evolve as many children as possible (subject to hardware constraints). Furthermore, the observations in Fig. 15 also indicate that a population has a higher chance of evolving better children in a set time frame if $\lambda$ is at least equal to $\mu$. The average fitness value of the entire population gradually reached its best fitness at different rates, depending on the population size.

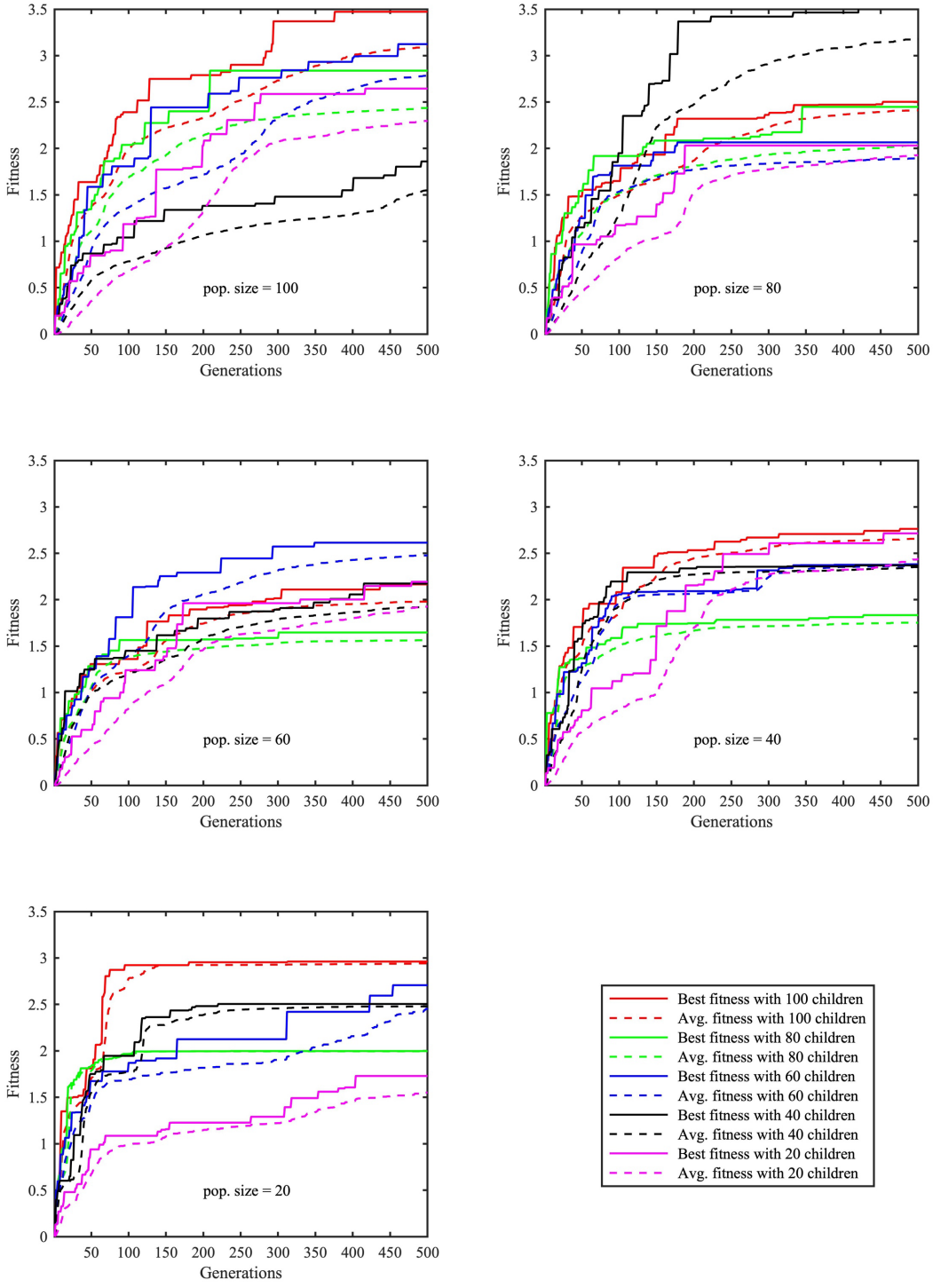



**Figure 15. Impact on fitness of the proportion of children to total population.** The number of children evolved is varied from 20 to 100 in each case. *Generally, the larger the children population size, the faster the evolution progresses, a 100-children population is therefore considerably better than the other options.*

## 3.2. Discussion

The experiments reported above offer multiple insights into the evolution process. The experiments were performed on a 4-thread processor with one thread handling the evolution and the other three performing fitness analysis in parallel. The experiments needed from hours to days to run. The population size, number of parts, number of generations run, and simulation time were the major factors in determining the time taken by the evolver. Ultimately, this underlines the general consensus in the literature that simulated evolution is a time consuming and computationally expensive process.

Observing the evolution parameters, the number of parts in a robot had a clear relation with the corresponding fitness (evident from Table XIII). Normally, more parts meant a higher fitness. Initial number of parts available for building the first population also had an effect on the progress of the population, though the relationship was not exactly clear as the number of parts present in the best robot at 1000 generations did not show any discernible pattern with the number of parts available for evolution.

The evolution process showed that there was a greater tendency for changes in the controller than in the morphology. In the entire evolution procedure, the robot morphology was altered 4 to 5 times in the first 100 generations and only a few times later on, depending on the overall number of generations evolved. This may be due to the low probability values set for body mutations, which are modified in experiments in subsequent chapters. Furthermore, the same low probabilities might have also caused the population to evolve identical individuals, that later gradually reduced the diversity of the population and ultimately reduced the standard deviation to zero. This happened despite when it was mathematically possible to have extremely high combinations of part connections. These problems could also be due to solutions being stuck in the local maximum. To overcome this, probabilities of evolution parameters should be altered or multiple trials with different starting points during fitness evaluation should be conducted.

It is possible that the unexpected behaviours observed in the obstacle avoidance experiments could be due to a poorly designed fitness function. Here the fitness function was only looking for a robot to reach as far as possible from the centre of the arena without imposing any other requirements such as allocating a fitness based on an obstacle avoidance

behaviour, so no value is placed on sensor data, hence they were deleted. This problem is further investigated in the next chapter.

Irrespective of changes in the arena, robots followed a single pattern or behaviour in their trajectories. This was because experiments were not made to generate a more robust controller by evolving with a more tailored fitness function while also using multiple starting points instead of always starting robots from the centre of the arena. On the contrary, these additions, specifically the use of multiple starting points, would mean spending more computational resources on something that did not even work as desired in a single case. Therefore, such a process was not conducted. Furthermore, due to the size of evolved robots and the design of used arena, it can be argued that a solution to the obstacle avoidance problem existed in both cases. However, the inability of the evolver to discover the solution could also be due to the lack of sufficient time allocated for evolution.

The experiments in *Section 3.1.5* also indicated that normally, the best value for the number of children evolved at the end of every population should be at least equal to the population size itself in the used ($\mu + \lambda$) population updating strategy. Here, the offspring and parent populations were equally considered for transfer to the next generation. Which meant that there was an opportunity for an individual to survive indefinitely which is unlike in biological systems where parents are eventually discarded.

Furthermore, after extensive experiments, it was observed that there are compelling reasons for more research to be performed to improve the effect of EAs in the co-evolution process. Every aspect of the evolution process from population initialisation, controller type selection, and fitness function design, to the applied EA should be studied individually and optimised or modified to reduce the generations taken to evolve results with required fitness.

While the process of simultaneously evolving the robot body and controller has been attempted since 1994 [18], the effectiveness of this process remains questionable. After days of evolution, the best evolved robot that attempted to traverse from a maze shown in Fig. 14 (c) was barely able to move away from the central part. Despite the apparent need to detect obstacles the evolved robots still lacked external sensors. Instead, the robot used an inherited cyclic trajectory rather than making decisions based on sensory feedback (Figs. 14 (a) and

(b)). The trend of a repeating trajectory was also observed in other experiments. In the evolved controller, though the RNN controller helped the evolved robots to start moving from early generations, there did not appear to be an improvement for obstacle avoidance capabilities. In the cases where suitable sensors were added, they were not used correctly in any of the experiments. This demonstrates that while EAs do work and can evolve solutions to problems, they are not necessarily intuitive solutions.

Through the steps undertaken to better understand the co-evolution process, this chapter shed light on a number of flaws with the state of the art. On one hand, the work provided details on how evolution parameters could be selected to evolve relatively better robots. But on the other, it revealed details on some of the unreported problems of co-evolutionary robotics such as evolving obstacle avoidance behaviour, or showing sensors are not used. Methods to solve these difficulties have been identified in the next two chapters to ultimately satisfy the thesis goal of evolving better mobile robots.

## 3.3. Conclusion

EAs are known for evolving unintuitive solutions to problems when applied to highly targeted design and optimisation problems in robotics [190]. In this chapter, EAs exhibited satisfactory results in evolving robots to perform repetitive actions like following a same set of steps with minute changes. While every element of the evolution process can be optimised, it is felt that the main hurdle for evolving better robots lies in the ineffectiveness of the fitness function to convey what the roboticist is aiming to accomplish. This is the main premise of the following chapter where a fitness function specifically for robot navigation is proposed.

# Chapter 4.

# An A-Star Algorithm-Based Fitness Function for Evolution

The literature revealed that any application specific fitness function so far developed relied on simple calculations to arrive at the final fitness of the robot. Additionally, the results from the previous chapter showed that effective selection of evolution parameters was not enough to produce satisfactory results from co-evolution. There, a key problem identified was the ineffectiveness of used fitness function to convey goal requirements. This is unlike in the neighbouring area of controller-only evolution that has seen several types of fitness functions such as aggregate, competitive, environmental, behavioural, incremental, tailored and training data based fitness functions applied during evolution [123]. Furthermore, within the limited number of studies that reported co-evolution in the literature review, (when compared to majority of work reported in evolution of morphology or controller [122]) none of them report how different fitness functions affect evolution for a particular application, nor how established robotics algorithms in respective applications affect co-evolution. One question that results from this is: Does using sophisticated fitness functions make a difference to the speed of evolution? In an attempt to answer this question, in this thesis, with a specific emphasis on the co-evolution process, multiple experiments were designed and tested to evolve mobile robot body plan and an ANN controller to perform navigation and obstacle avoidance in a virtual arena. The fitness function design is based around the A-star algorithm commonly used in path planning [134].

In this chapter, the research questions to be answered are the following:

1. Why do fitness functions used in co-evolution always rely on simpler calculations to arrive at a final fitness?
2. Can successful fitness function types applied in the rest of ER help evolve better robots while co-evolving body and controller?

3. Can using task specific algorithms implemented in traditional robot control process make a difference to co-evolution?

It is expected that the answers to these questions will eventually aid in evolving better mobile robots.

In an effort to answer these questions, a sophisticated fitness function was built for robot navigation and obstacle avoidance. *Sophisticated* here means moving away from typical fitness functions used in co-evolution and instead building one with a combination of multiple factors currently unseen in co-evolutionary robotics. Keeping in mind how a robot needs to react differently to environmental feedback in every step it takes, an incremental type fitness function (commonly used in other areas of ER) was designed. This would ensure that the real-time performance of each robot was properly recorded, which does not happen when a simple distance-based fitness function is used. Simple fitness functions only focus on the final outcome without paying attention to the steps a robot takes to reach there. As the task to be solved was navigation and obstacle avoidance, the fitness function was integrated with a commonly used path planning algorithm applied in traditional mobile robotics.

There are several path planning algorithms available for selection while designing the fitness function. *Section 2.4* explained different types of such algorithms used in robotics. Among the probable candidates that could be used for designing fitness functions: RRT [127] and PRM [130] were discounted due to their inability to guarantee an optimal solution; risk of non-convergence to solution for Voronoi [131] and artificial potential excluded these [132]; and high time complexity for mathematical model-based algorithms meant these were discounted. In the remaining algorithms, node-based algorithms provided a fast searching ability and easy implementation. From them, A-star [134] was chosen as it was proven to find a solution if exists and was much faster to converge than Dijkstra's algorithm [133]. D-star was discounted as dynamic obstacles were not being used [135]. It must also be noted that there were several other node-based algorithms available that performed better than A-star, but since the focus here was to test the hypothesis, the A-star algorithm was expected to be sufficient.

| **Algorithm 1: EA using A-star algorithm-based fitness calculation** | |
|---|---|
| 1. | Initialise:<br>$\mu$ random robots in a population $pop$<br>GEN > 0 //Generations |
| 2. | *for* $gen = 1$ to GEN do: |
| 3. | Compute Evaluate ($p$) for each $p \in pop$ |
| 4. | Rank $pop$ and select parents |
| 5. | Variation operations on parents to produce $\lambda$ children in $pop_c$ |
| 6. | Compute Evaluate ($p$) for each $p \in pop_c$ |
| 7. | Select survivors from combined $pop$ and $pop_c$ |
| 8. | Update $pop$ |
| 9. | *end for* |
| 10. | Return best robot from $pop$ |

## 4.1. Proposed Algorithm for Fitness Calculation

To test the effectiveness of the fitness function, a range of obstacle avoidance challenges were developed. The rationale behind obstacle placement and goal coordinates selection are explained later. For each of these challenges, the evolver needed to generate a virtual robot that could move from the centre of the arena to a specified location. The task complexity was increased by gradually adding obstacles to a 4 m by 4 m arena. The individual fixed parts available in *RoboGen* used for evolution were about 5 cm in length and so, depending on the number of parts, a 4 m by 4 m arena was sufficiently large. For each of the challenges, attempts were made to produce mobile robots which were evolved using two different fitness functions. The first fitness function allocated fitness solely based on the Euclidian distance to destination at the end of the simulation. The result was a value on a scale of 0 to 10, with 10 being maximum fitness meaning that the robot reached the set goal. The other fitness function used the A-star algorithm to allocate fitness at its core. The entire co-evolution process is shown in *Algorithm 1*.

While encouraging robots to closely follow the A-star solution, the fitness function also encouraged robots to reach the goal with a higher average speed or get closer to the goal if they did not reach the destination at the end of simulation. To incorporate these, a weighted incremental fitness function was designed as it allowed for fitness calculation in each time-

| **Algorithm 2: Function Evaluate** | |
|---|---|
| 1. | *procedure* EVALUATE ($p$) |
| 2. | Initialise: //Fitness evaluation initialisation<br>TR > *0* //Trials<br>T > *0* //Evaluation time |
| 3. | *for* $tr = 1$ to TR do: |
| 4. | Initialise:<br>Initialise 3D simulator with robot *p*<br>$d = 0, s = 0,$ |
| 5. | $if\ no.of\ parts > 15$<br>*fitness = 0* |
| 6. | $else$<br>Convert map into grids |
| 7. | Add obstacles into map and mark corresponding grids as *closed* |
| 8. | Using robot size, mark narrow spaces as *closed* |
| 9. | Solve map for an A-star solution. |
| 10. | $if$ solution does not exist |
| 11. | $fitness = 0$ |
| 12. | $else$ |
| 13. | *for* $t = 0$ to T do: |
| 14. | $if$ goal is reached<br>$fitness = 50 + \frac{100-t}{10}$ |
| 15. | *else* |
| 16. | Calculate robot speed *u* |
| 17. | Use *u* and *t* to locate an imaginary robot following A-star route |
| 18. | Compare distance *d* between actual and imaginary robot $d \mathrel{+}= d$ |
| 19. | *if* robot is stationary, *++s* |
| 20. | *end for* |
| 21. | Calculate distance between robot and goal $d_f$ |
| 22. | $fitness \mathrel{+}= \frac{100-d}{10} + \frac{5(2a-d_f)}{a} + \frac{100-s}{10}$ //$a$ is width of the arena |
| 23. | *end for* |
| 24. | Return best *fitness* |
| 25. | *end procedure* |

step of evaluation while allocating fitness based on multiple priorities. With reference to *Algorithm 2,* while using the proposed A-star based fitness function, the process of evaluating the fitness of each of the evolved robots is as follows: At the beginning of the fitness evaluation simulation, the map of the environment was converted into 5 cm grids as this corresponded to the smallest size of an obstacle. Then the grids that overlapped with obstacles were marked to be excluded from the route calculation. Since the grid size was smaller than the size of the actual robot, to avoid the problem of the A-star algorithm finding a path that cannot be traced by the robot, the robot size was also fed to the algorithm. This

was accomplished by generating a new imaginary boundary around the actual obstacles with the same size as the robot. The result was a map with an area that the robot could transverse. Based on this information, the A-star algorithm solved the map for a possible grid-to-grid solution to reach the goal.

If a solution was found by the algorithm the fitness function moved to the next step, otherwise the process was terminated and zero was returned as fitness. Robots that were not able to move freely to the goal were removed. The fitness function also limited the number of parts in the robot to 15 to avoid evolving robots long enough to reach the goal without moving. The behaviour of the evolved robot was then compared against the ideal route developed by the A-star algorithm. At every time-step, the candidate robot position was converted into grid positions and a corresponding position was identified from the A-star solution. This comparison was performed by dynamically matching the instantaneous speed of a candidate robot with the imaginary robot that used the A-star solution. The distance ($d$) between these robots was then accumulated over the period of simulation (100 s). The accumulated distance was lower if the robot was able to track the A-star solution closely. Along with allocating a fixed fitness for reaching the goal, for evolving faster robots the time taken to reach the goal was also considered through the function $f_{gt}$. To discourage a robot from getting stuck, for every instance of a robot remaining stationary a penalty function $f_t$ was also included. The time taken for the robot to reach the goal was $t$. The final distance between robot and goal was $d_f$, $a$ was the length of the arena and $p$ was the total time the robot was stationary.

At the end of the simulation, a final fitness was calculated using the following formula:

$$f = f_a + f_g + f_{gt} + f_d + f_t \quad (4.1)$$

where,

$$f_a = (100 - \Sigma d)/10 \quad (4.2)$$

$$f_g = 50 \text{ if the goal was reached or } 0 \quad (4.3)$$

$$f_{gt} = (100 - t)/10 \text{ if the goal was reached or } 0 \quad (4.4)$$

$$f_d = 5(2a - d_f)/a \quad (4.5)$$

$$f_t = (100 - s)/10 \quad (4.6)$$

The individual components of the fitness function in *Equation 4.1* (except $f_g$) were normalised between 0 to 10. Therefore, adding maximum permissible individual weights in the equation yielded 90, which was the maximum achievable fitness of any robot. The fitness for reaching the goal was fixed to be 50 and since the combined fitness of the rest of the

**Table XIV. Evolution parameters.**

| Parameter | Value |
|---|---|
| Population size | 40 |
| Number of evolved children | 40 |
| Probability of brain mutation | 0.3 |
| Sigma value of brain | 0.7 |
| Brain bounds | 3:3 |
| Probability of node insertion | 0.3 |
| Probability of sub-tree removal | 0.3 |
| Probability of duplicating sub-tree | 0.1 |
| Probability of swapping sub-tree | 0.3 |
| Probability of node removal | 0.3 |
| Probability of modifying parameters | 0.3 |

weights was less than 50, it became possible to immediately differentiate a robot that had reached the destination. This feature could be useful while making future improvements to the evolution process.

## 4.2. Experiments

During fitness evaluation, every simulated robot was allowed 100 s (due to the complexity of task) to solve the map and arrive at the goal. The rest of the simulation parameters were set to the default values in *Chapter 3*. A complete list of used evolution parameters is shown in Table XIV. To expedite the process, the experiments were performed on a High-Performance Computing System comprising 50 nodes with 20 computing cores in each node. Ten sets of experiments were conducted for each scenario with different seeds from 1 to 10 for the random number generator of the evolver in every iteration.

### 4.2.1. Obstacle-less Navigation

The first experiment was a scenario without obstacles. Each robot was placed in the middle of the arena and was expected to reach the goal *A* at (-1.5, 1.5). At the end of the evolution, both fitness functions were able to evolve robots to reach the goal. Red and green lines to *A*

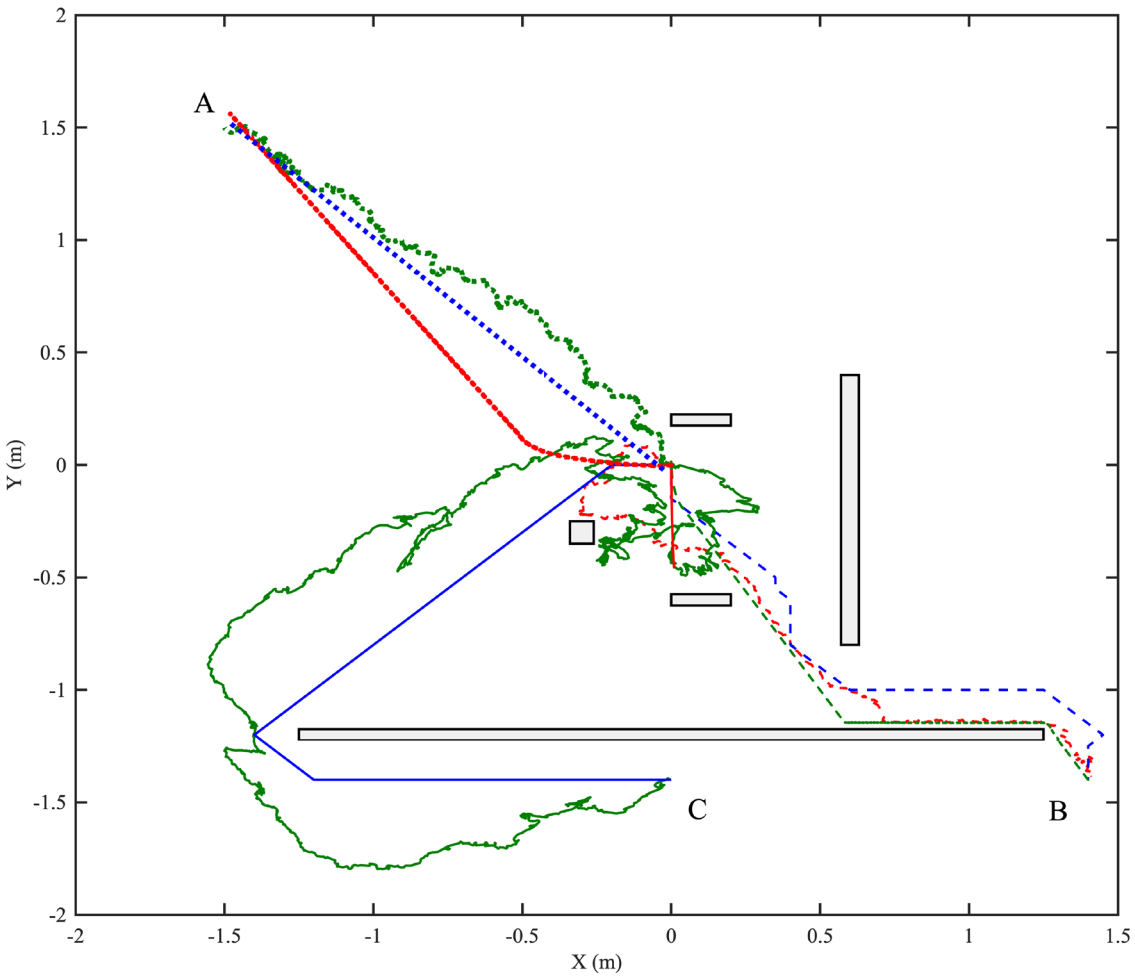



**Figure 16. Robot trajectory and ideal A-star solutions for multiple scenarios.** Ideal A-star solutions are in blue, robot trajectory with A-star based fitness function are in red and basic fitness functions are in green. Routes to A are dotted lines, to B are dashed lines and to C are solid lines. *The short vertical solid red line from centre shows that the robot is unable to reach C. This is despite performing the same experiment multiple times with different seeds to the random number generator of evolver. This could have been avoided by allocating a major share of fitness to the closeness of robot route to A-star solution instead of currently allocated 1/3$^{rd}$ weight in fitness calculation.*

in Fig. 16 plot the route traced by a chosen evolved robot with the basic and A-star fitness functions (for this test, the obstacles in Fig. 16 should be neglected).

### 4.2.2. Navigation with Obstacles

An arena with obstacles was designed (shown in Fig. 16) and robots were evolved to reach *B* at (1.4, -1.4) and *C* at (0, -1.4) in separate experiments. It can be seen from Fig. 16 that the robots were able to reach the destination except when the A-star fitness function was applied to point *C* (solid red colored vertical line in middle of the Fig. 16).

## 4.3. Results

To summarise the results of multiple runs with several goal positions, three different cases are reported as they are sufficient to demonstrate the observed patterns. Through each scenario, the complexity of the task was gradually increased. It was reasoned that *A* should be the easiest to reach as a straight-line solution was available, *B* could be reached without a major change in trajectory and *C* would be the hardest to reach as it was behind a large obstacle. The fitness versus generation curves for the three different destinations are presented in Fig. 17 and Fig. 18.

Generally, the A-star based fitness function was able to evolve successful robots in all except one case when the goal was at point *C.* In the first experiment reaching point *A* the basic fitness calculation was sufficient to evolve satisfactory robots, while the A-star based evolver was able to evolve smaller and faster robots. Both fitness functions needed almost the same number of generations to reach the goal for the first time, but the computation requirements for the A-star fitness function was significantly higher than for the basic fitness function. This resulted in each generation taking longer to evolve when using the former fitness function.

The effect of the initial seed on the evolution process is unclear. While applying the basic fitness function, results from all ten experiments show similar variations and the output matched closely with each other (Figs. 17 (a), (d), (g)). In contrast, the progress of evolution had a much wider spread with the A-star fitness function (Fig. 18 (a), (d)) except for when the evolver was unable to arrive at a satisfactory solution (Fig. 18 (g)). This variable behaviour is due to variation in initial evolution conditions induced by the random number generator. Though reasons for large variations to results is not directly obvious, it can be explained through the concept of the *butterfly effect* [191]. Which describes how minute changes in initial conditions can amount to a large difference in a later state.

The standard deviation of each population was highest when there was a rapid change of fitness (Figs. 17 and 18). Later, the value reduced to zero showing that the average fitness of the population was the same as the best fitness in the population. But the scenario and

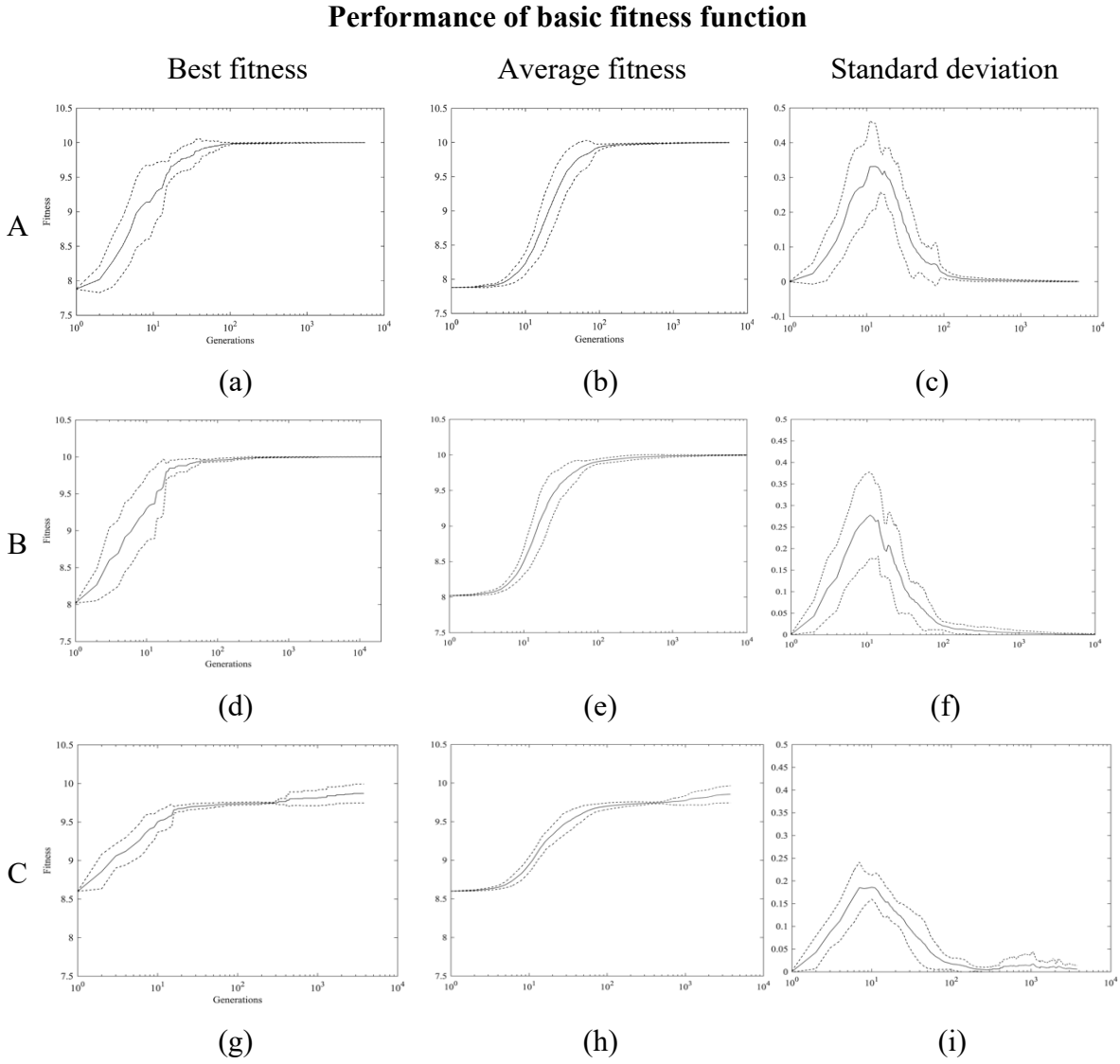



**Figure 17. The progress of evolution over generations (logarithmic scale) while using basic fitness function for ten experiments in each scenario.** The middle line is mean for 10 experiments and the dotted lines are the standard deviation on either side of the mean. (a, d, g): Mean of best fitness measured while reaching points A (a), B (d) and C (g). (b, e, h): Mean of average fitness of entire population over generations for reaching points A (b), B (e) and C (h). (c, f, i): Mean of standard deviation of fitness of entire population for reaching points A (c), B (f) and C (i). *Fitness of 10 shows that the robot reached the goal. Observations from all three scenarios show similar time taken to reach the goal despite exhibiting variation between overall maximum and minimum fitness in each case. An exception is while reaching C when half of the experiments do not converge to a solution in the given timeframe. In these cases, the robots move to the farthest point along the displacement until an obstacle blocked the path.*

fitness function specific increase or decrease rate of standard deviations were identical: this implied that there was a minimal effect of seed once the evolver reached the first possible solution. After this, similar generations were sufficient to allow every individual of the

**Table XV. Comparison of fitness functions.**

| Criteria | Fitness Function Type | | | | | |
|---|---|---|---|---|---|---|
| | *A-star* | | | *Basic* | | |
| | ***A*** | ***B*** | ***C*** | ***A*** | ***B*** | ***C*** |
| Success (%) | 100 | 90 | 0 | 100 | 100 | 50 |
| Worst standard deviation | 26 | 22 | 5 | 0.515 | 0.45 | 0.25 |
| First generation to reach goal | 20 | 21 | N/A | 18 | 19 | 500 |
| Last generation to reach goal | 500 | 5000 | N/A | 90 | 90 | N/A |
| Best fitness | 74 | 72 | 18.8 | 9.98 | 10 | 10 |

N/A- no robot reached the goal

population to reach peak fitness. As evident from Table XV, the standard deviation was higher for a higher distance between goal and origin.

While no robots were able to reach point *C* with the A-star fitness function, five experiments using the basic fitness function generated successful robots (Table XV). The detrimental effect of using a displacement-based fitness function is clear, as in the five non-successful cases, the long plateaus with almost zero standard deviation in Fig. 17 (g) show that robots were extremely close but unable to reach the goal, which was on the other side of the obstacle.

The steps in Figs. 18 (a) and (d) and a sudden variation of standard deviation indicate that one experiment at a time converged to a satisfactory solution. The horizontal overlapping line segments in multiple graphs from Figs. 17 and 18 (e.g. Fig. 18 (i) with overlapping mean and standard deviation lines for majority of evolution) suggest that all ten experiments behaved similarly during that period. While evolving robots for obstacle avoidance, the two key findings were that the controller was unable to perform large direction changes after the first quarter of the fitness evaluation.

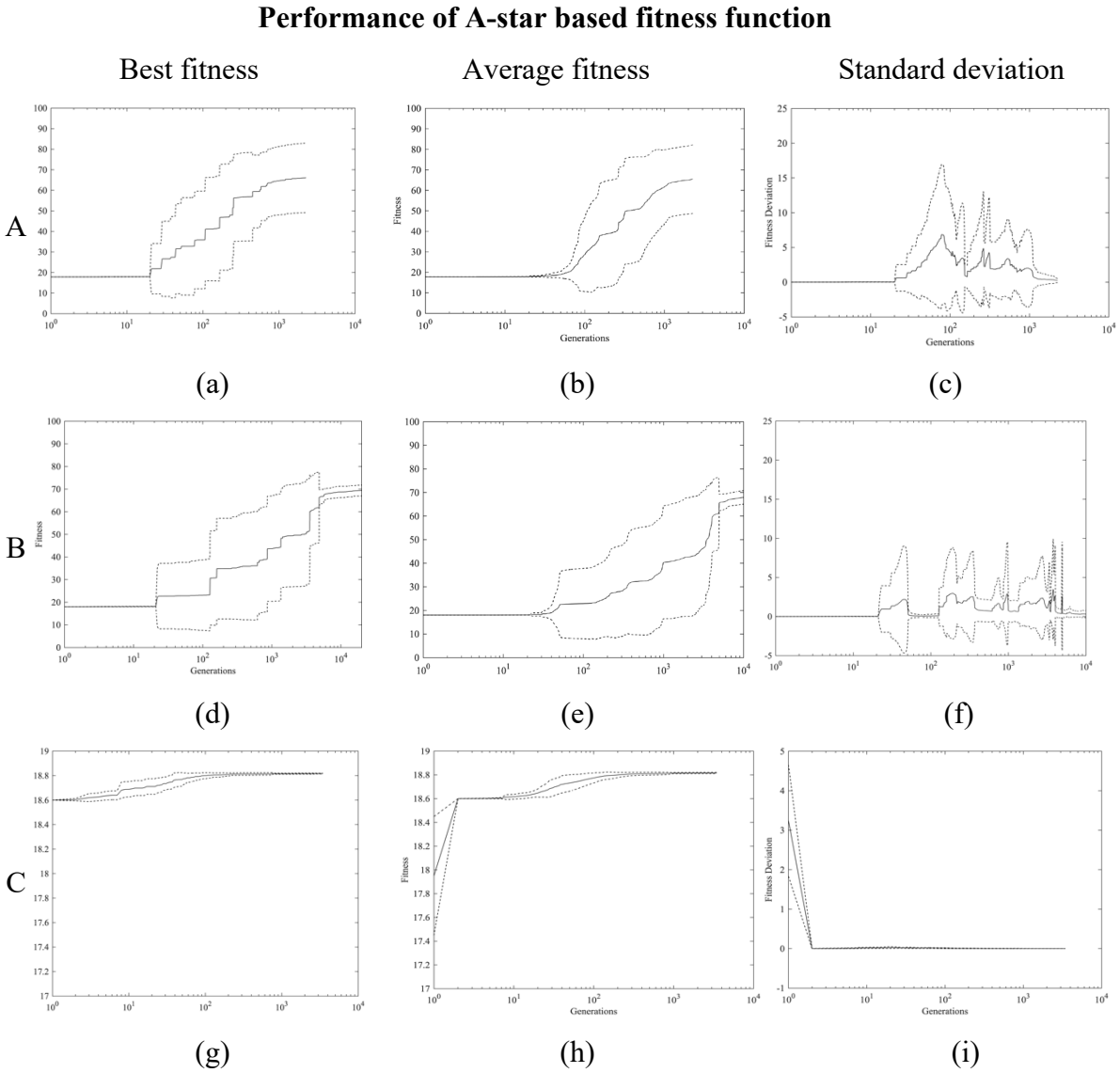



**Figure 18. The progress of evolution over generations (logarithmic scale) while using A-star fitness function for ten experiments in each scenario.** The middle line represents the mean and the standard deviation above and below average is shown by the dotted lines respectively for ten experiments. (a, d g): Mean best fitness measured while reaching points A (a), B (d) and C (g). (b, e, h): Mean of average fitness of entire population over generations for reaching points A (b), B (e) and C (h). (c, f, i): Mean of standard deviation of fitness of entire population for reaching points A (c), B (f) and C (i). *Fitness of 50 shows that the robot reached the goal. The experiments converge to solutions at different times (as shown by standard deviation curves) except while reaching C depending on the chosen seed for the random number generator. This happens due to random initialisation of evolution that can be explained by the concept of the butterfly effect (see glossary for a definition).*

Even when obstacle sensors were present, despite running the evolution process for 37,500 generations, their suitable utilisation was not observed. The change of direction was either the result of collision with obstacles (Fig. 16) or open-loop control without using obstacle sensors or at random. When an obstacle sensor was present, it was not placed in the direction of motion. The problems around obstacle sensors were due to the lack of sufficient encouragement currently given to evolve such behaviours. All this suggested a need for improving the evolution of the controller during the evolution process, investigation of this is detailed in the next chapter. An example of evolution finding a solution that is not desired by the designer was when the evolver generated long robots when it was not restricted in terms of the number of parts a robot could have. This allowed robots to reach the goal with a minimal set of movements.

It was also found that the current weights set for different factors in the fitness function also caused problems. For example, a surprising finding was how the evolver became stuck, by not evolving a robot to move past an obstacle. Particularly when it was trying to reach *C* (short vertical red line in Fig. 16). The initial hypothesis to explain this was that it did not receive sufficient encouragement from the fitness function to move past that point. However, step-by-step analysis of the fitness allocation showed that this was not the case. As a result, the A-star based evolution was unable to generate robots to reach *C* despite running the evolution for the maximum computation time of the High Performance Computing (7 days). A possible solution here could be to allocate more weight to the A-star based fitness than the currently set 1/3$^{rd}$ limit.

In contrast, the basic fitness function needed just 500 generations in the best case and close to 8,000 generations in the worst case to solve the same problem (Fig. 17 (g)). But as seen from Fig. 16 (green line), it could be safely stated that the route taken was not close to that of the optimal solution, and A-star solutions were better in this regard.

## 4.4. Discussion

As the fitness function was designed to help a robot stay on or close to the A-star route, ideally the robot should have been able to make turns when necessary. But this was not observed which questions the effectiveness of the fitness function. To reduce computation

time, the A-star fitness function only solved the map once at the beginning of the evaluation (depending on the robot size). In the next generation when the controller was reused, the A-star solution could generate an entirely different path even when the old path might still be a valid but not optimal solution. Moreover, since the fitness function was fixated on one ideal solution, instead of allowing the robot to focus on reaching the goal it ended up trying to reach the ideal route and ultimately failing both. From another perspective, the fitness calculation did not consider the robot's capability to reach the goal from its instantaneous position. A new A-star solution can exist from the real-time position and this could have been considered.

After solving the map, a robot was discarded if it could not reach the goal because of its large size. From an evolutionary perspective, the robot could have been allowed to survive and allocated a non-zero fitness based on how close it was to the goal. The fitness function only considered the trajectory of the robot to allocate fitness. This suggests the need for incorporating fitness functions which performed an in-depth analysis of both morphology and controller before making a decision. On the other hand, this new method will require considerable computational resources and so might question the need for using EAs.

In all of the experiments, the robot could navigate to a solution by blindly remembering a path to the goal by constantly searching for the best possible fitness at every time-step, or by having a controller react to the environment based on real time feedback from sensors, or by using a combination of both. For example, if it is assumed that the controller was adopting the first option, at the end of simulation the robot could have taken some correct actions from the best solution owing only to the randomly generated neural network. In the next step, when the same robot was chosen as parent, the fitness function would not shed light on which part of the NN was useful in the previous generation, effectively making the evolver perform mutation and crossover blindly, often leading to poorer performance in the children. The long plateaus during evolution and one of the reasons why more than 95% of the works in ER are in evolving controllers rather than in co-evolution could be explained with this.

A common observation in all the experiments above was that the standard deviation of populations reached zero for the majority of the time during long runs of evolution. This suggested that the population became dominated by a small number of (or just one)

solution(s). Even though this is a fundamental problem associated with ER, the fitness function was also a cause. Rather than evolving diverse robots which excelled in different aspects (a robot might have an efficient morphology or superior controller), because every robot was given a single fitness value, differently skilled robots did not progress further. Therefore, parent selection should consider fitness as a group of individual fitnesses corresponding to each skill, and the same information needs to be used while mating. The counter argument here could be that evolution would then be subject to bias and thereby moving away from a true random nature.

Further, the fitness function was not testing the capability of the controller to achieve the task at hand. With the help of fitness functions, the evolution could be sped up by eliminating the robots whose controller did not show any capability of solving the problem. For instance, if the task is obstacle-less point-to-point navigation, the robot just has to make a single turn to the direction of the goal and move towards it in a straight line. This can be accomplished by a simple neural network with a handful of connections. On the other hand, if the task is to handle complex obstacle avoidance, a simple neural network might not be sufficient. A similar approach can also be applied on the morphology by discouraging robots which do not have the potential to solve the problem.

The comparison of results from both fitness algorithms was from the perspective of evolving successful solutions. Therefore, deductions could be further improved by analysing other factors such as trajectories, size, shape, power consumption and by physically testing the best individual from each generation. But this in-depth statistical analysis of features and behaviors would have taken a considerable amount of post-processing. Another possible solution could be changing the linear time independent allocation of fitness to adaptive fitness functions which dynamically change over the lifetime of each robot and at different stages of evolution.

As with results from the previous chapter, the behaviour of evolved robots could have been further generalised by performing more experiments with different simulation conditions. For example, whether using Euclidian fitness function or A-star based fitness function, the focus was just for evolved robots to move from a fixed-point $A$ to $B$. To further generalise the conclusion for mobile robot navigation, multiple points could have been selected. This

was not performed as the evolver was already run 10 times to generalise for one case and a consistent performance was not observed even in those 10 experiments.

The value of using EAs in robotics to find solutions to deterministically solvable problems is an ongoing debate. For instance, it could be argued that instead of using an A-star based fitness function, a robot controller could easily be designed with the A-star algorithm for path planning. However, such an approach only works if the robot body is fixed during evolution as different morphologies warrant different controllers and a single controller approach will not be appropriate. Furthermore, the main premise of ER is to arrive at non-obvious solutions to problems or automatically generate solutions from scratch without human intervention as in the experiments reported till now.

## 4.5. Conclusion

In an effort to co-evolve morphology and controller of robots for navigation and obstacle avoidance, a fitness function based on the behaviour of the robot was compared with a possible A-star solution to the problem. Results for a basic fitness function that simply used Euclidean distance to allocate fitness are mixed. Thereby not giving conclusive evidence to validate the initial hypothesis that aimed at using established task specific algorithms to aid evolution, while applying successful fitness function types currently used outside co-evolution. This also explains why fitness functions in co-evolutionary robotics rely on simpler calculations to arrive at the final fitness. Since the focus was on applying algorithms just for path planning and as co-evolution depends on several other factors which were not considered, the full benefits of the A-star based fitness function were not observed. This points to the need for developing fitness functions derived by combining functionality-based algorithms in the future along with improving other factors of evolution. Finally, several recommendations for designing fitness functions were also discussed. These results along with the findings from *Chapter 3* indicate how the ANN controller is still unable to perform properly by responding to the turning directions given by the fitness function despite when it has the resources to do so. A possible reason is that the ANN is not given enough time to properly learn to utilise the robot's body. Therefore, in the next chapter, efforts are focussed on evolving controllers that could better utilise the available resources.

# Chapter 5.

# The Reinforced Co-Evolutionary Algorithm

The previous chapter indicated concerns associated with evolved controllers. In the co-evolution process an EA simultaneously evolves a mobile robot body and controller. Then a randomly modified or added body part is fitted with a random set of neurons. Additionally, the NN weights are modified and/or new neurons are added during the mutation and crossover operations. This creates a problem, as the controller is not given enough time to improve itself before the next stage of the evolution: if the controller improves during variation operations (mutation and crossover), even a minute change in morphology will immediately impede the progress. This is because for proper function, every morphology warrants a uniquely built controller, which means each controller needs to be individually tested and improved. This is not the case currently.

A second problem is that the newly evolved robots are expected to accomplish the task immediately after they are created. This is in contrast to biological evolution where the suitability of each candidate/organism for reproduction is not fully tested immediately, but usually occurs throughout a learning and maturing processes, mainly through the influence of environmental factors. Furthermore, Polmin [192] observed that almost 50% of psychological abilities of human beings are attributable to environmental influences and the rest are inherited through genetics. If this idea is extended to ER, it would mean that a co-evolved robot that is evaluated as soon as it is created would only be using its inherited skills, and the skills gained as a result of the mutation and crossover operations. This leaves an untapped potential in each of the robots to learn and grow and exhibit a higher performance than demonstrated during the fitness evaluation.

Controller improvement for evolved robots can be achieved through a number of ways. Literature in ER shows that more than 95% of robot evolution is only on the controller. Therefore, there are several possible approaches that can be adopted from the vast already

available literature. Following the findings in the previous chapter, a second method is to analyse each controller and identify its performance based on a fitness function. Based on the results, a new evolution epoch can be initiated to evolve only the controller. Furthermore, the fitness function design can again be considered. However, the unreliable performance of fitness functions in evolution seen in the previous chapter suggests alternate avenues for controller improvement. If an ANN based controller is available on each robot, then an alternate approach is to apply AI/ML methods to improve the controller. Furthermore, these methods have already demonstrated their ability to modify ANNs to suit various applications, especially in robotics. Consequently, different ANN learning methods were explored, and a suitable option was chosen.

The main hypothesis in this chapter is that the introduction of a learning phase for every evolved robot will not only allow it to better utilise the morphology but also provide sufficient environmental feedback to better handle the task. The learning phase will therefore reduce the impact of the random modifications and provide time for each robot to better utilise its functional parts through the controller. This is tested through a learning algorithm that trains each robot before evaluation by the fitness function. A similar work [118] reported in the past year, also attempted to solve the problem of building controllers for co-evolved robots when the morphology is unknown at the beginning of evolution, and this study can be considered as developed in parallel with it. They approached the same problem by developing an evaluation function that balanced between total distance travelled by the robot and its closeness to the required trajectory. Though promising results were generated, their focus was on applying the algorithm on a few selected robots, thereby limiting the potential of the evolver to fully utilise the effects of the learner.

Through the introduction of a learning phase, the following research questions are attempted to be answered:

1. Can the learning phase available to each evolved robot make a positive change to their behaviour and evolution epoch?
2. Can the behaviour of evolved robots be modified to make them properly utilise available sensors? This is not observed in the preceding chapters and literature review.

3. Can general recommendations be made for parameter selection during evolution and learning?

Answers to these questions are expected to be a step towards improving the current state of the art of mobile robot co-evolution. The experimental setup used here remained the same as the previous chapter: The same hardware and software were used, and evolution and simulation conditions applied, (including reusing the virtual map from the previous chapter). When learning parameters or evolution parameters were modified or new parameters were introduced or set, the reasons for these are explained in respective sections.

*Section 2.5* provided a brief introduction to learning algorithms in robotics. Unlike typical learning applications in robotics, designing an algorithm for co-evolved robots is challenging. In the case of Supervised Learning (SL), the input-output mapping is already known through the training data, and the SL algorithms simply have to find a model that fits this mapping. However, in case of co-evolved robots, no such training information was available, as inputs and outputs changed randomly. Therefore, building training data was also impractical. Further, Unsupervised Learning (UL) methods were not applicable due to the lack of training data. In this circumstance, the only available information was the goal specification, or coordinates, that indicated where the robot should reach. Hence, Reinforcement Learning (RL) was chosen, as when compared to Unsupervised Learning and Supervised Learning methods it could work with limited background information by learning through rewards gained.

RL methods are either model-based or model-free. The model-based method was unsuitable as the robot design and environmental dynamics were unknown in advance. In model-free approaches, the policy search-based model-free approaches were preferred over value function-based approaches as they had the option of integrating expert knowledge into policy at the time of initialisation with fewer parameters. The key reason why RL methods excel over other methods is the availability of an exploration phase in the learning process that allows the robot to explore new experiences while performing the task. Even though these principles could have also been applied on evolved robots, for the demonstration of a proof-of-concept policy gradient methods from DS were chosen as they could directly learn the policy without going through the lengthy process of policy exploration, evaluation and update. Among the policy gradient approaches in DS, the chosen version was an online direct gradient-based reinforcement learning algorithm called OLPOMDP [149], as it was a

lightweight algorithm that offered the option of direct policy update at every time-step and was guaranteed to converge. In this particular case, because the evolved robot always had a controller (except in the initial generations), the DS method had the advantage of passing on the learnt policy from one generation to the next.

Even though combining EA and RL methods is not new, with several EA-RL approaches published (see *Section 2.6),* the use of RL for enhancing the effectiveness of EA has been mainly used in constrained applications such as parameter control. The literature also suggests that they have not yet been applied in robotics. Therefore, a new EA-RL hybrid algorithm referred to as a Reinforced Co-evolutionary Algorithm (ReCoAl) is proposed to not only improve the robot co-evolution process, but also to take the algorithm a step closer to the biological evolution and learning paradigm.

## 5.1. The ReCo Algorithm

The Reinforced Co-evolution Algorithm (ReCoAl) designed as part of this study is explained in this section. In normal evolution, a newly created robot has its fitness evaluation as soon as it is created. After the evaluation is completed, the fitness measure is fed back to the evolver for parent selection. Instead in ReCoAl, each robot was subjected to a new learning process before it was placed in a virtual 3D simulator for fitness evaluation. The goal of this learning process was to modify the ANN controller while keeping the morphology fixed for that generation.

An overview of the ReCoAl can be explained with the help of the flowchart in Figs. 19 and 20, pseudo code in *Algorithm 3* showing the entire evolution process and *Algorithm 4* showing the learning and evaluation process for each of the robots. The chosen reinforcement learning algorithm OLPOMDP was integrated into *Algorithm 4*. Here, $z_t$ was the eligibility trace, T was the maximum time allocated for learning, $t$ was the time-step, $\beta$ was the discount factor that increased or decreased the historic memory and $\propto$ was the learning rate. At the beginning of each episode, the simulator was reset, and the robot was initialised with the most recent controller. In the first instance, this was the controller created by the evolver and subsequently it was the modified NN at the end of the most recent learning episode.

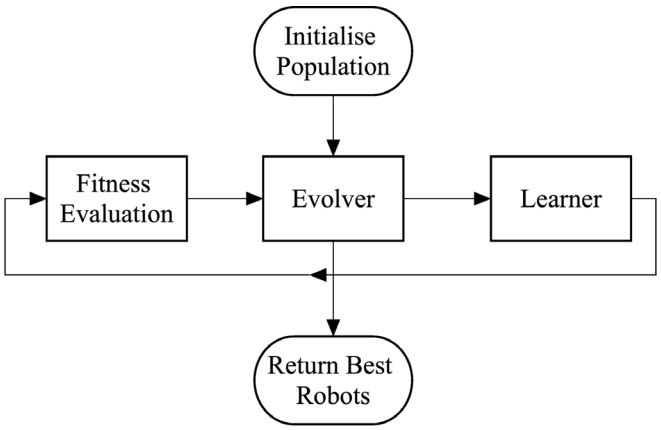



**Figure 19. The ReCoAl architecture overview.** *The conventional evolution process is interrupted by a learner that intercepts robots while they move for fitness evaluation. After the learning process, the new robot is used for fitness evaluation. The updated robot and fitness value are fed back to the evolver for population update and parent selection, respectively.*

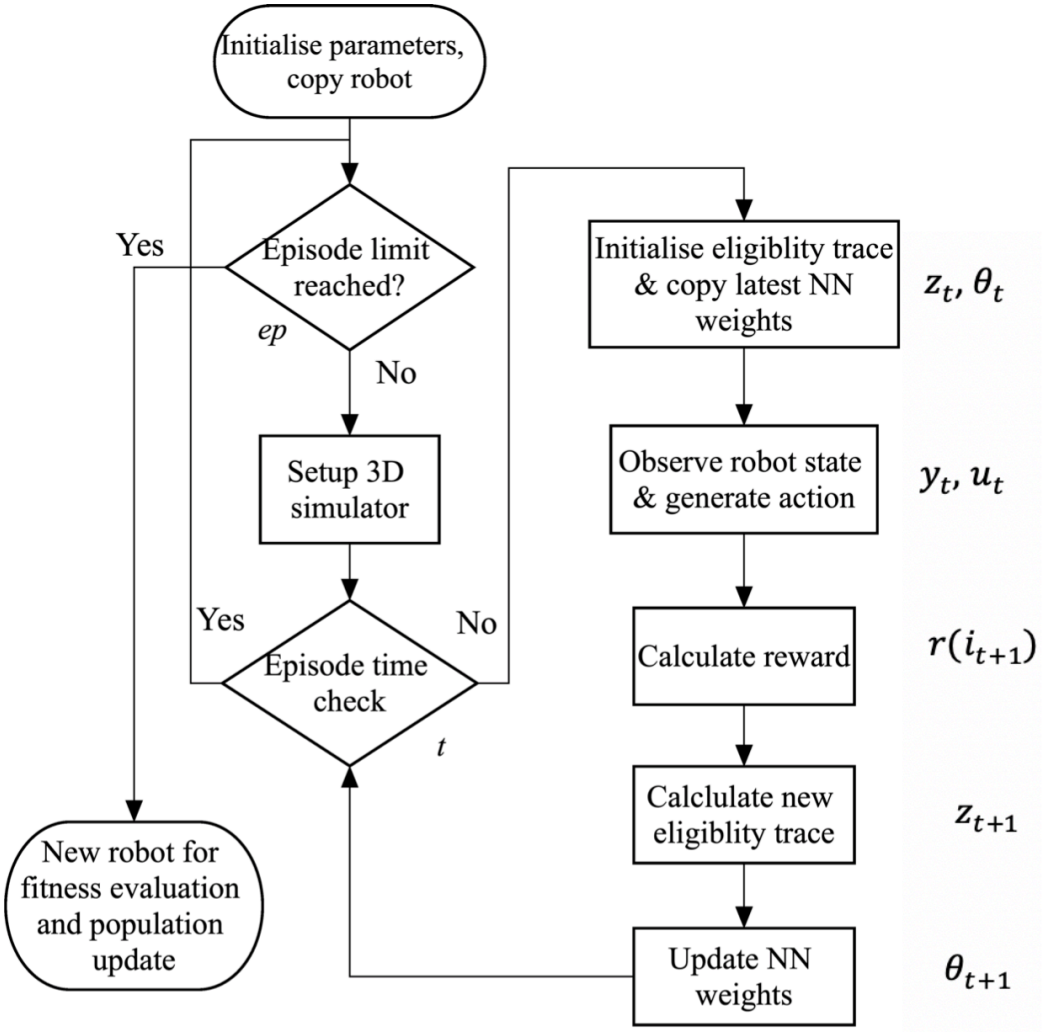



**Figure 20. Learning process in ReCoAl.**

| **Algorithm 3: Reinforced Co-evolutionary Algorithm (ReCoAl)** | |
|---|---|
| 1. | Initialise:<br>$\mu$ random robots in a population, $pop$<br>GEN > 0 //Generations |
| 2. | *for* $gen = 1$ to GEN do: |
| 3. | Compute Evaluate ($p$) for each $p \in pop$ |
| 4. | Rank $pop$ and select parents |
| 5. | Variation operations on parents to produce $\lambda$ children in $pop_c$ |
| 6. | Compute Evaluate ($p$) for each $p \in pop_c$ |
| 7. | Select survivors from combined $pop$ and $pop_c$ |
| 8. | Update $pop$ |
| 9. | *end for* |
| 10. | Return best robot from $pop$ |

| **Algorithm 4: Function Evaluate** | |
|---|---|
| 1. | *procedure* EVALUATE ($p$) |
| 2. | Initialise: //Learner initialisation<br>EP >0 //Episodes<br>T > 0 //Episode length in seconds<br>Set $\beta, \alpha$ //Learning parameters |
| 3. | *for* $ep = 1$ to EP do: |
| 4. | Initialise:<br>Initial state $i_0$<br>Initialise 3D simulator with robot $p$ |
| 5. | $if\ ep = 0$<br>Copy parameter values received from evolver $\theta_0 \in R^K$ |
| 6. | $else$<br>$\theta_0 = \theta_{ep-1}$ |
| 7. | Set $z_0 = 0$ ($z_0 \in R^K$) |
| 8. | *for* $t = 0$ to $T$ do: |
| 9. | Observe state $y_t$ |
| 10. | Generate control action $u_t$ according to current policy $\pi(\theta, y_t)$ |
| 11. | Observe the reward obtained $r(i_{t+1})$ |
| 12. | Set $z_{t+1} = \beta z_t + \frac{\nabla \pi_{u_t}(\theta, y_t)}{\pi_{u_t}(\theta, y_t)}$ |
| 13. | Set $\theta_{t+1} = \theta_t + \propto r(i_{t+1}) z_{t+1}$ //Update policy |
| 14. | *end for* |
| 15. | Return $\theta_{ep} = \theta_{t+1}$ |
| 16. | *end for* |
| 17. | Initialise: //Fitness evaluation initialisation<br>TR > *0* //Trials |
| 18. | *for* $tr = 1$ to TR do: |
| 19. | Initialise 3D simulator with robot $p$ |
| 20. | Update weights from $\theta_{ep}$ |
| 21. | Evaluate *fitness* |
| 22. | *end for* |
| 23. | Return best *fitness* |
| 24. | *end procedure* |

Learning started by observing state $y_t$ (as in *Algorithm 4.9*[3]*)*, which was the sensor readings from the simulator. This data was fed into the NN policy which generated a control action $u_t$ for the motor. The learner then allocated a scenario dependent reward based on the state. The eligibility trace was then computed with the policy gradient, $\beta$ and the old eligibility trace (*Algorithm 4.12)*. The policy was finally updated and the NN weights were adjusted through, $\theta_{t+1} = \theta_t + \propto r(i_{t+1})z_{t+1}$.

To compute the error in *Algorithm 4.12*, a standard backpropagation mechanism was applied. Depending on the kind of neuron used (simple, sigmoid or oscillator), the error signal was calculated and backpropagated into the network to obtain $\frac{\nabla\pi_{u_t}(\theta,y_t)}{\pi_{u_t}(\theta,y_t)}$. The servo motor's output control signal was then compared with the expected control signal during the error calculation.

With reference to a possible FNN as seen in Fig. 21, the generalised error calculation equations could be derived through the following steps: The error at output layer could be represented as,

$$E = \frac{1}{2}\sum_{n}(t_n - y_n)^2 \qquad 5.1$$

where $n$ is the number of motor neurons and $t_n$ is the target signal. With $k$ the index of output neurons, $j$ the index of the hidden neurons, $i$ the index of the input neurons, $x_k$ the activation signal input at the $k^{th}$ neuron, $y_k$ the output signal at the $k^{th}$ neuron and $w_{jk}$ the weight connecting the $j^{th}$ hidden neuron to the $k^{th}$ output neuron (see Fig. 21), then the error differentials at output $\frac{\delta E}{\delta w_{jk}}$ can be written from *Equation 5.1* as:

$$\frac{\delta E}{\delta w_{jk}} = \frac{\delta}{\delta w_{jk}}\left(\frac{1}{2}\sum_{n}(t_n - y_n)^2\right) \qquad 5.2$$

[3]*Line 9* in *Algorithm 4.*

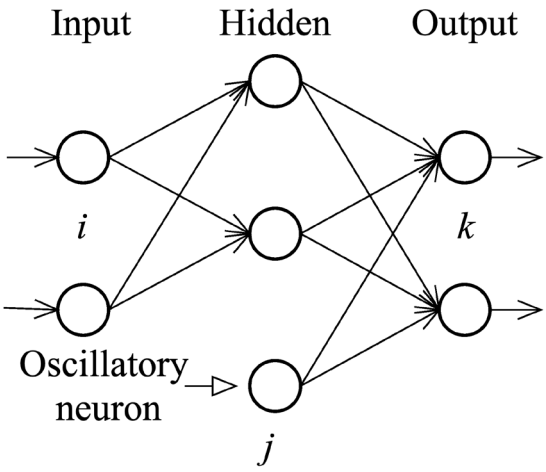



**Figure 21. The feedforward NN architecture used.** *i, j, k* are indexes of input, hidden and output neurons, respectively. *The oscillatory neuron does not accept connections from the preceding layer but only feed forward to other layers. Input neurons have linear activation function, hidden and output neurons have sigmoid activations if they are not oscillatory neurons.*

Using chain rule,

$$\frac{\delta E}{\delta w_{jk}} = \frac{\delta E}{\delta y_k}\frac{\delta y_k}{\delta x_k}\frac{\delta x_k}{\delta w_{jk}} \tag{5.3}$$

To find out each of the derivatives above, the activation signal $x_k$ is,

$$x_k = \sum_j w_{jk}\, y_j \tag{5.4}$$

where $y_j$ is the output at $j^{th}$ hidden neuron and

$$\frac{\delta x_k}{\delta w_{jk}} = y_j \tag{5.5}$$

As the output neuron always has a sigmoid activation function,

$$y_k = Sigmoid(g_k(x_k - b_k)) \tag{5.6}$$

here $g_k$ and $b_k$ are the *gain* and *bias,* respectively and

$$Sigmoid\ (z) = \frac{1}{1 - e^{-z}} \tag{5.7}$$

By using the identity,

$$\frac{\delta(Sigmoid\ (z))}{\delta z} = Sigmiod(z)\big(1 - Sigmoid(z)\big) \tag{5.8}$$

Partial differential of *Equation 5.6 w.r.t* the activation signal gives,

$$\frac{\delta y_k}{\delta x_k} = (g_k y_k(1 - y_k)) \tag{5.9}$$

For calculation of $\frac{\delta E}{\delta y_k}$,

$$\frac{\delta E}{\delta y_k} = \frac{\delta}{\delta y_k}\left(\frac{1}{2}\sum_n (t_n - y_n)^2\right) \quad 5.10$$

and simplifying this yields,

$$\frac{\delta E}{\delta y_k} = y_k - t_k \quad 5.11$$

The *Equations 5.11, 5.5* and *5.9* can be then substituted in *Equation 5.3* as,

$$\frac{\delta E}{\delta w_{jk}} = -g_k y_k y_j\,(t_k - y_k\,)(1 - y_k) \quad 5.12$$

The *Equation 5.12* is the gradient of weights connecting the output neurons. Similarly, the gradient of weights of neurons originating from the hidden neurons can be calculated through the following equations:

$$\frac{\delta E}{\delta w_{ij}} = \frac{\delta E}{\delta y_j}\frac{\delta y_j}{\delta x_j}\frac{\delta x_j}{\delta w_{ij}} \quad 5.13$$

$$x_j = \sum_i w_{ij}\, y_i \quad 5.14$$

$$\frac{\delta x_j}{\delta w_{ij}} = y_i \quad 5.15$$

$$y_j = Sigmoid(g_j(x_j - b_j)) \quad 5.16$$

$$\frac{\delta y_j}{\delta x_j} = (g_j y_j(1 - y_j)) \quad 5.17$$

Using the chain rule for $\frac{\delta E}{\delta y_j}$ results in:

$$\frac{\delta E}{\delta y_j} = \sum_k \frac{\delta E}{\delta y_k}\frac{\delta y_k}{\delta x_k}\frac{\delta x_k}{\delta y_j} \quad 5.18$$

The *Equations 5.11* and *5.9* correspond to the first two terms in *Equation 5.18*. To obtain $\frac{\delta x_k}{\delta y_j}$,

$$x_k = \sum_j w_{jk}\, y_j \quad 5.19$$

$$\frac{\delta x_k}{\delta y_j} = w_{jk} \quad 5.20$$

Therefore,

$$\frac{\delta E}{\delta w_{ij}} = g_j y_i y_j(y_j - 1)\sum_k w_{jk} g_k y_k\,(t_k - y_k\,)(1 - y_k) \quad 5.21$$

The *Equations 5.12* and *5.21* were used to calculate the gradients in the NN. These gradients and the rewards were responsible for the step-by-step updating of the NN weights. The $(t_k - y_k)$ term was the error at each output neuron.

*RoboGen* does not incorporate wheels into the robot's structure, hence only oscillatory gaits were allowed. As a result, oscillatory locomotion was induced by servo motors. Therefore, servo motors were expected to rotate to 90° or -90°. So, the target signal was set as,

$$t_k = \begin{cases} 0, & y_k > 0.5 \\ 1, & y_k \leq 0.5 \end{cases} \quad 5.22$$

When oscillatory neurons were present in evolved ANN, instead of passing on the input signal, it only added on a sinusoidal signal to the next layer as per *RoboGen*. Therefore, the gradient calculation at the oscillatory neuron's incoming weights were not performed.

The steps in *Algorithm 4.8* to *4.14* were repeated until 10 seconds were elapsed after which a new episode was initiated with the updated neural network from the last step. Each episode time was set to a low value when compared to the actual fitness evaluation as the expectation here was to just let the NN learn to use the available resources and not solve the task. The process was repeated for a predefined set of episodes after which the robot was tested with the fitness function. The results and the robot were then fed back to the EA for updating the population and parent selection.

*Bootstrapping* is a well-known problem in ER in the early generations of evolution. Here it meant that the robots were evolved with zero weights in the NN. When this happened, the learning process could not continue as the gradient would always be zero. In such cases, the learning process was skipped, and the robot moved for standard fitness evaluation to save computational resources and time.

## 5.2. Experiments and Results

The experiments discussed below were designed to assess the performance of ReCoAl. Using a scenario-based reward and the error measured at the outputs, NN weights were updated in every time-step. The functionality of *RoboGen* was modified to add the additional learning phase into the simulator engine. A new simulator first initialised the environment with the same parameters as the fitness evaluation simulator engine in *Chapter 4* (unless

specified below). After the learning process was completed, the new controller was passed on to the fitness evaluator where the evolved robot was updated with the new controller and evaluation took place. Later, instead of simply sending back the fitness value to the evolver for parent selection, the updated NN weights were also passed back through the communication channel. These new weights were then added onto the corresponding robot in the population.

### 5.2.1. Reward Functions

The experiments were run with scenario specific rewards, of which the different types were presented in *Section 2.5.4*. The scenarios were designed to investigate navigation with obstacle avoidance. Two different types of reward allocation methods are proposed for each of the scenarios. Both of them have exhibited success in RL applications in robotics. In the first, reward *r* was allocated at every time-step given as,

$$r = -\sqrt{\left(x_g - x_t\right)^2 + \left(y_g - y_t\right)^2} \tag{5.23}$$

where $(x_g, y_g)$ were goal coordinates and $(x_t, y_t)$ were the real-time coordinates of the robot. Therefore, the reward approached zero as the robot got closer to the goal. Furthermore, this Euclidian distance-based reward function can work well when there are no obstacles, but may not work in complicated arenas. In the other tested reward function, the reward *r* was allocated as,

$$r = \begin{cases} 0, & |\sigma| < 10° \\ -1, & 10° < |\sigma| \leq 45° \\ -10, & |\sigma| > 45° \end{cases} \tag{5.24}$$

where $\sigma$ was the steering angle (in degrees) needed to ensure that the robot faced towards the goal. When the direction of movement was between 0° and 10° of the goal, 0 was allocated, while it was between 10° and 45° of the goal -1 was allocated and in all other cases, -10 was given.

Sometimes, the robots are generated with distance sensors, but these have been shown to be not used properly in evolved robots. Therefore, in navigation tasks, the direction of movement was forced to be in the direction of this selected obstacle sensor or sensors. When the robot moved in the direction of the sensor/s, the sensor reading would become meaningful as it was combined with back propagation with the aim to help the controller make better decisions. This was accomplished by using the reward in *Equation 5.24* with $\eta$

(instead of $\sigma$), the steering angle (in degrees) needed to make the robot turn in the direction the distance sensor was facing:

$$r = \begin{cases} 0, & |\eta| < 10° \\ -1, & 10° < |\eta| \leq 45° \\ -10, & |\eta| > 45° \end{cases} \quad 5.25$$

When there was more than one obstacle sensor, the preference was given to the sensor closest to the direction of movement. Sensors in the frontal plane of the robot were also preferred as the movement of the robot would be in the plane perpendicular to the frontal plane of the robot.

To test the performance of ReCoAl, a number of experiments were designed with the used experiment setup in *Chapter 4*. This setup also aided in a direct comparison between results from both these chapters. Therefore, the ReCoAl that was integrated into *RoboGen* was run to evolve robots to reach point *A* at $(1.4, -1.4)$ in the same virtual arena as in the previous chapter. Evolver and simulator parameters also remained almost the same throughout and the fitness function applied was the simple *Euclidian* distance-based from the last chapter. Three different reward functions as mentioned previously were also tested one at a time in each of the experimental conditions.

The new learning process warranted four main parameters that needed to be set for every experiment. They were; the learning rate, discount factor, number of episodes and episode length. While the discount factor was set at 0.95 (as recommended by OLPOMDP's creators), literature indicated there was no consensus on the value of the learning rate. Therefore, in most of the experiments conducted, multiple learning rates of $\propto = 1,\ 0.1,\ 0.01, 0.001, 0.0001$ were set to account for both ends of the spectrum. Episode length was first set at 10 s which was a low value as compared to the fitness evaluation time of 100 s. The expectation here was that each robot should be able to learn to use available resources in that 10 s and later complete the task in the longer fitness evaluation. However, based on observations in the experiments below, the episode length was later modified to 50 s and 100 s in certain cases. Beginning with the length of episodes set at 10,000 in the initial experiments, it was later reduced to 3000 to balance between run time and performance. From combinations made with functions and parameters discussed above, a number of experiments were performed. Experiments were first conducted with the default NN controller available in *RoboGen* instead of the FNN type explained in *Section 5.1.* The

difference between the FNN type and the inherent ANN type used in *RoboGen* is in the way neurons are wired. In the default case, every neuron is connected to every other neuron except for feedback connections at the input layer where there is no feedback. It must be noted that this is unlike an RNN architecture where along with a feedforward connection, a self-feedback connection is also present in hidden layers (see *Section 2.3* for the default ANN architecture in *RoboGen*). Even though it was theoretically possible to calculate a gradient at every NN weight in such a network, it was not attempted due to the mathematical complexity and consequently lengthy calculations required for gradients estimation.

To simplify the process, instead of calculating the error for every NN weight, the backpropagated error at every neuron was calculated as detailed in [193] where an absolute error at the output was injected into the other neurons. The target output signal in *Equation 5.22* was also modified to generate 1 at output neurons because an oscillatory motion was already available from the feedback mechanism at every neuron. This error was then used in *Algorithm 4.12* to calculate the eligibility trace.

As reported in *Section 5.2.2* below, two experiments were conducted, the first with *Euclidian* type reward function and then with the reward function to enable obstacle sensors. Results indicated that rewards did not converge in either of the cases. This suggested problems with the chosen learning process and ANN controller type combination. Therefore, ANN type was modified to FNN and a new learning process as explained is *Section 5.1* was conducted.

Experiments with the new FNN type controller were first performed with the same parameters set in RNN type controller experiments. They were evolving robots to reach point $A$ at $(1.4, -1.4)$ with 10 s episode lengths for 10,000 episodes using multiple learning rates. With these settings, performance of all three reward functions were one by one evaluated. Results from each of the experiments were also studied to tune parameters in the next set. Later, the effect of learning rate and episode length on evolution was also studied in an effort to further improve the evolver's performance. Episode lengths of 10 s, 25 s, 50 s, 100 s were paired with learning rates of 1, 0.01 and 0.0001 to identify the relationship between both these parameters. Finally, ReCoAl's performance was compared with a standard EA while evolving robots to reach point $A$.

### 5.2.2. Experiments with a Recurrent Neuron-Based Controller

#### *5.2.2.1. Robots evolved to reach point A with 10 s episode and Euclidian rewards*

Modified algorithm from *Section 5.1* was applied to evolve robots to reach point *A* with coordinates $(1.4, -1.4)$ in the same arena as *Chapter 4*. Multiple values of learning rates were used ($\propto = 0.1, 0.01, 0.001, 0.0001$), along with a discount factor of $\beta = 0.95$, and an episode length of 10 s to allow each robot to pass through a learning phase of 10,000 episodes. A simple Euclidian reward function (*Equation 5.23*) was applied. Fig. 22 shows the performance of the learning algorithm. The y axis is the average of rewards received by all of the best robots (irrespective of generation) in every episode. In an ideal case, the rewards received need to move from large negative values up to 0 (a badly performing robot would have accumulated greater negative rewards). It can be seen that the rewards are not converging towards zero. When $\propto$ is set to 0.1 (in blue), it demonstrated that the rewards were collectively moving in the *wrong* direction. Despite the improper performance of the learners (Fig. 22), the best robots were able to get closer to the goal (Fig. 23). Fig. 24 shows the individual learning performances of the best robots from each generation. It can be seen how the learner was not able to induce a positive change in the NN.

#### *5.2.2.2. Robots evolved to reach A with 10 s episodes and rewards to use obstacle sensors*

Here the reward function from *Equation 5.24,* was applied to encourage the robots make use of distance sensors (when available). The results indicated consistent oscillatory rewards received throughout the learning process over the entire evolution. This can be seen in green and black oscillatory curves (Figs. 25 and 26). The oscillations suggested that the changes induced by the learning algorithm were insufficient to generate any noticeable modifications to the robot behaviour at the output.

These findings are consistent with the literature which shows that even a simple RNN network that just has one self-feedback connection per neuron (and only at the hidden layer) is notoriously slow to converge while using backpropagation algorithms due to the residual nature of feedback signal. That is, the output at each neuron depends on all other outputs generated from previous time-steps. This is the primary reason for their unpopularity and why NNs like LSTMs (Long-Short Term Memory RNN) are developed to make the learning process manageable. Therefore, in all of the experiments below, the default NN in *RoboGen* was converted into a FNN.

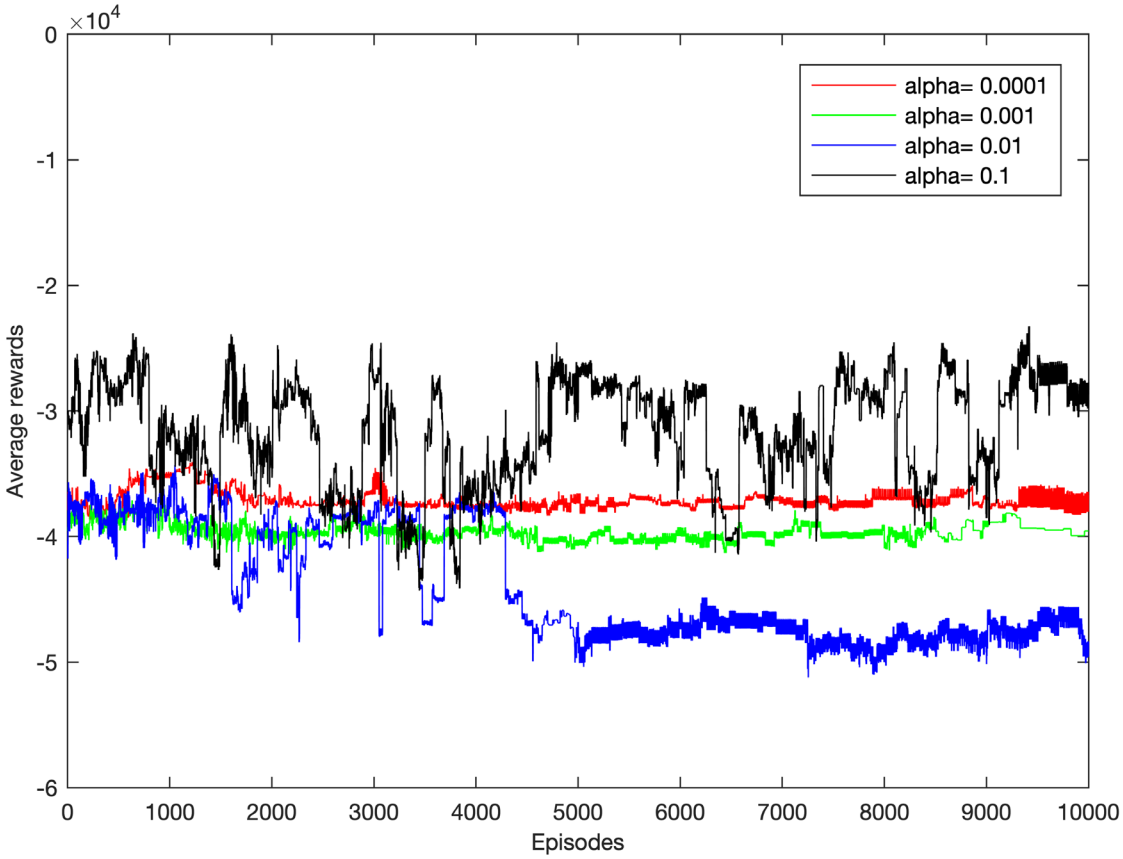



**Figure 22. Cumulative rewards received by the best fit robots averaged over the total generations versus episodes when using recurrent neuron controller.** Each episode was 10 s, $\beta = 0.95$ and multiple values of $\propto$ (see legend) were used for enabling the robots to reach *A*. The rewards shown on the y-axis represent an average of the rewards obtained in every episode by the best fit robot in every generation. In this case, about 40 rewards were averaged. Rewards selected were negative of the Euclidian distance to the goal. *All four curves show random behaviour and are not trying to converge towards zero. In an ideal case, the reward curves need to move up towards 0 which is the best possible reward. Rewards are negative values with the most fit robot having the smallest negative value.*

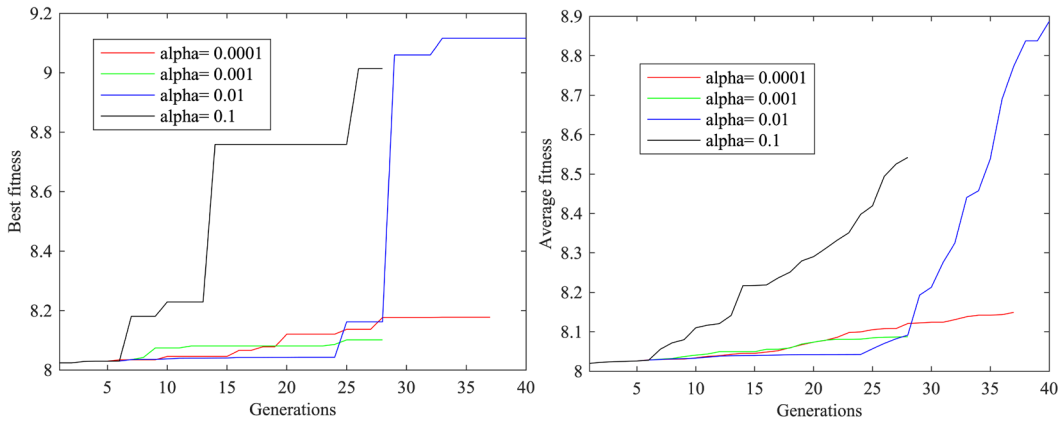



**Figure 23. Best fitness and average fitness of population while using recurrent neuron controller**. Learning algorithm was tested for multiple $\propto$ and evolving robots to reach point A. *Both graphs indicate improvement in performance of the evolver over time with* $\propto = 0.0001$ *displaying the best performance. Not all evolvers completed 40 generations as maximum computation time was exceeded.*

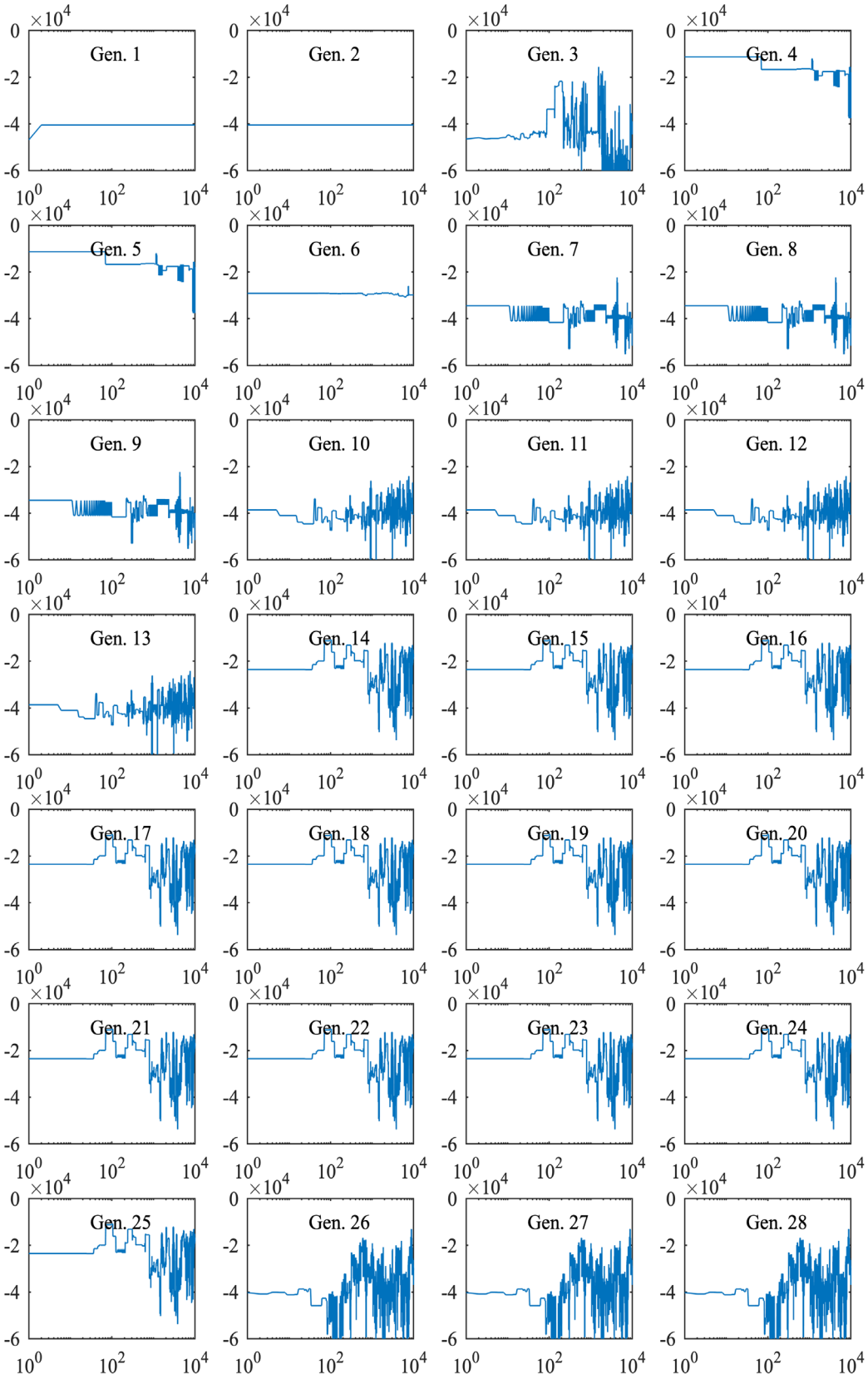



**Figure 24. Cumulative rewards vs episodes (in logarithmic scale) of the best fit robot for every generation.** Individual robot performance refers to experiment in Fig. 22 when $\propto = 0.1$. *Flat lines show no change in rewards accumulated at the end of every episode over learning. When a properly converging algorithm is applied, the rewards should gradually move towards zero (though not seen above due to problems with the learner).*

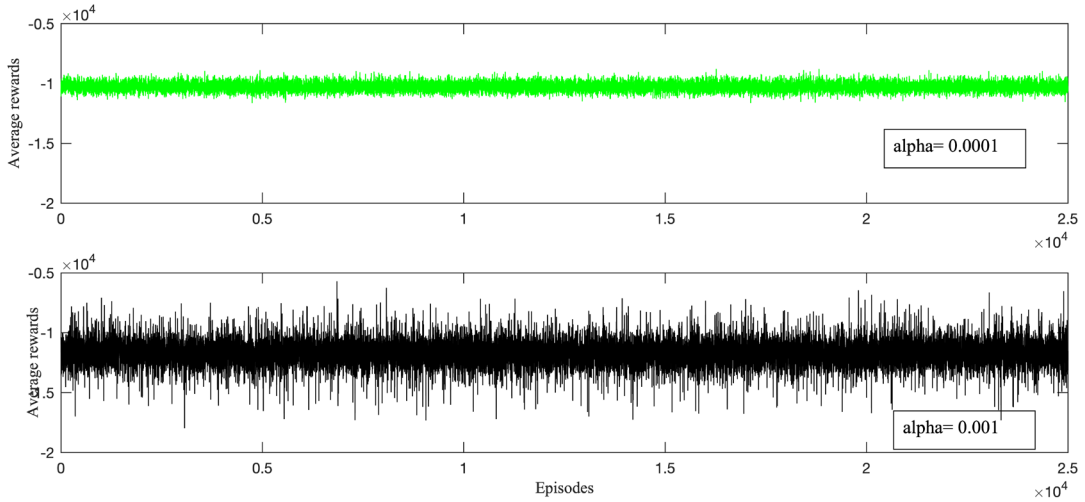



**Figure 25. Averaged episode specific cumulative rewards.** Reward function encouraged the use of distance sensors in the recurrent neuron-based controller. All other parameters remained the same as in the experiment shown in Fig. 22. *Both sets of experiments show the ineffectiveness of the reward function. The noise like lines for both alpha values indicate that the learner is not able to suitably modify the NN weights to direct the rewards towards 0.*

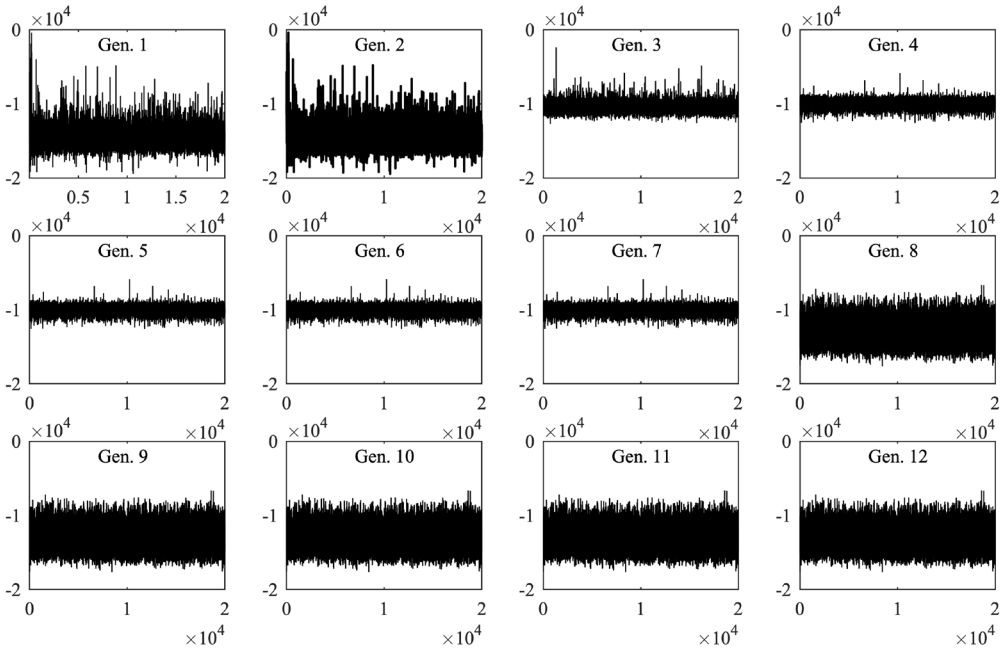



**Figure 26. Cumulative rewards vs episodes.** Learning algorithm performance for best fit robot in each generation. Robots were trying to reach A. The reward function encouraged the use of distance sensors. The graphs shown correspond to experiments from Fig. 25 with $\propto = 0.001$. *Consistent noise-like graph during individual robot learning confirms that the learner does not function as planned.*

### 5.2.3. Experiments with a FNN-Based Controller

The results for the experiments in *Section 5.2.2* indicated an unsatisfactory outcome due to the simplified backpropagation algorithm used for the eligibility trace ($z_t$) calculation. Therefore, the NN in *RoboGen* was converted to a FNN and the backpropagation calculations explained in *Section 5.1* were performed to measure the error at every neuron-neuron weight instead of every neuron.

To help the new robots start moving at the beginning of evolution, an oscillatory neuron was placed in the hidden layer of the NN. Since the oscillatory neuron did not accept any input from the preceding layer, five sigmoid neurons were also added to the hidden layer for accepting inputs from the always present IMU sensor which generated six sensor outputs at the input layer in the FNN. These additional neurons also helped to avoid the bottleneck that could lead to underutilisation or ignoring of input signals. That is, if there were no hidden neurons other than an oscillatory neuron, the input signals would never reach the output layer in an FNN architecture as the input neurons were not connected to the oscillatory neuron. Furthermore, when there were several input and output neurons with a few hidden neurons, there would be a higher chance for the generation of poor control signals due to the data lost in the hidden layer (the data would be lost because of a many-to-one-to-many connection).

With this new FNN controller, the same set of experiments as presented in the previous *Section 5.2.2* were performed.

#### *5.2.3.1. Robots evolved to reach point A with 10 s episodes and Euclidian rewards*

In the first set of experiments, the robots needed to reach *A* with 10 s learning time in each of the 10,000 episodes. For each learning rate, five experiments were performed with the seed to the random number generator varied from 1 to 5. Fig. 27 shows the learner's performance in the whole evolution process, while Fig. 28 plots individual robot's (seed = 1) performance. Both Figs. 27 and 28 show that the learning rate of $\propto = 0.0001$ obtained the maximum positive rewards (red curve) and how the same experiment (with $\propto = 0.0001$) was the fastest to reach the goal (red curve in Fig. 29). It must also be noted that setting the learning rate to 1 made the system not responsive to rewards and so not able to make any significant changes to the NN. Consequently, future rewards were also affected, as indicated

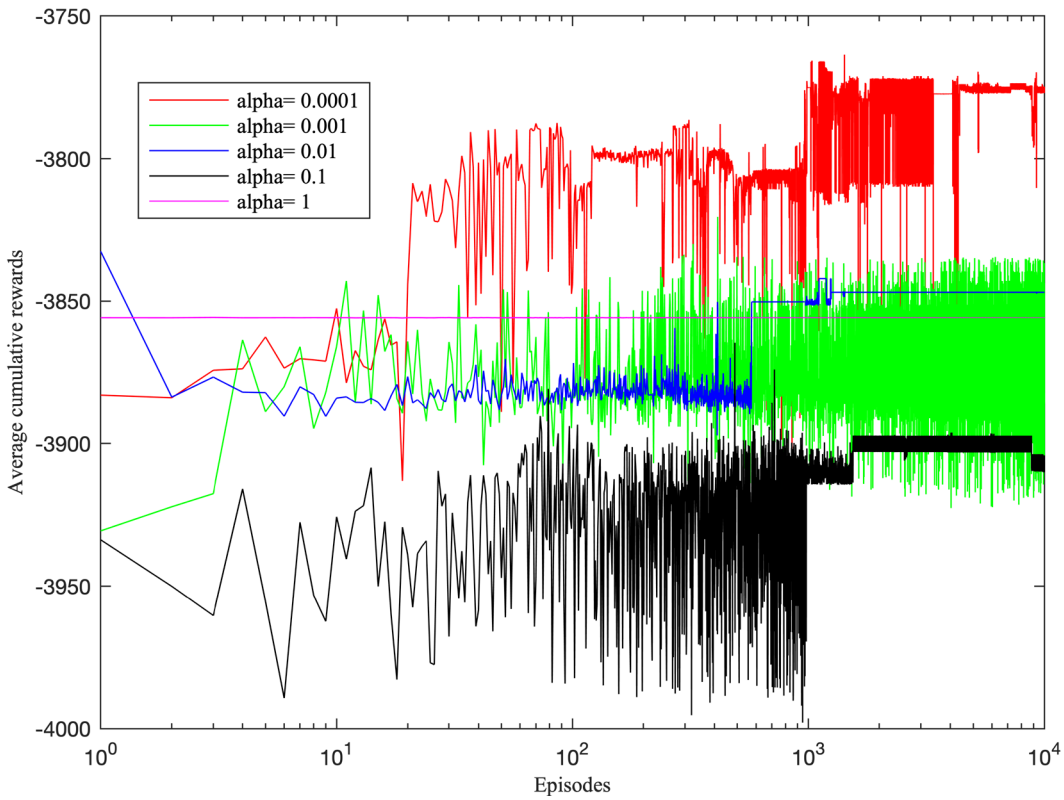



**Figure 27. Averaged rewards vs episodes (logarithmic scale).** Multiple $\propto$ used as shown in the legend, Euclidian distance-based reward function was applied, each episode was 10 s, robots tried to reach point *A* and seed was 1. *As in previous experiments, rewards need to show a movement towards 0 to demonstrate positive learning. Except when* $\propto = 1$ *(straight line in magenta), experiments with all other* $\propto$ *values show some level of positive learning.* $\propto = 0.0001$ *appears to converge faster in all of the generations. The oscillations indicate that either most of the robots' rewards in each episode oscillate or there is a large deviation in the rewards for multiple robots from each generation in each episode.*

by the magenta straight line in Fig. 27. This line therefore demonstrates that a higher learning rate means a slower response of the learner, which eventually results in a slower evolution (also shown by magenta curves in Fig. 29). An interesting observation was that the rewards obtained tend to gradually create unstable behaviour as the robots progress through different episodes (green curves, Fig. 27). The same can also be observed by green curves ($\propto = 0.001$) from generations 9 to 100 in Fig. 28. The behaviour also resulted in very slow progress of evolution as shown by the green curves ($\propto = 0.001$) in Fig. 29.

The trajectories of the best robots from multiple generations before and after they undergo the learning is shown in Fig. 30. The red lines show how each robot moved before learning,

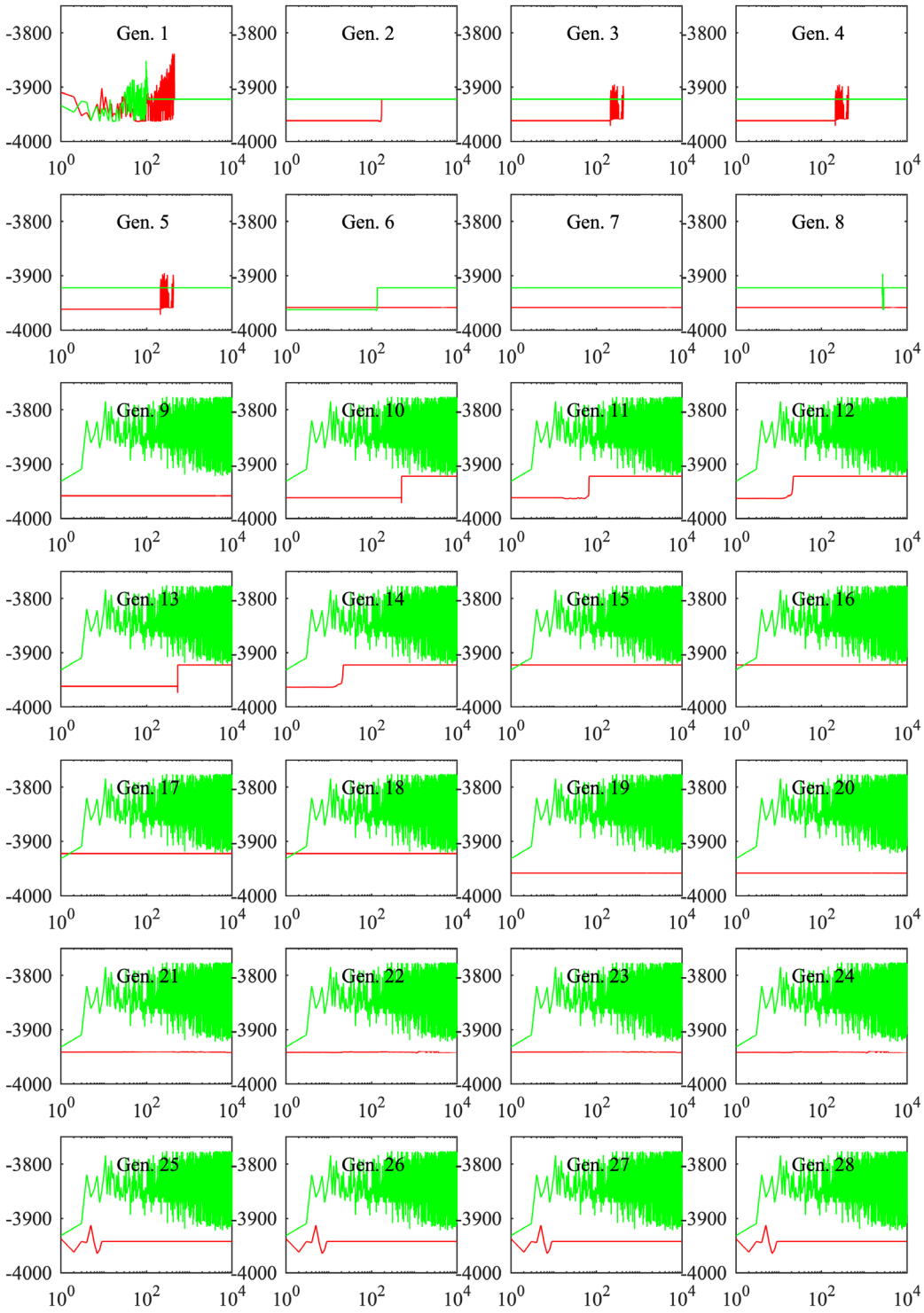



**Figure 28. Part 1. The variation of rewards received by the best robot in every generation vs episodes (logarithmic scale).**

and blue lines show the route taken after learning. The favourable effect of the learner to make the robot move was apparent in all of the trajectories. They also demonstrated how the robot was able to make direction changes to move towards the goal.

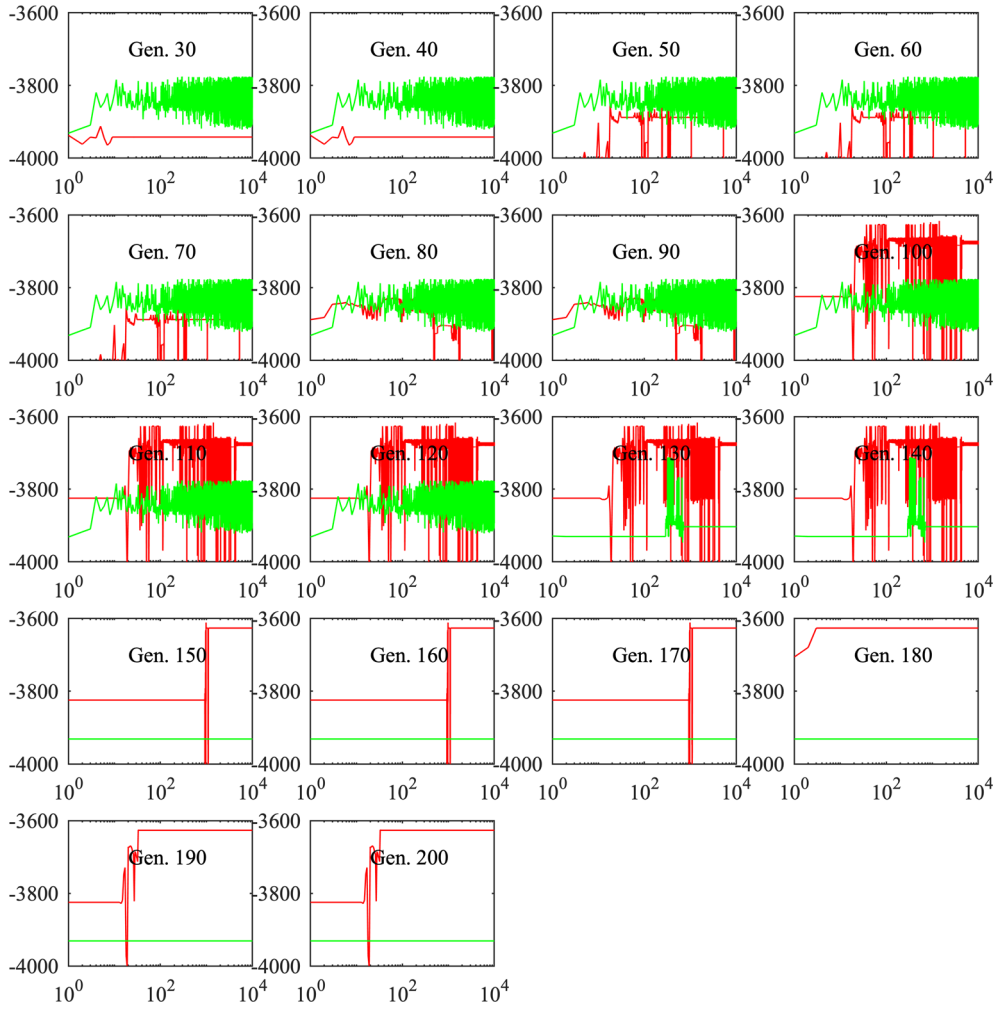



**Figure 28. Part 2. The variation of rewards received by the best robot in every generation vs episode (logarithmic scale).** The experiments shown are from Fig. 27. Experiments with $\propto = 0.0001$ is shown in red and $\propto = 0.001$ is shown in green. *Both red and green curves initially show positive progress. However, the diverging behaviour shown by the green curves in the later episodes leads the evolver to slow down the performance (Fig. 29).*

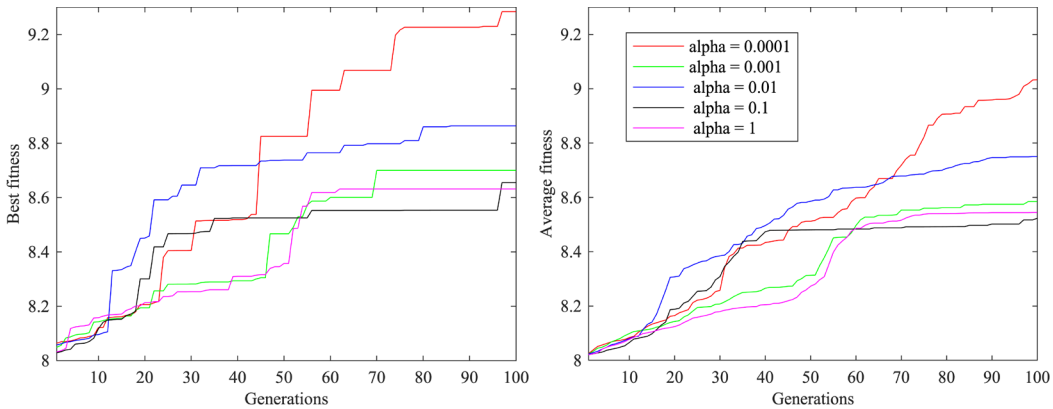



**Figure 29. Averaged progress of evolution from five experiments while generating robots that can reach point *A*.** Same parameters as used for Fig. 27. *A fitness of close to 10 shows that robots reached the goal. When* $\propto = 1$ *(magenta lines), though the speed of evolution is fastest in the initial generations, the evolver stops making changes to the best robots in later generations which shows that the learner is negatively influencing evolution. Consistent with Fig. 27,* $\propto = 0.0001$ *shows the best overall performance.*

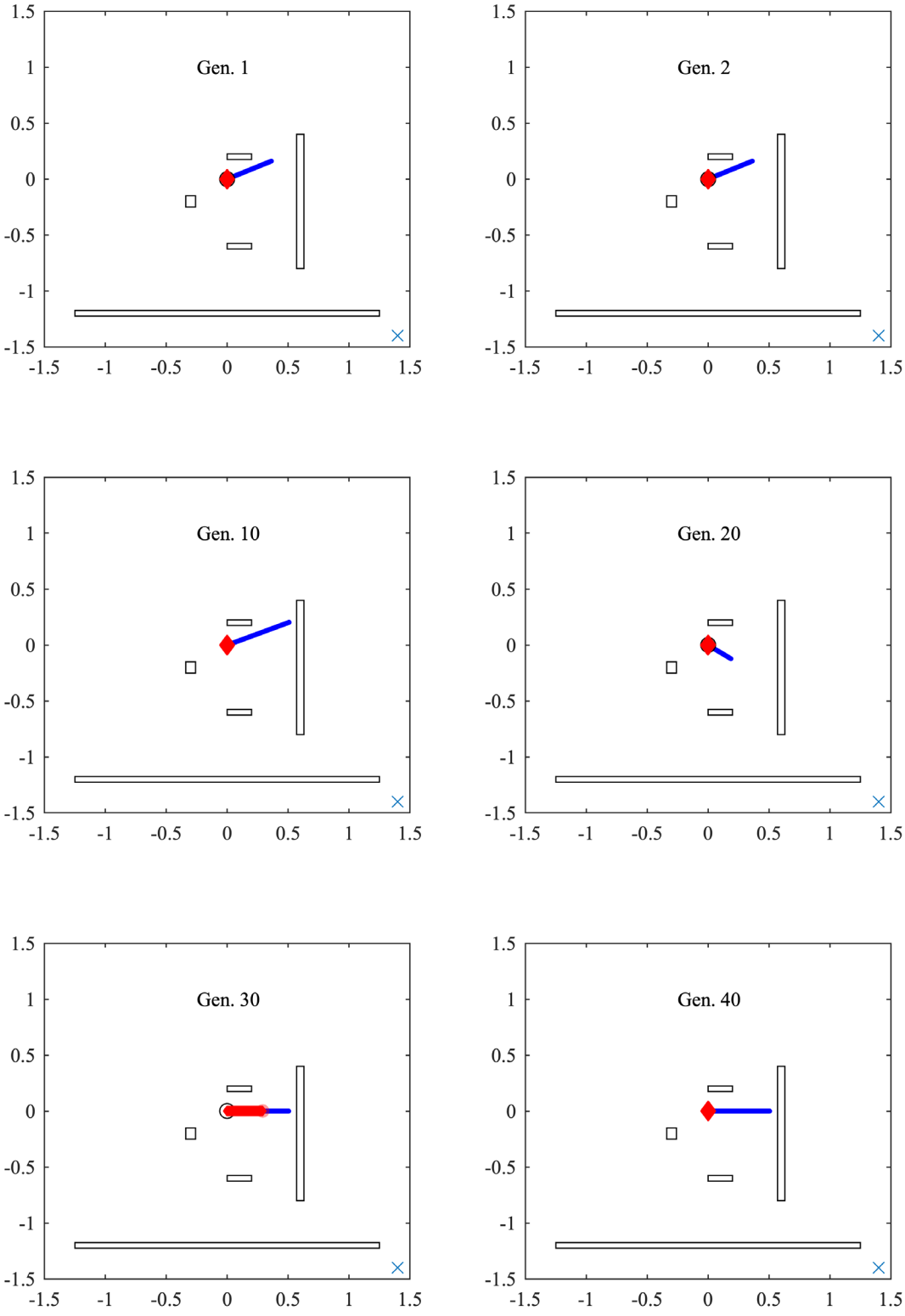



**Figure 30. Part 1. Robot trajectories before and after learning.**

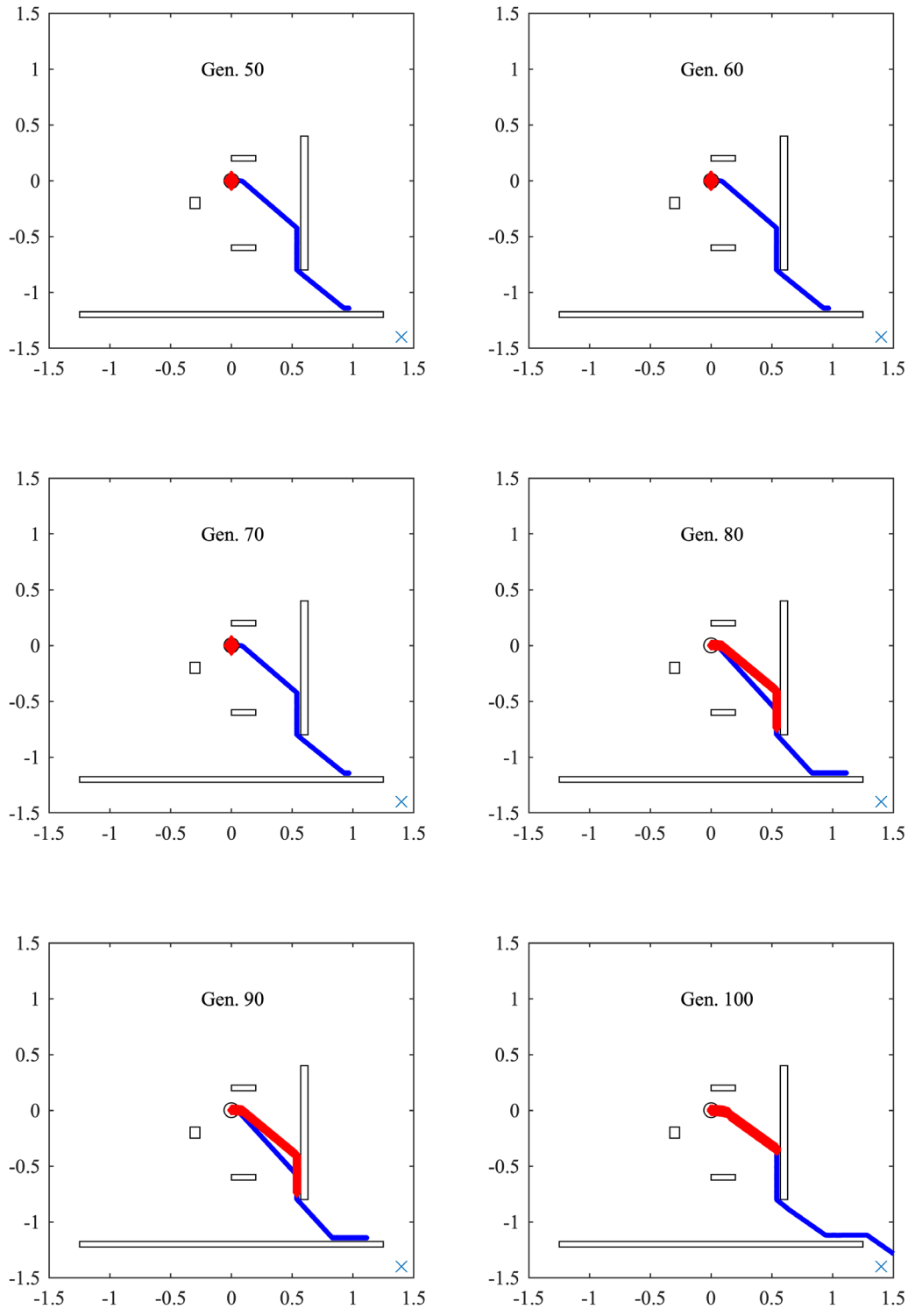



**Figure 30. Part 2. Robot trajectories before and after learning.** Corresponding best robots over multiple generations from the experiment in Fig. 29 ($\propto = 0.0001$) were tested with NN weights before learning (with red lines) and trajectory of the same robot after learning is shown in blue. Robots needed to move from the centre to A at (1.4, -1.4) (cross). *Red dots seen in many of the images indicate robots remaining stationary. The progress shows how robots first learn to move then slowly turn towards the goal. The direction changes are visible in the later generations as the learner (in Gen. 50 and 60) is able to make initially stationary robots head in the direction of goal as per feedback from the reward function.*

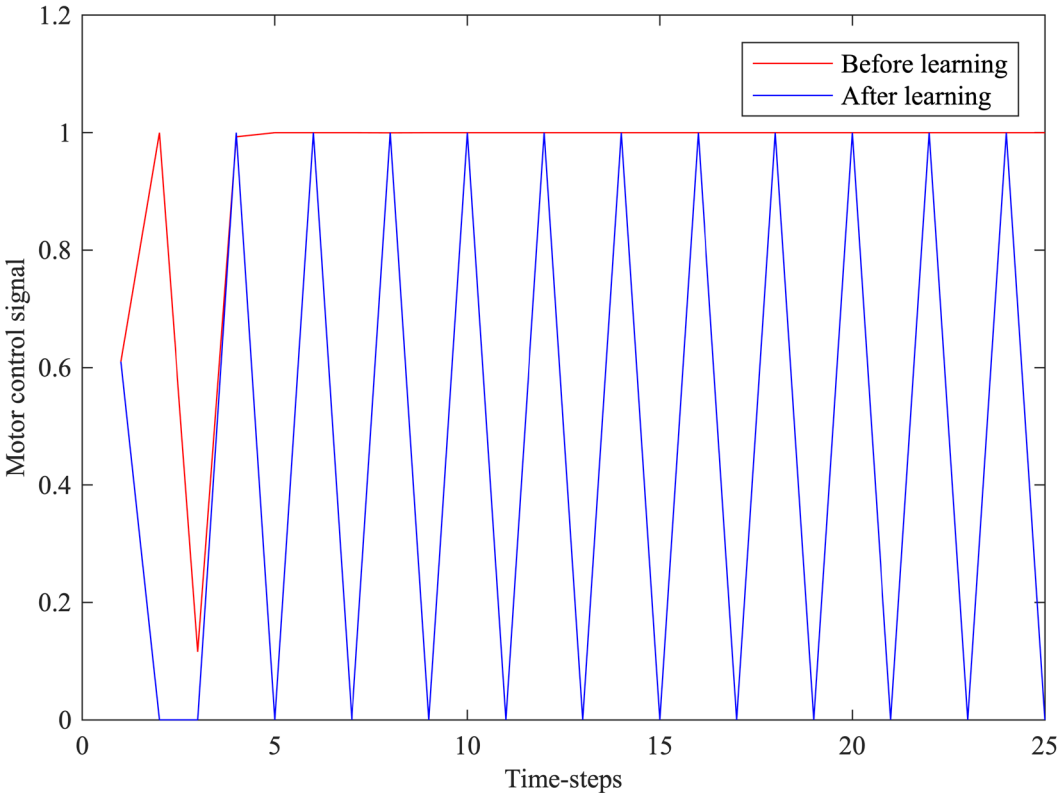



**Figure 31. The effect of ReCoAl on an individual robot.** The motor input signal is plotted over time for the best robot in generation 20 as shown in Fig. 30 and Fig. 32. *The red lines indicate that the motor signal does not change after the initial 5 time-steps. After the learning process is completed, the robot's ability to move by generating an oscillatory movement is evident from the triangular waveforms that oscillate between 0 and 1. These oscillations continue over the entire simulation process.*

Fig. 31 shows how the learner was able to make the robot use the motor by generating alternate 0 or 1 signals necessary for an oscillatory movement. The motor control signal corresponds to the robot in Generation 20 from Fig. 30. As seen in Fig. 31, the control signal before learning (red line) fluctuated between 0.6 and 0.1, while finally settling at 1 after 4 time-steps. On the other hand, after 4 time-steps, the new learnt controller was able to precisely switch between 0 and 1 at the output. The robot morphologies in Fig. 32 indicate that the evolver only modified the robot body four times in the entire evolution run. The size of the robot helps to further emphasise the effect of the learner to generate trajectories in Fig. 30 as the algorithm was able to modify the controller to utilise the morphologies of robots with just three parts. This is contrary to the results generated before as previously, better performing robots were generally longer.

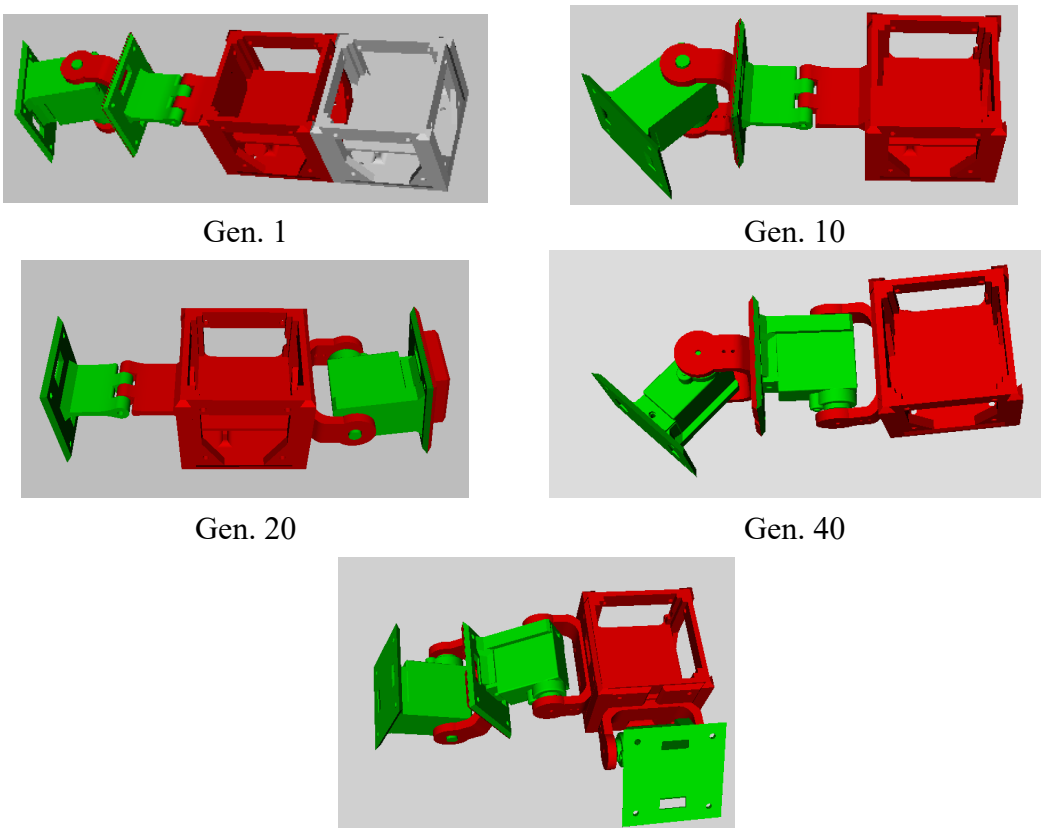



**Figure 32. Physical representation of best evolved robots in each generation while using ReCoAl**. The robots correspond to the experiment in Fig. 28 ($\propto = 0.0001$) and the trajectories of Fig. 30. *Images show how they are often built with a handful of parts. Despite this limitation, the algorithm does manage to teach the robots to reach the goal by Gen. 100.*

The magenta lines in Figs. 27 and 29 indicate that higher learning rates reduced the speed of the evolver. Therefore, to find out if this was indeed due to the learner, the best robots in every generation were analysed from experiment with its seed equal to 1. It was found that the best robots only changed once over the entire epoch despite the evolver generating ten new robot children during every generation. This demonstrated that none of the evolved children in all of the subsequent generations were able to match the performance of their ancestor from generation 4 which brings into question the learner's effectiveness. The trajectories of those best robots are in Fig. 33 and the corresponding robots are shown in Fig. 34. The motor control signals plotted in Fig. 35 indicate how the learner was trying to drive the control signal over the maximum and minimum permissible limit of 0 and 1.

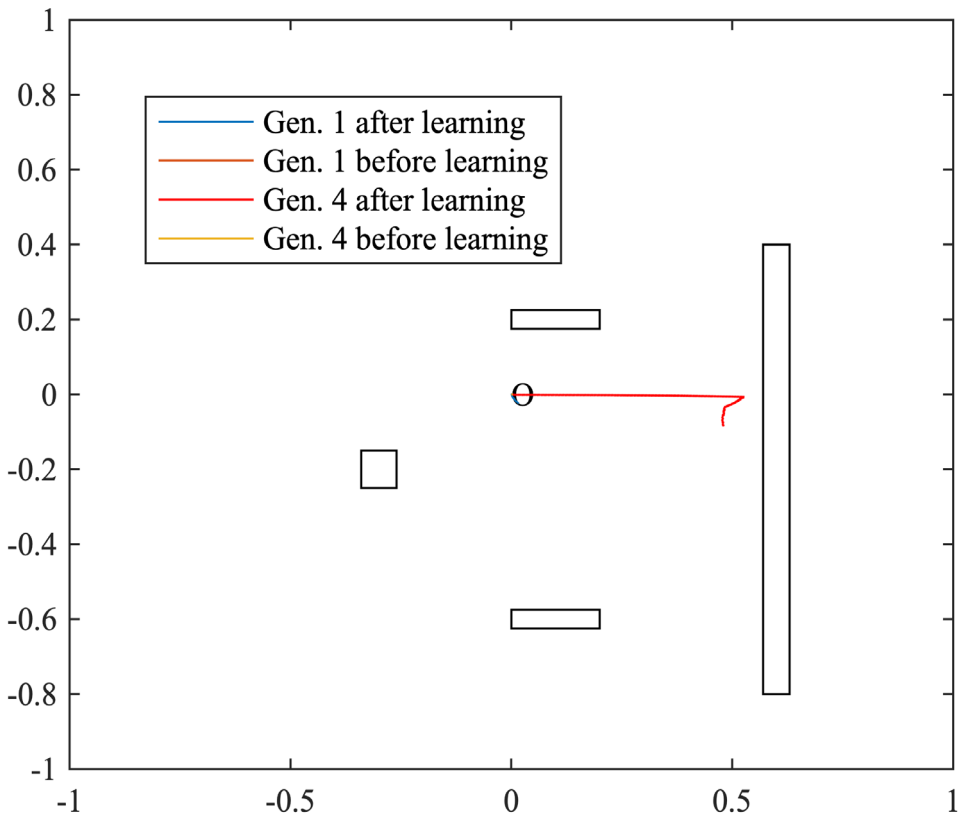
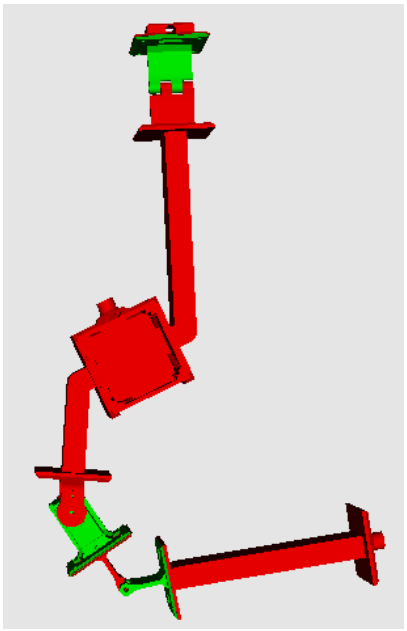
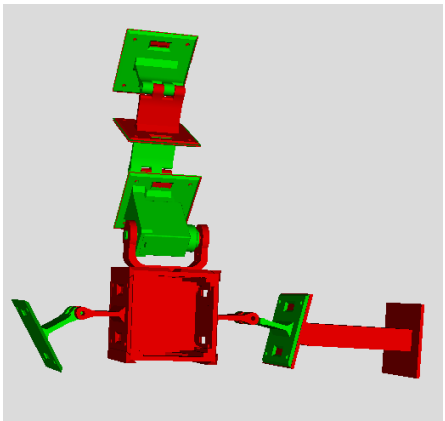



**Figure 33. Behaviour of robots when taught with a high learning rate.** The robots correspond to experiments in Fig. 27 ($\propto = 1$). *The red line demonstrates how only the learnt robot in generation 4 is able to move, even then it does not reach the goal. None of the other colours are visible as the robots fail to move in those cases.*



Gen. 1



Gen. 4

**Figure 34. Best robots in corresponding generations over the entire evolution.** Robots shown are from Fig. 33. *Only one change in morphology of best robot is recorded for the entire run.*

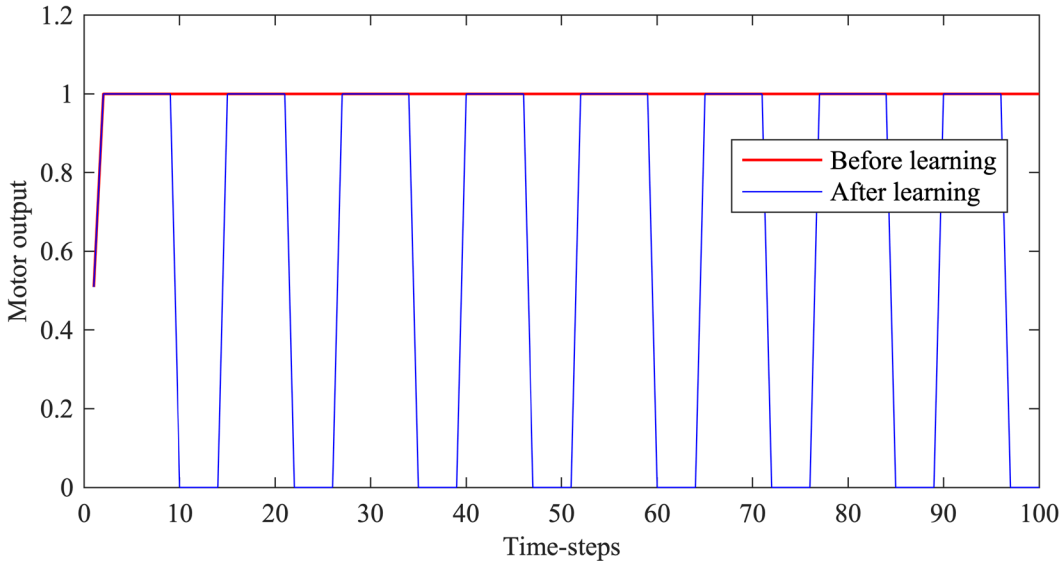



**Figure 35. Motor control signal when taught with a high learning rate.** Signal shown is of the best robot in Gen. 4 from Figs. 33 and 34. *Intermittent flat peaks of blue suggest that the learner is generating a control signal higher than the maximum permissible values. However, the effect of the learner is still visible as a flat red line before learning is converted into an oscillating signal.*

#### *5.2.3.2. Robots reaching point A with 10 s episodes and rewards to encourage the use of distance sensors*

To encourage the use of distance sensors, the Euclidian reward function was replaced with the reward function in *Equation 5.24*. All other parameters and settings remained constant during these experiments. The results obtained are displayed in Figs. 36 and 37. Curves in Fig. 36 show the unresponsiveness of the system (except with a learning rate of 0.01). All other curves demonstrate very minute changes throughout the episodes. The trend of lower learning rates corresponding to faster evolution was also observed similar to previous experiments in *Section 5.2.3.1* (Fig. 37).

#### *5.2.3.3. Robots reaching point A, 10 s episodes and (0, -1, -10) rewards*

To check if the lack of progress in the previous experiments was due to an inability of the reward function to make a difference, the same reward values of (0, -1, -10) were applied to again evolve robots that could reach point *A*. While $\propto = 0.001$ showed positive overall changes (black curves in Fig. 38), as in all previous experiments the oscillatory behaviour persisted.

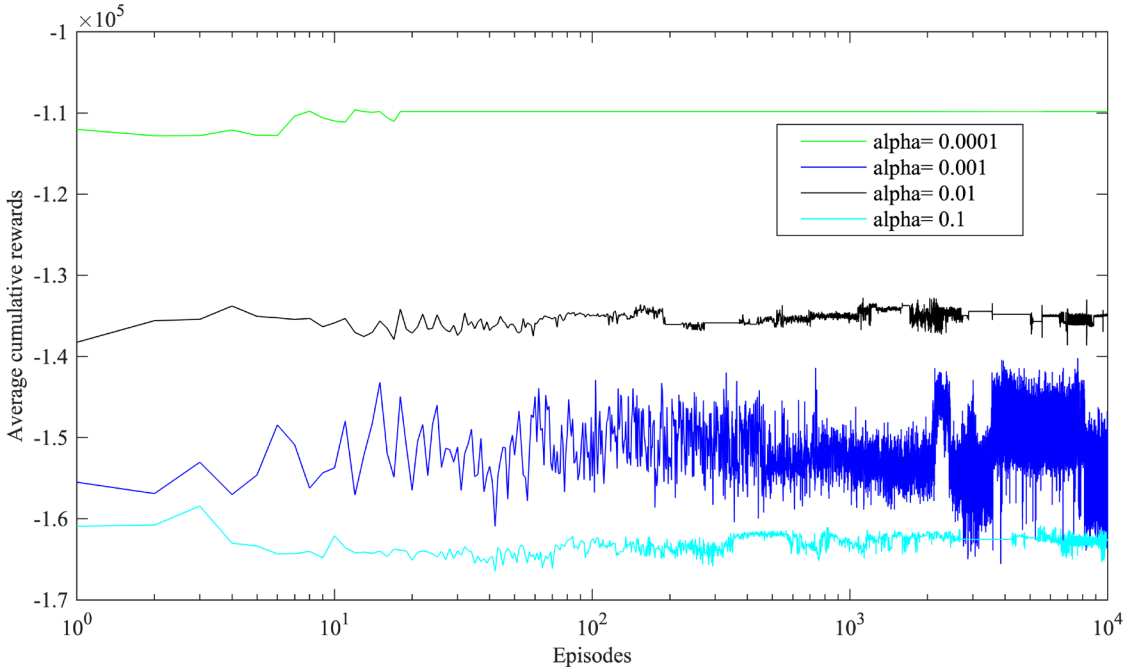



**Figure 36. Rewards vs episodes (logarithmic scale).** Experiments try to force the use of distance sensors through a reward function, while using the same settings from experiments in Fig. 27. *Nonconvergence of the curves shows that the reward function is unable to teach the robots use its sensors.*

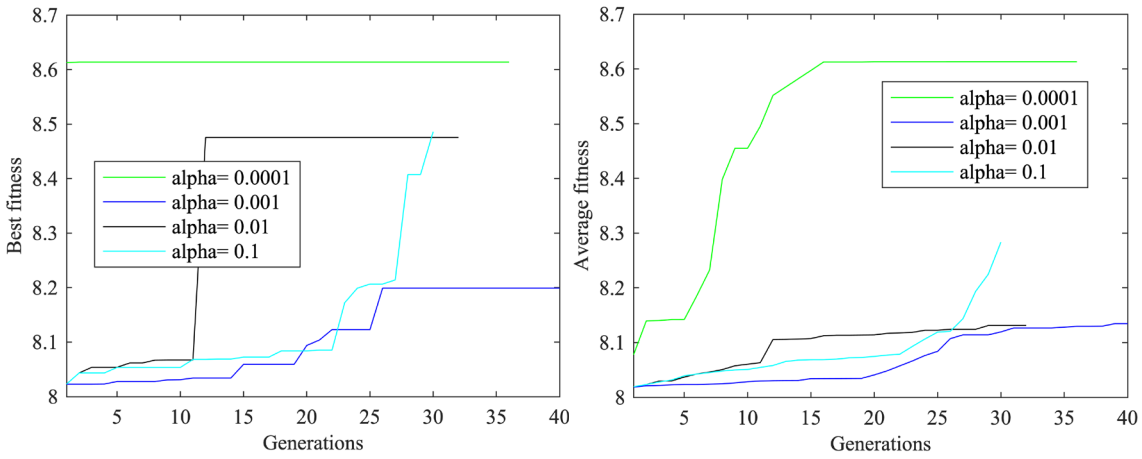



**Figure 37. Progress of population when robots are trying to reach point *A* and use distance sensors.** The performance corresponds to experiments in Fig. 36. *The best fit robot using* $\propto = 0.0001$ *does not change over the entire evolution and starts at a much higher fitness compared to normal values. This shows the unintended effect of the reward function to evolve better fit robots in the first generation as the best robots in this case do not change over the entire course of evolution. The green line in Fig. 36 is this best robot and how rewards gradually move from* $-1.12 \times 10^5$ *to* $-1.10 \times 10^5$ *over episodes show that the learner made a positive change.*

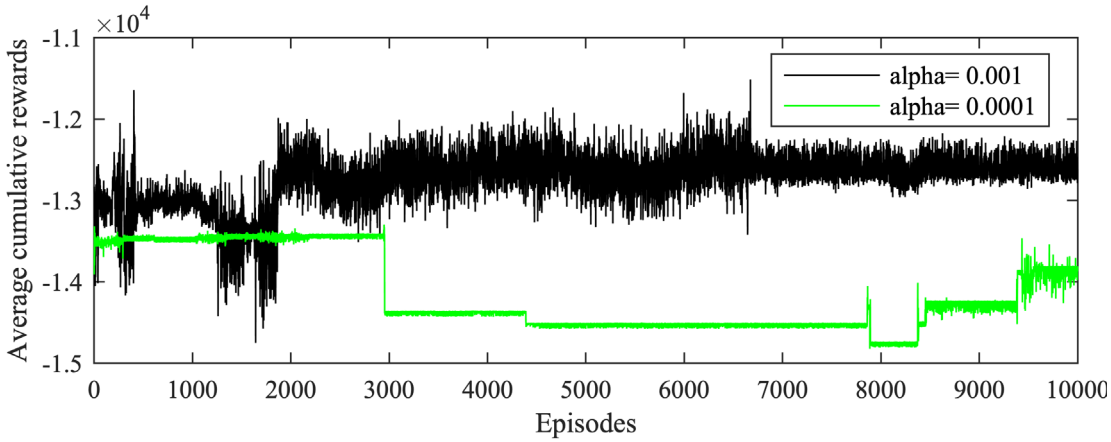



**Figure 38. Average cumulative rewards vs episodes.** Rewards used are (0, -1, -10) and robots are expected to reach point A with all other settings identical to those for Fig. 27. *Unlike previous experiments when* $\propto = 0.0001$ *performed better in the long-term, a decrease in rewards is received, shows the reward function and* $\propto$ *combination has a detrimental effect on learning.*

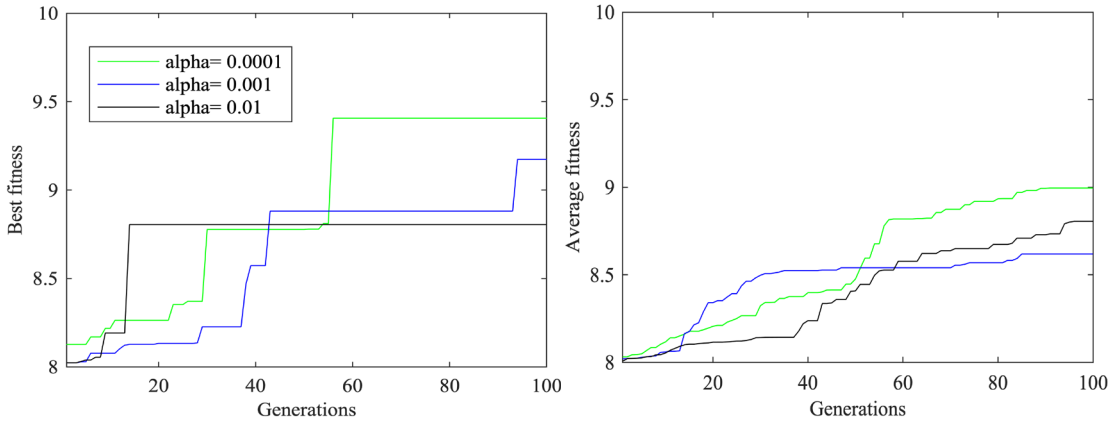



**Figure 39. Fitness vs generations for experiments in Fig. 38.** *Experiment with* $\propto = 0.0001$ *is the fastest as a population to reach the goal while* $\propto = 0.01$ *has the highest rate of fitness change.*

Even though $\propto = 0.0001$ displays consistent performance over most of the generations as shown by the tightly grouped green curves in Fig. 38, the rewards eventually did not converge. Hence, these results indicated problems with the rewards function. But the fitness improvement behaviour was again consistent with previous experiments (Fig. 39).

#### *5.2.3.4. Robots reaching point A, 100 s episodes and Euclidian rewards*

In the previous experiments, there were cases when robots made correct manoeuvres in the first 10 s of the fitness evaluation and changed direction away from the goal later on. This

suggested that longer episode lengths could help with the evolution. To test this, episode lengths were set to 100 s (also length of each fitness evaluation) in experiments from *Section 5.2.3.1.* The observations are plotted in Figs. 40 and 41. These graphs show interesting results as not only $\propto = 0.0001$ was no longer the best performer, but higher values of learning rate started to show variations in received rewards unlike in all previous experiments. Furthermore, the system stopped showing positive changes to the learning process.

#### *5.2.3.5. Robots reaching point A, 50 s episodes, Euclidian rewards and new EA parameters*

One of the problems in the previous experiments was that the evolver was only building small robots with a few parts at a time which did not change often over the entire evolution process. To counter this, the availability of the number of initial parts was increased from 10 to 15 and the probability of mutation related to body part addition was increased from 0.3 to 0.5. To save computational resources, both learning episode time and fitness evaluation time were set to 50 s and the number of episodes during learning was reduced to 3000 episodes because Fig. 27 indicated stagnant rewards after about 3000 episodes. As shown by the graphs in Figs. 42 and 43, learning rates of 0.01 and 0.1 exhibited a similar performance at the end of the evolution with $\propto = 0.01$, demonstrating the best learned performance from the set. Contrary to the majority of the previous experiments, $\propto = 0.0001$ was the worst performer.

#### *5.2.3.6. Robots reaching point B, 50 s episodes, Euclidian rewards and new EA parameters*

The settings from *Section 5.2.3.5* were maintained to evolve robots to reach point *B* at (-1.5, 1.5). Results shown in Figs. 44 and 45 display similarities to the results in *Section 5.2.3.5.* In both experiments, the learning rate at 0.001 and 0.01 demonstrated satisfactory performance in the learning process. Up to 50 generations, the experiment with learning rate at 0.1 evolved the best robots. The performance then stagnated for the rest of the simulation. The longer initial phase of around 10 generations for the best fitness to change in Fig. 45 suggested that robots were moving in the wrong direction as point *B* was diagonally opposite to the standard movement of the newly initialised robots.

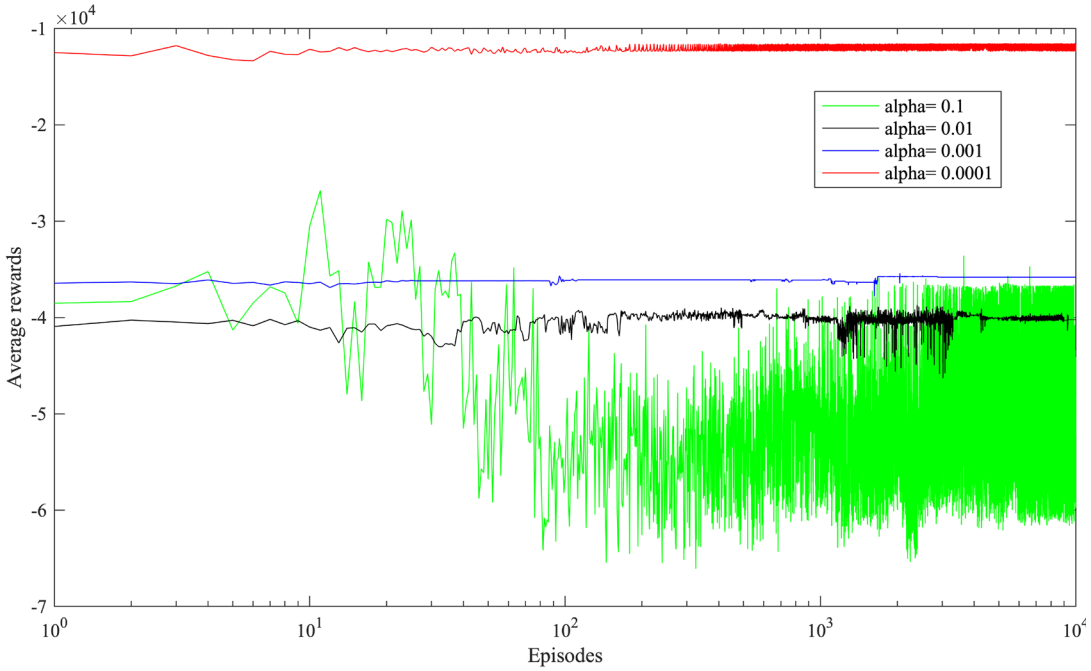



**Figure 40. Cumulative rewards vs episodes (logarithmic scale) while trying to reach point *A*.** Longer episodes length (100 s) all other parameters remained the same as in previous FNN based controller experiments. *The graphs display minute changes to rewards in the initial experiments and the eventual onset of undulations with an increasing amplitude depending on the value of* $\propto$. *Higher alpha values start to induce changes to the rewards unlike in previous cases.*

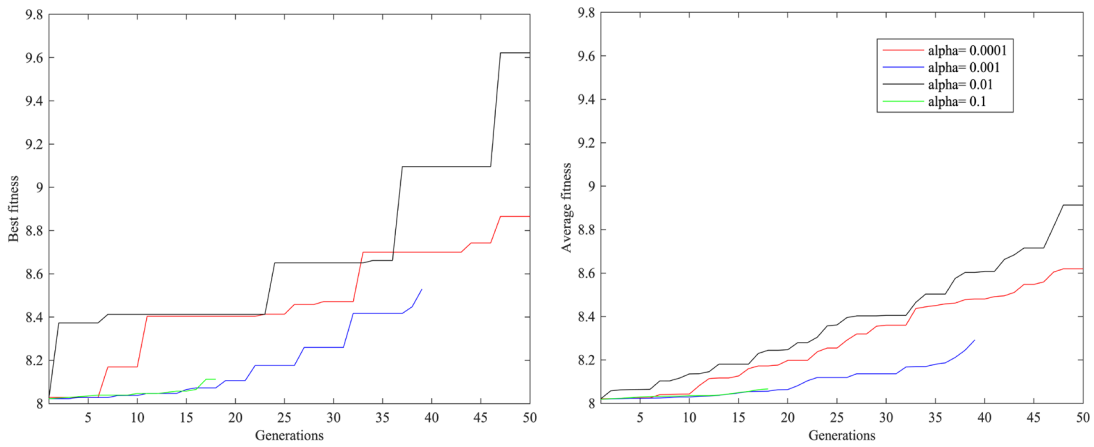



**Figure 41. Fitness changes over generations while using longer episodes (100 s) for learning.** Observations correspond to experiments in Fig. 40. *Unlike previous experiments when a lower* $\propto$ *results in a faster increase in fitness,* $\propto = 0.01$ *reaches the goal first. However, this rapid change in fitness results in a delayed response to the average fitness of the entire population.*

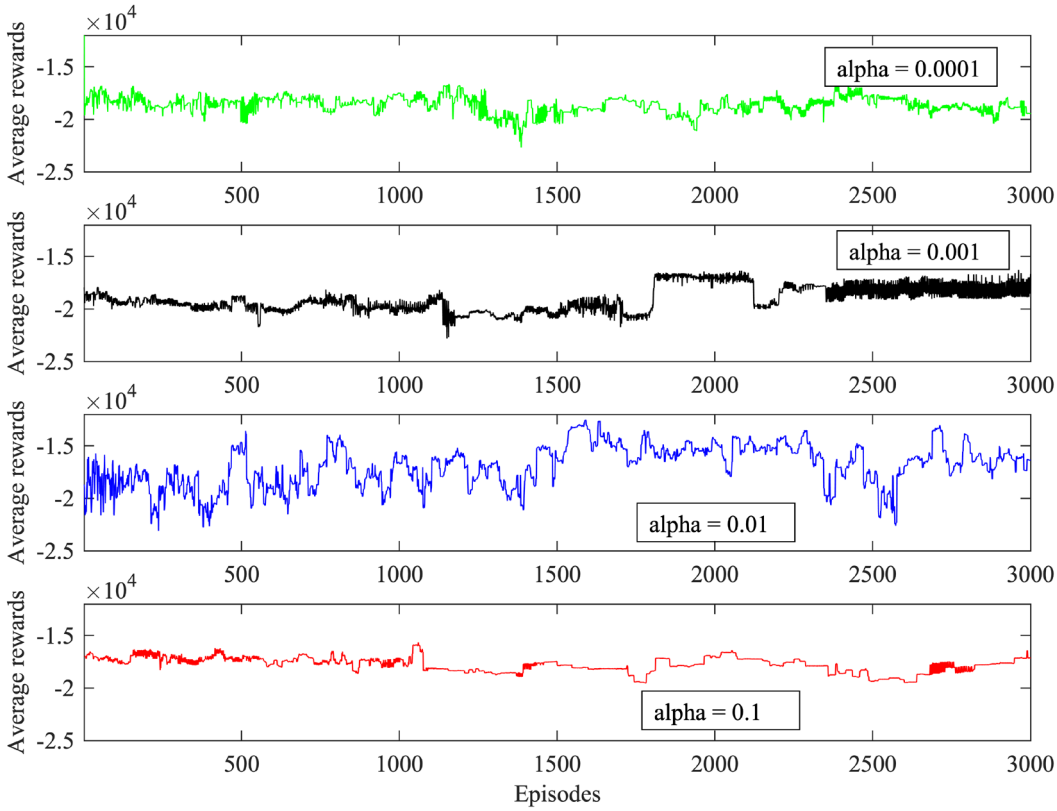



**Figure 42. Learner performance while attempting to reach point *A* with new parameters in EA and episode parameters.** The mutation rates for body part addition increased to 0.5 from 0.3, up to 15 initial parts available for evolution from 10, learning episode and fitness evaluation time set to 50 s from 10 s and 100 s, respectively. *The experiment with* $\propto = 0.001$ *shows slight improvement in rewards received when compared to the rest.*

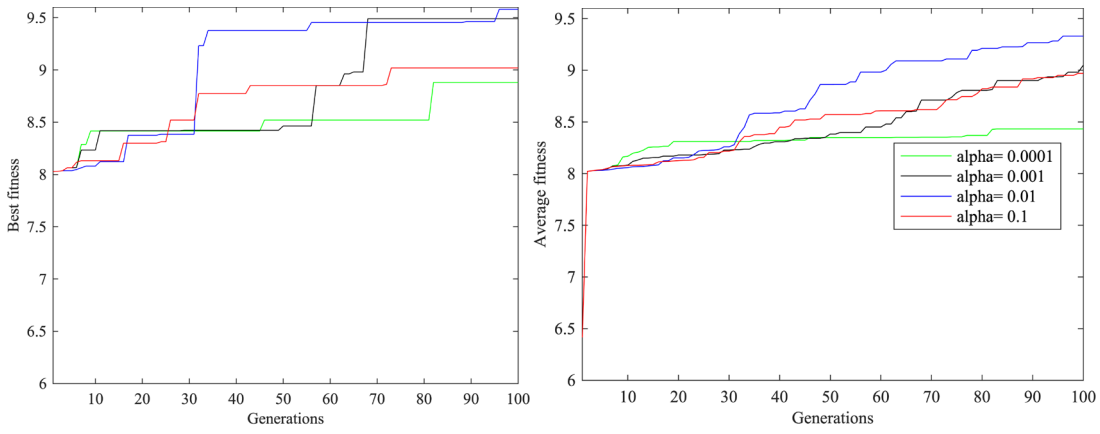



**Figure 43. Fitness change shown over generations to reach point *A*.** New parameters used for experiments in Fig. 42 applied. *Further to the change in trend observed in the last few experiments that higher* $\propto$ *values do not correspond to faster evolution,* $\propto = 0.0001$ *indicates the slowest population growth.*

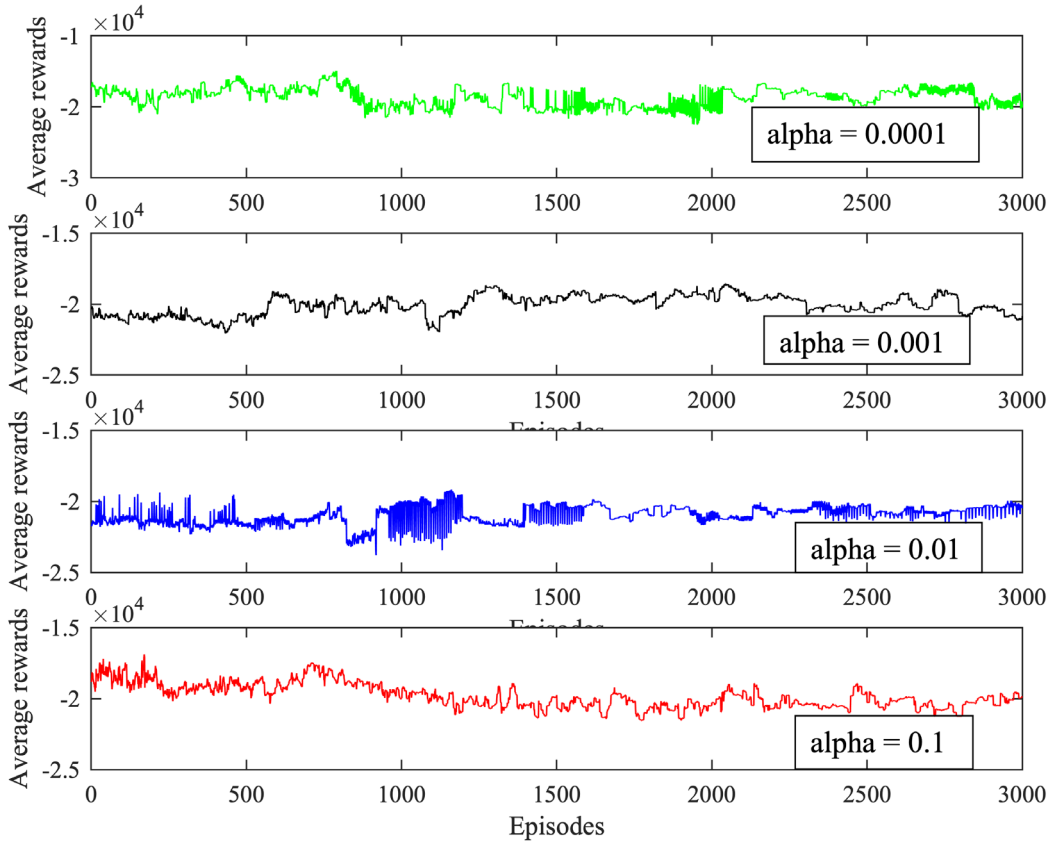



**Figure 44. Average rewards vs episodes.** Robots tried to reach point *B*. The evolution parameters and learning parameters remained the same as in experiments shown in Fig. 42. *Even though the curves shown above are average values over multiple generations, when* $\propto = 0.001$ *indicates that rewards are very close to each other.*

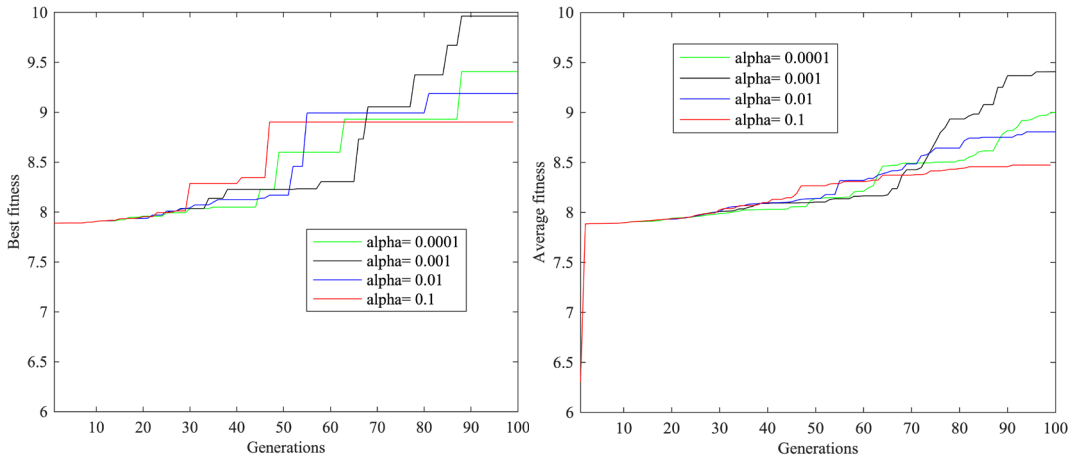



**Figure 45. Fitness changes of robots over generations for experiments in Fig. 44**. *The key observation is how, irrespective of the* $\propto$ *value, the evolver exhibits identical performance until about 20 generations. At the end of the evolution, the experiment with* $\propto = 0.001$ *evolves at least one robot to reach the goal.*

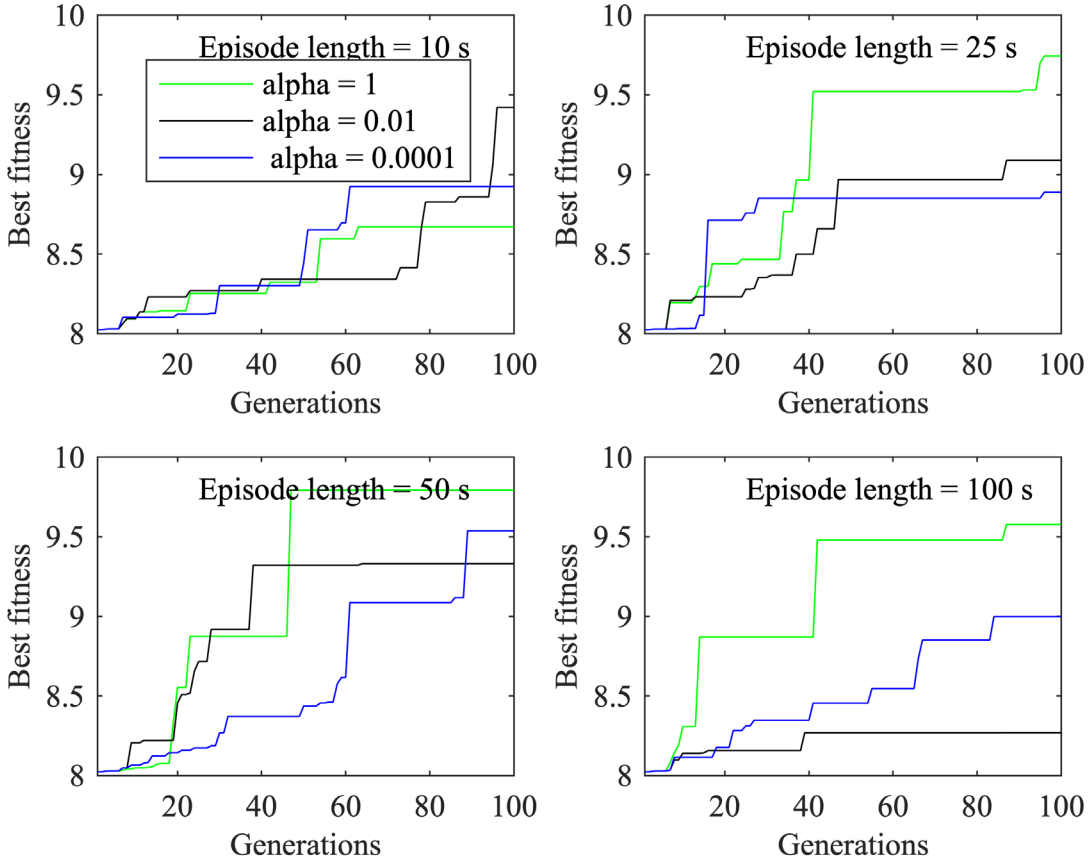



**Figure 46. Effect of episode lengths on best robot fitness.** All parameters remained the same as for Fig. 27. The number of episodes was set to 3000, robots were evolved to reach *A*. *An increased learning rate shows better performance in higher episode lengths. Episode length of 50 s shows the best performance for* $\propto = 1$. *However, if episode lengths are shorter, lower learning rate demonstrates faster performance of the evolver.*

### *5.2.3.7. Relationship between learning rate and episode length*

Figs. 29 and 41 demonstrate how an increase in episode length from 10 s to 100 s made the evolver that used $\propto = 0.0001$ drastically reduce its speed of evolution. In an effort to obtain the relationship between different learning rates and episode lengths, multiple experiments were conducted. For the experiment in *Section 5.2.3.1,* episode lengths were set at 10 s, 25 s, 50 s and 100 s with learning rates of 1, 0.01 and 0.0001. The progress of evolution in each of the combinations is shown in Fig. 46 and Fig. 47. Results indicated that for faster performance of the evolver, depending on episode lengths, different learning rates needed to be chosen. For optimal performance with a 10 s episode length, the learning rate should be 0.0001 and for any episode lengths above 25 s, a learning rate of 1 was preferred.

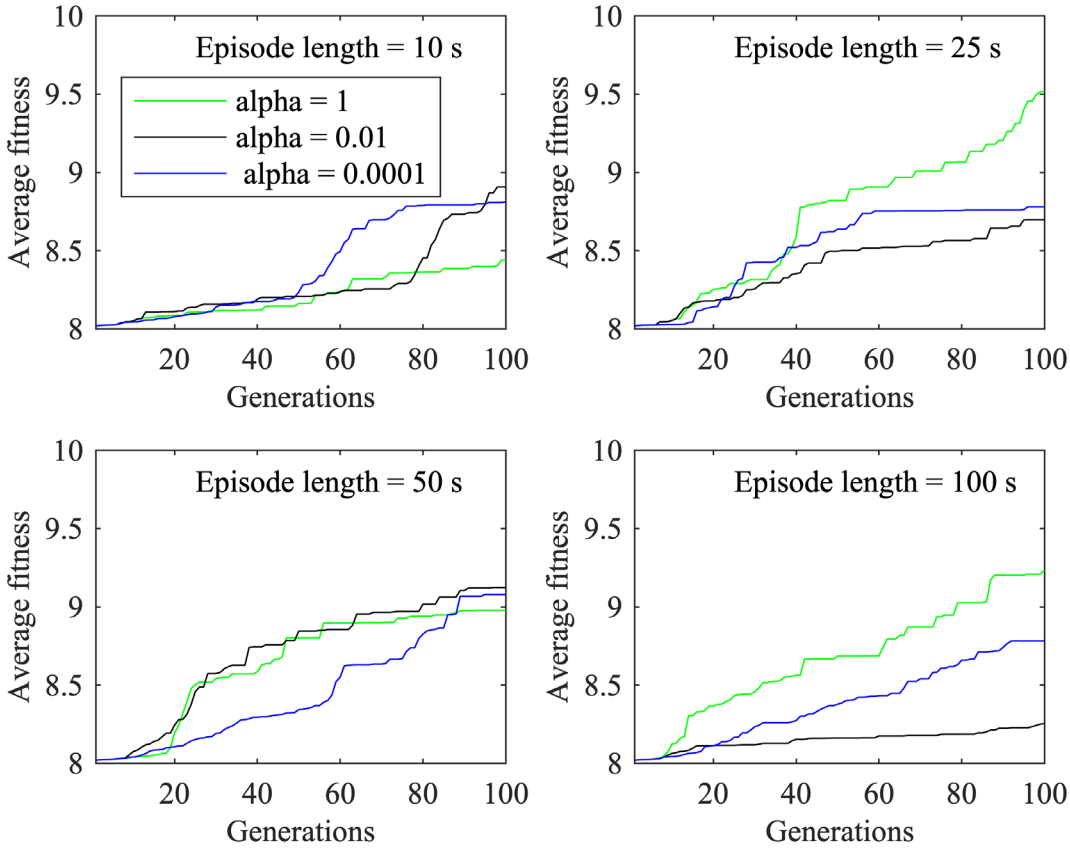



**Figure 47. Effect of episode lengths on average fitness of population.** Results correspond to experiments in Fig. 46. *An episode length of 25 s shows that average fitness of population behaves best with* $\propto = 1$.

### 5.2.4. Experiments with a Standard EA

To compare the performance of ReCoAl with the standard EA, robots were evolved to reach *A* at (1.4, -1.4) and B at (-1.5, 1.5). Fig. 48 shows how the fitness of the robots improved over time and Fig. 49 plots the trajectories of multiple robots while evolving to reach *A*. Robots evolved with ReCoAl had less than five parts (Fig. 32), while robots evolved with the standard EA were considerably longer (9 parts in the first generation) due to the random initialisation of the evolver, not due to the learner. This unfair advantage helped them reach the goal faster than robots evolved with ReCoAl (Fig. 50).

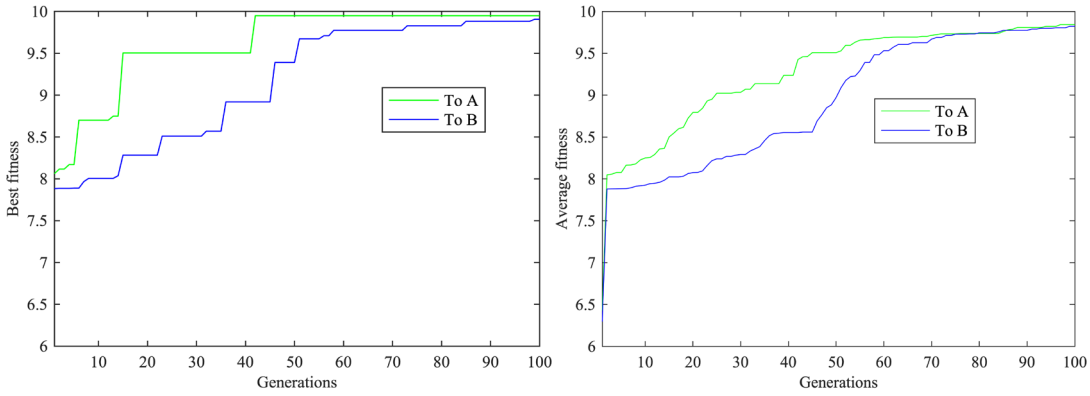



**Figure 48. Evolution of robots that reached points *A* and *B***. A standard EA applied using same parameters as for Fig. 27. *Both experiments demonstrate that robots are evolved to reach the goal. Robots reach A first even though there are obstacles in the route to A and not B.*

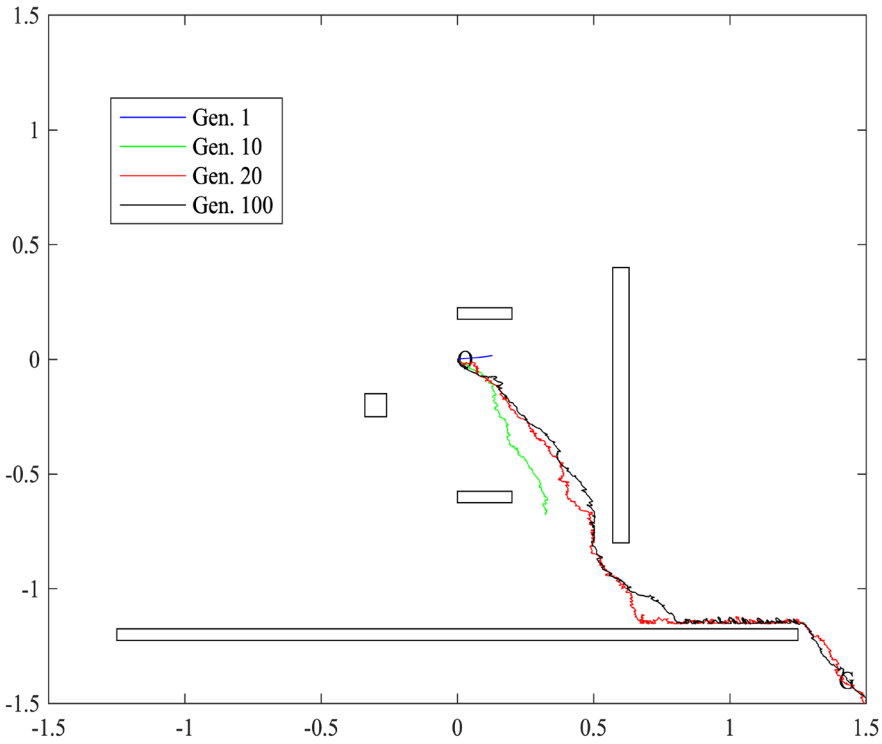
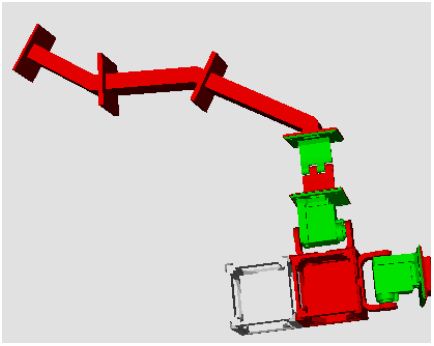
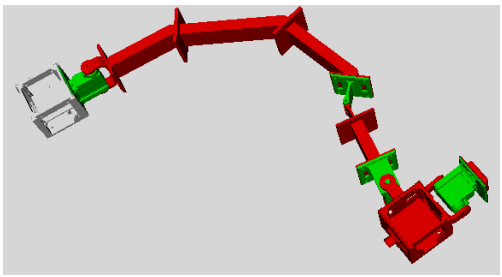


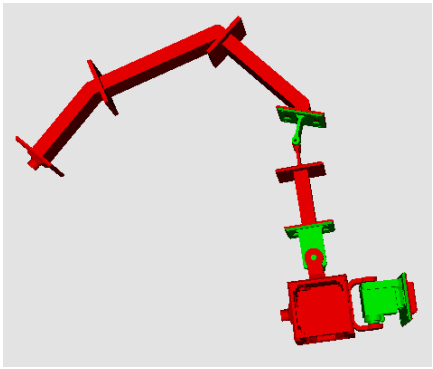

**Figure 49**. **Behaviour of best robot in different generations while evolving with a simple EA**. Robots shown were evolved to reach *A* by the same experiment as for Fig. 48. *The trajectory for the best robot in the first generation shows slight movement from the centre while later, multiple robots reach the goal. The same robots are shown in Fig. 50. The performance changes can be correlated with the best fitness values in Fig. 48.*



Gen. 1



Gen. 10



Gen. 40

**Figure 50. Physical representation of the best evolved robots in each generation while using a normal EA**. *Images show how the standard EA is able to start with a bigger robot (more than 40 cm in length) than the corresponding ReCoAl robot (always less than 15 cm) in Fig. 32. However, this is a result of randomness of the evolver and therefore cannot be used for comparison. Due to this unfair advantage, robots evolved with pure EA are able to move faster to the goal.*

## 5.3. Discussion

The ReCo algorithm was designed specifically to improve the co-evolution process of mobile robots and there was strong evidence to suggest that the evolution process affected the learning process with both positive and negative effects.

Positively, the trajectories of the evolved robots using the ReCoAl exhibited sufficient changes in the correct direction due to feedback from the reward function. This was evident from robot trajectories in Fig. 30. For instance, the best robot in generation 50 progressed from an immobile state before learning. It first moved towards the right of the arena then made an almost 45° turn to face the goal and moved along a straight line to the goal. The

robot thereby got maximum cumulative rewards, as a distance-based reward function was used. The effect of the learner to force the proper use of actuators was also clear as the triangular waveform in Fig. 31 exactly replicated the target signal set in *Equation 5.22* in every time-step from the beginning, from a control signal that remained constant after about 5 time-steps. The combination therefore demonstrated that both the error signal calculated, and reward signal generated, were sufficient to first create an oscillatory waveform to move. The learner was then able to make the necessary changes to the NN weights to trigger the direction change for moving towards the goal.

Among the negative effects of the learner, the improper selection of evolution parameters could drastically reduce the speed of evolution. There were several examples demonstrating this; when the learning rate was set at 1, the robot fitness hardly changed (Fig. 29). There is documented evidence in the literature on how high learning rates negatively affect the convergence of learning algorithms [149]. However, this unnoticed slowing effect on the speed of successful evolution was contrary to the general expectation that the variation operations during evolution would make random changes that could lead to robots with better fitness as shown in EA only evolution (Fig. 48).

This negative effect was also emphasised by the results in Figs. 33, 34 and 35. The only favourable change generating the best robot in the entire evolution run occurred in the $4^{th}$ generation and that robot was preserved as the best robot since no children generated in the subsequent generations were able to generate a better fitness. This demonstrated how a wrong parameter selected during learning could negatively affect evolution. Furthermore, the ancestor robot was only preserved due to the $(\mu + \lambda)$ population strategy that provided parents an equal chance to progress to the next generation. In contrast, only allowing children to be present in the new population could again reduce the fitness of the most fit robot. Furthermore, the ineffectiveness of the learner was further demonstrated by how the learner periodically generated control signals that were outside the necessary bounds (Fig. 35).

An important factor that affected the outcome of learning process was the evolved morphologies and corresponding NN parameters. There were instances when the robots did not have enough morphological capabilities to reach the goal. Even with the maximum possible speed, the robot would not have been able to reach the goal due to the size and

number of actuators. There were also cases when robots were evolved with motors facing upwards or in a plane that could not generate motion that interacted with the ground (e.g. Fig. 34, Gen. 1), thereby making the robot unable to move.

From the perspective of an evolved controller, the learner only manipulated the weights of the NN. Gain and bias of each neuron also played a role in obtaining a favourable outcome which was modified by the evolver and not the learner. Even when the learner tried to counter their effects, the large values of gain and bias made compensation extremely difficult or even impossible within the given timeframe and learning parameters.

Experiments with positive results demonstrated how ReCoAl favoured co-evolution. The long-term learning behaviour demonstrated by some of the robots (such as the ones from generations 9 to 100 that use 0.001 as learning rates, Fig. 28) are unclear. The gradual increase of the undulations (green curves in Fig. 28) suggests an unstable behaviour. Though the exact cause of this is unknown, it is possible that the combination of error function (which has an oscillatory nature) coupled with an ever-increasing eligibility trace ($z_t$) could be the cause.

The random nature of evolver began the EA only based evolution with much longer robots. This created a greater advantage over robots with fewer parts using ReCoAl making the performance comparison unfair. For the same reason, the time taken by evolvers are not considered in this discussion as the standard EA evolved much faster than ReCoAl. To counter this, both evolvers needed to evolve with the same seed robot and same evolutionary parameters. However, the ability to set a seed robot option was unavailable in the software.

When each learning episode was set at 10 s, there were robots in the population that performed well in the first 10 seconds by moving towards the goal. However, during the actual fitness evaluation they moved *away* from the goal which resulted in a poor fitness value. To avoid this problem, longer episode lengths were applied. Nevertheless, this did not yield satisfactory results due to improper learning parameter selection. For shorter episode lengths, lower learning rates were sufficient to generate effective robots. The experiments suggested that for optimal evolution rates, episode lengths and learning rates are related. For optimal performance when learning for 3000 episodes, 10 s episode length needed a learning rate of 0.01. For any episode lengths above 25 s, a learning rate of 1

performed well. Normally, 25 s episode and learning rate of 1 showed the best overall performance for the entire population.

The effect of rewards on the learning process was also visible throughout the experiments. In cases when reward functions made a positive impact on the results, their effect was more evident in the initial stages of each of the episodes, as indicated by the initial direction changes displayed by successful robots. However, major directional changes did not occur later, suggesting that rewards were only favoured initially. Though the (0, -1, -10) rewards have shown success in several RL applications in robotics, in these experiments the rewards did not make noticeable changes to the weights. As a result, the experiments designed for sensor use did not produce meaningful results. Furthermore, problems were created as the reward was allocated to direction and not for movement itself. It meant a robot could reap rewards by just facing the right direction and not moving. This form of result is referred to as *reward hacking*. In other expected problems with the reward function, directional or Euclidian distance-based rewards would only have worked in obstacle-free arenas or arenas with simpler obstacles. This also was an important factor to consider while analysing the obtained results. But as a next step, improved reward functions can always be built.

The learnt knowledge in a controller was passed down the generations. This was apparent when in several experiments, newly created robots began performing move and turn manoeuvres before the learning process began. This highlighted the appropriate choice of selecting a DS based RL algorithm. However, the problems of reward design and parameter selection still remain as challenges. While an error generator that induced an oscillatory movement did create robots that follow precise instructions, it was certainly not enough when more than one actuator was present in the robot. Depending on the position of the motors, the oscillatory motion of robots created cancelling effects or even made direction changes difficult. Nevertheless, these problems were anticipated as the robot morphology was not known beforehand and a proper error function design was not straightforward.

## 5.4. Conclusion

This chapter was built on the key observation of how improperly evolved controllers in co-evolved robots were affecting the course of evolution. For countering this problem, it was hypothesised that learning can aid mobile robot co-evolution process. To demonstrate this,

ReCoAl was designed and tested in the *RoboGen* environment to co-evolve robots that perform navigation. The algorithm was designed to improve the robot co-evolution process, with an emphasis on the controller generated during the process. From performed experiments, the algorithm when applied to a recurrent neuron-based controller did not show any advantages due to the architectural limitations associated with the NN type. However, when applying on an FNN based controller, the system was able to evolve robots, even with very few initial parts itself, thereby fulfilling the first objective of evolving better controllers that can better utilise the morphology for locomotion. Furthermore, recommendations have also been made for selecting the various learning parameters available during the evolution and learning process (*the third objective*). This chapter also tested a new method to evolve robots that could properly use its available sensors. However, this was unsuccessful which is again consistent with the literature and results from previous chapters (*the second research question*).

In summary, the results not only indicate that learning can aid the evolution of robots, but it can also have detrimental effects, depending on the effectiveness of the learning algorithm. This finding is consistent with Polmin's observation in humans, that says our developed abilities are equally as dependent on the environmental factors during one's lifetime, as on genetic factors. Similarly, the environmental factors and learning algorithm along with the inherited skills are crucial in generating a robot capable of reaching the goal. Ultimately, the results showed positive outcomes, but full use of the algorithm was not possible due to the number of limitations outlined in the previous section. But recommendations have been made to help researchers properly select available learning parameters. Nevertheless, ReCoAl has shown an ability to create a robot from a small number of parts that can reach a goal in a set time, which demonstrates its potential for improvement.

# Chapter 6.

# Discussion and Conclusion

## 6.1. Discussion

This section is divided into two parts; the first discusses implications of the results reported in the previous chapters with reference to the overarching aim of this thesis, and the second links these results with various concepts from evolutionary biology.

In an attempt to improve co-evolution process, *Chapter 3* focussed on steps undertaken to have a better understanding of the process itself. In addition to expected results or evolved robot behaviours as reported in the literature, the chapter identified some of the unreported or under-reported problems currently faced while co-evolving mobile robots, for instance sensor availability and its use, which is a fundamental requirement for proper functioning of mobile robots. This is not given enough weight in relevant literature. On the other hand, results in experiments from *Chapter 3* not only indicated improper use of sensors, but also demonstrated that most times during evolution, sensors were not even available on evolved robots. However, an argument supporting this observation, is that sensors are not always necessary for robots to solve the required tasks and so there is no evolutionary pressure to use sensors. Even then robots were not generated that could properly perform the tasks, as, the evolver was not given sufficient or relevant feedback on how each of the robots were performing.

To tackle this, fitness function design was the focus of the following chapter. This is also under researched in co-evolutionary robotics, and was the first objective in *Chapter 1*. The intention to evolve robots that performed better than when compared with normal co-evolution was attempted through the use of an A-star algorithm-based fitness function designed for navigation and obstacle avoidance. A second aspect explored was the effect of cumulative robot performance as measured through step-by-step behaviour. Inconclusive results indicated a number of problems in applied evolutionary principles. One of these was

information reduction resulting from combining multiple factors into a single number, which resulted in loss of otherwise usable information during parent selection.

A similar underutilisation of data resulted from treating robots as a single entity during evolution instead of performing targeted changes during variation as in evolutionary biology. During the crossover operation, a random *cut-paste* operation was performed on the entire body and/or controller instead of incremental changes to underperforming aspects of each robot. This change most of the time impeded any progress achieved till then. On the other hand, only targeted changes happen during crossover operations in biology. That is, only corresponding genes responsible for each feature are combined during crossover operation in a human offspring genome.

Out of possible solutions to the above problem, a controller-only improvement method was introduced through a learning phase applied to every evolved robot which worked by making incremental changes to co-evolution instead of making significant changes to the way an evolver operated. A new algorithm called ReCoAl incorporated a RL algorithm at its core to incrementally modify a robot controller based on a real-time feedback from the corresponding robot's behaviour in a virtual simulator. ReCoAl also attempted to solve previously mentioned problems identified in *Chapter 4* mainly via the addition of an extra layer of feedback used for immediate robot improvement. This feedback applied real-time incremental targeted changes to the controller instead of a single random and large change that otherwise happened during evolution. Furthermore, the new process attempted to suitably use available sensors and actuators to link them with the controller to ultimately achieve navigation tasks. This was through specifically designed reward functions and error functions encouraging proper sensor and actuator use respectively.

Results generated with ReCoAl demonstrated the proof-of-concept that, such learning could improve evolution (second objective in *Chapter 1*). While both positive and negative effects of learning were observed from a broader perspective, in-depth analysis revealed how only error functions made a positive difference while reward functions most of the time did not.

Reward functions designed specifically for sensor use, did not seem to make any observable changes to evolution or behaviour of robots, but function designed just for navigation showed satisfactory results. Although the answer to the question of why sensors do not work

in co-evolved robots still remains elusive, a possible reason for such behaviour while using ReCoAl can be explained as follows.

In a simple FNN controller with a certain number of inputs and outputs, an output signal is generated based on input signals and FNN parameters. The output signals' effect in a virtual environment is measured in every time-step through a reward. This reward is the effect of a certain input, output and FNN parameters combination. In the RL algorithm used, this reward was fed back into the controller to modify FNN weights through backpropagation. Assuming that the input signals remain the same in the subsequent time-steps, it can be mathematically proven that rewards gained will eventually reach the best possible value in a finite time if such a solution exists [144]. This means rewards should be able to force the use of sensors at the end of learning. However, in reality, after the second time-step both input and FNN parameters change as a result of the previous time-step's output and learning processes, respectively. During these changes, the input signal (also sensor output) variation as a result of the robot's interaction with the environment is not factored while proving the convergence of the RL algorithm used. Instead, only the FNN parameter change is accounted while proving its convergence. This factor can indeed cause the algorithm to not function properly.

Furthermore, among other findings when using ReCoAl, as with EAs, learning combined evolution also depended largely on a number of learning parameters fixed during the process. This discovery prompted the execution of a number of experiments to identify relationships between learning rate and episode lengths. It was found that normally, evolution performed better when higher learning rates were coupled with higher episode lengths, as higher learning rates allowed a greater memory of previous robot actions. Naturally there was more that can be remembered in longer episodes. The corollary of low learning rates limiting the use of the available step memory in longer episodes was also observed.

To summarise: efforts were undertaken to identify current problems with co-evolution from the literature and work reported in *Chapter 3*. After attempting to solve them one by one through *Chapters 4* and *5*, a number of questions still remain as challenges when co-evolving mobile robots. Therefore the broader implications of the results are now discussed from an evolutionary biological perspective.

In addition to the anticipated effects of EAs, a closer look at its underlying theory with generated results shows unexpected parallels with findings from evolutionary biology. EAs are mainly inspired by Darwinism which explains how fitter individuals have a better chance to pass on inherited characteristics to produce fitter offspring. But while much of this model remains intact, there are other factors that need to be acknowledged to explain the observed improvements in co-evolution.

Though ReCoAl is inspired by Polmin's observation of human behaviour, its roots can be found in modern evolutionary biology. The Baldwin effect explains how an organism's ability to learn new skills within its lifetime affects its reproductive success [194]. The results of ReCoAl demonstrated this: the robots that learnt and modified their NN were consistently selected for variation. Furthermore, in these two theories and in the Evolutionary Algorithm variation operations always occur in the genotype space, with no information flow from phenotype to the corresponding genotype. On the other hand, the ReCoAl's performance hinged on its ability to pass on a learnt controller to subsequent generations, which can be explained by *transgenerational epigenetic inheritance*. In biology, this describes how a parent's newly gained traits can be passed on to a child without making structural changes in the DNA of the child [195]. Similarly in ReCoAl the genotype structure of the robot, that is the body tree and NN structure, was not varied and instead just the learnt NN weights, corresponding to new skills learnt in biology were inherited by the child.

There are also notable differences alongside similarities of concepts in both ER and evolutionary biology. In humans, the body or the phenotype information is transcribed in genes, with each or a set of genes responsible for a particular feature. While specific genes control a particular feature, the actual feature is defined by *alleles* that hold variable information (e.g. even though there are unique genes responsible for eye colour, the actual colour is defined by alleles [196]). However, in the gene structure used for this thesis, only the genotype representation was fixed (as a tree form) unlike in biology where the entire gene structure is fixed. Every part of the robot was added as a branch of the tree which carried the location of the connection, the part type, its structural details and the neurons associated with the part along with its parameters. Therefore a gene corresponded to a part, and the allele represented variable parameters in that part. This is equivalent to a one-to-one

mapping from gene to feature in these robots while often many-to-one mappings occur in humans. The primary reason for this was to make features easily trackable, unlike the human genotype which is being studied to find out more of the genotype-phenotype mappings.

While this simplicity can be acceptable in ER, it is worth noting that when a biological species reproduces, the offspring also belongs to the same species (e.g. human parents have human children). However, when a robot genotype mates with another, the result is infrequently a robot that is visually different through its structural features, even though this change does not happen often (as demonstrated in *Chapter 3*). This shows the addition of new genes into the robot. Even the addition of a neuron in the controller can be categorised as a major alteration. Such transformations, when happening in a biological organism, lead to an entirely new species (referred to as speciation). The occurrence of speciation in living organisms is equally rare and inter-species reproduction is extremely unlikely to happen in evolutionary biology. There are several theories trying to explain how this can happen and the debate is ongoing [197].

Therefore, instead of step-by-step changes during evolution, ER conducts the process more quickly to induce rapid changes to the offspring. It is one of the possible reasons why, co-evolution is not that successful. Thus, the question still to be asked is: For effective evolution should one gradually modify genes, or add or delete them frequently? Or maybe ER is designed to accelerate the whole process. This means ER is not devised to evolve among a single species but through an inter-species mating mechanism that is currently unexplained by observed evolutionary biology principles.

Microevolution describes subtle changes happening in a biological population through mutation, selection, gene flow and genetic drift in a short period of time (not more than a few generations). Here, gene flow is the mixing of alleles from multiple populations due to immigration, while genetic drift is the loss of certain alleles in a population due to random sampling while reproducing. Microevolution when observed over a longer timescale (millions of years in biology) is referred to as macroevolution [198]. In ER, mutation currently only takes place during reproduction and not during a robot's lifetime. Gene flow does not occur at all as just one population participates in this evolution. In biology however, mutation and gene flow can happen multiple times in a generation.

Biology shows that multiple factors affect parent selection during reproduction. It is not just based on optimising one fitness measure per individual, but involves a combination of multiple factors, with a lower preference given to individuals that are outside a set of criteria. Some criteria are related directly to environment or opportunity and others less directly related to visible fitness. The resulting uniqueness is one of the causes of species extinction due to a lower probability for such organisms being selected for variation operations. In contrast, only a single fitness measure drives selection while robots evolve, and their uniqueness often helps to introduce new favourable traits to future offspring.

The lack of ER's success could also be caused by the improper population size selection during evolution. Selecting an effective population size for the conservation of species is a well-studied problem in biology. Despite this, there is still no consensus on what it should be, other than agreeing that with multiple factors affecting their lifecycle the number will be different for each species. Harmon *et al.* even stated that there is no such thing as an effective population size [199]. Explained in [199], the method designed for its calculation considers the two key factors of inbreeding and genetic drift, which is based on the premise that inbreeding is known to cause harmful gene combinations and genetic drift results in the reduction of genetic diversity.

The inbreeding coefficient ($\Delta F$) in [199] can be calculated as:

$$\Delta F = 1 - \left[1 - \frac{1}{2N}\right]^{t} \tag{6.1}$$

where $N$ is the population size, $t$ is the number of generations and a $\Delta F$ resulting higher than 0.6 is considered high. The results from ER also show inbreeding causes a reduction in genetic diversity. In the experiments from *Chapter 4,* at about 400 generations the coefficient reaches very close to 1 when the population size is set to 20 and the same happens again at 100 generations when the population size is 10 (see *Chapter 5*). It means that most of the robots are inbred during long evolution epochs. This is consistent in the observations where the entire population is filled with identical individuals after some time. To overcome this, though some methods have been introduced [6], there is not much research available in population design for ER. An example of such a method is behavioral diversity-preservation: this periodically searches for uniqueness in the population and applies a lock on selected robots to save them from deletion later to improve the speed of evolution [200].

The concepts in this thesis were tested with the help of a pre-existing software package, *RoboGen.* Consequently, its limitations also affected the course of this investigation. Among some of its functional problems that were encountered: the software occasionally crashed for generations over 37,500; the simulator often returned errors during evolution; and the software did not allow the removal of neurons during evolution which can happen to genes in biology. The underlying design limited the fitness to a single number and a fixed function stopped the implementation of new ideas related to *Chapter 4*. There was limited flexibility during population design and EA options selection. Furthermore, the originators of the software provided no guidelines on the material properties of the parts to help with physically building robots.

## 6.2. Conclusion

This thesis aimed to build on the current knowledge-base used to automatically generate mechatronic systems. The literature review suggested Evolutionary Robotics (ER) as a viable area to focus on for achieving it. In ER, the subtopic that deals with simultaneous evolution of mobile robots or virtual creatures with evolved body morphology and controller has been studied since the 1990s. Less than 1% of all works in ER report buildable co-evolving robots, which was the area of focus of this work.

Based on the direction given by the literature review and experimental results from *Chapter 3*, two main approaches to improve this evolution process were tested. The effect of a sophisticated fitness function for evaluation as opposed to using simpler fitness functions applied in the literature was examined with an A-star algorithm based incremental fitness function designed for evolving robots capable of navigation and obstacle avoidance. Though the results showed positive outcomes, its effectiveness was not fully justified. A major difficulty identified is about the poor controller that was unable to exploit the available sensors and actuators. To overcome this problem, a hybrid evolutionary and reinforcement algorithm (ReCoAl) was proposed. It worked by introducing a learning phase for every evolved robot before fitness evaluation. Results showed that the learning algorithm was able to evolve robots that could use its actuators and move at its maximum speed even with just a few parts that make up the robot. ReCoAl was built on a module that runs alongside *RoboGen* which makes it easier for testing new learning algorithms for co-evolution.

On a final note, evolutionary approaches have found successful application in robot morphology design and simultaneous evolution of controller and structure among other application areas. But the question to be asked is: Can such an evolutionary process outperform the current system of individual manual robot programming? Even though the answer to it may not be positive, at least for the present, it can be hoped that the full benefit of artificial intelligence based EAs for the co-evolution process is something to look forward to in the future.

With the benefit of hindsight, after attempting to incrementally solve a number of problems in co-evolution through a variety of approaches, it can be seen that co-evolution remains as an open field of study with a number of unsolved questions. It can be stated that the technology has still not reached its peak and will continue to evolve towards fully *automatic synthesis* of robots in the future or towards the *evolution of things* [201]. Even though ER is inspired by biological evolution which has evolved trillions of organisms over billions of years, artificial evolution is still far away from being conducted at that scale. Nevertheless, this should gradually change as on one side the understanding of the biological evolution deepens while on the other side the technology matures.

## 6.3. Future Work

Throughout this thesis, individual recommendations have been made in appropriate sections that reflected on the literature review and observations from the experiments conducted. After considering the factors discussed in *Section 6.1,* several areas can be identified that one may study to take the co-evolution process forward.

The experiments from *Chapter 4* indicated that pure distance or goal-only based fitness functions were not sufficient to evolve fitter robots. Therefore, the first step can be to gradually add factors (discussed in *Section 4.4*) during fitness function development. This could be through applying morphological and controller suitability, and behavioural assessment factors in fitness function. Simultaneously, the fitness function structure can also be changed to vary over a period of evolution allowing a gradual addition of skills over

evolution. For comparison, factors neglected due to high computation costs should also be incorporated.

Even though attempts were made to develop the evolver to use an obstacle sensor when available, that functionality is still virtually non-existent. This stresses one of the main challenges still faced by the research community. However, in this particular case, the problem could also be due to poor reward design which is another major challenge in Reinforcement Learning. Bearing in mind how the Euclidian reward worked well in the initial phase of each episode, a non-linear reward system can be designed that increases the reward over each episode. Additionally, the ReCoAl can be sped up by adding checks to ignore morphologies unsuitable for learning and performing selective learning by analysing ancestor data of the offspring. Error function design for the learner is another area that can be improved by examining morphology.

Considering that ER roots can be found in biology, there is still much that can be done to take it closer to biological systems. Some of the methods that could be added on to the list in ER are:

- Creating better representations for genotype and phenotype as this will help in the performance evaluation and mating process.
- The progress analysis and more selective variation operations can be more easily performed by implementing ancestry tracking methods in the genotype. For example: identifying the family tree of a robot can help in tracking how each feature is introduced or lost.
- Adding new rules during mating to ensure individual functionalities from the parent are preserved in children. For instance, during variation operations, the basic genome structure of the robot can remain the same and only alleles can be changed instead of the entire gene which is randomly modified during variation operations in ER.
- To enable phenotype plasticity specific to co-evolution that allows changes in the phenotype due to changes in environment. There are already reported works in neural network plasticity.
- To perform a much more comprehensive fitness evaluation by adding multiple factors into it. Instead of just one number deciding fitness, each feature should have

separate fitness, and during fitness comparison these individual fitnesses should be compared to help complement features in parents.

- ER progresses by occasionally making drastic changes on its offspring, instead of a step-by-step change as seen in natural evolution. Considering the multidimensional nature of robots and lack of proper progress, it could be time to take a step back and rethink whether natural evolution is right or not.
- Methods like microevolution and macroevolution and concepts like gene flow and gene repair theories also seem suitable for ER.

Nevertheless, it must also be remembered that closeness to natural paradigms does not necessarily mean improved performance. There are still many open questions in evolutionary biology, and natural selection has shown that it is not always the best problem solver, e.g. the human eye has its photoreceptors facing the wrong way.

Furthermore, specific problems that could be immediately addressed are:

- Methods in designing effective populations to avoid inbreeding and preserving genetic diversity.
- Finding a solution to improper sensor use in co-evolved robots to build robots that can properly use sensor feedback in the control process.
- Enabling continued learning in evolved robots and being able to test this function during fitness evaluation. However, this might be too ambitious in the near future as the field is still struggling to co-evolve robots that can perform primitive tasks.
- Building a stable software environment for co-evolution as very few such packages exist and there are many constraints in them that stop ER reaching its full potential.
- Understanding how to properly tune evolution parameters is extremely important as there are no systematic approaches available in ER.
- Clarification on the trade-off between just evolving morphology and co-evolving it with a controller in the light of recent evidence that states co-evolution is better than controller only evolution [202].
- Methods to fix the *reality gap* between virtual simulation and real systems, as insufficient works have been reported here and it is one of the major hurdles in evolving physically buildable robots.

- Research is also needed in developing benchmark problems in ER to help with the comparison process.

# GLOSSARY

**Allele:** Form of a gene that decides phenotype traits.

**Artificial Neural Network:** A control method inspired by the neural connections in a human brain.

**Artificial Intelligence:** The field that studies intelligent agents that are designed to work in circumstances not known beforehand.

**Braitenberg vehicle:** A mobile robot that responds to sensory feedback by changing actuator signals without using a separate controller.

**Bootstrapping:** In ER, the tendency of an evolver to get stuck when all individuals in the population have identical fitnesses.

**Butterfly effect:** Observation of how even minute changes to initial conditions of an algorithm can cause large variations to the output at a later state.

**Controller:** The software of the robot that performs control operations.

**Co-evolution:** The process of simultaneous evolution of robot body and controller.

**Crossover:** Random (copy-paste) operations performed on multiple parents using genome segments from each other to generate offspring.

**Denavit-Hartenberg parameters:** Parameters used for robot joint coordinate axes allocation while using DH rules.

**Evolution:** The process conducted using an EA to generate robots for an application.

**Evolutionary Algorithm:** Algorithms inspired by the biological evolution process.

**Evolutionary Computation:** Branch of engineering that primarily uses EA for solving problems.

**Evolutionary Robotics:** Branch of robotics that use EC for robot development.

**Fitness function:** The method which computes the extend of suitability of a robot to perform a task.

**Generation:** A single cycle of an evolution process which results in a new population.

**Gene flow:** Transfer of genetic information between multiple populations.

**Genetic drift:** Change in phenotype traits due to random sampling in genotype.

**Genotype:** Robot representation in ER that is synonymous to genes or genome in biology.

**Hox gene:** The set of genes in some biological organisms that are responsible for the body plan along the head-tail axis.

**Lego:** A set of bricks that can be assembled into a structure.

**Inertia Measurement Unit:** A system that can measure linear and angular motions along all three axes.

**Markov Decision Process:** A discrete time stochastic control process that makes decisions from an action space with the use of state-action probabilities and rewards.

**Machine Learning:** Study of computer science algorithms that gradually improves itself when performing a particular task.

**Microevolution:** Change in allele frequencies in a biological population that happens over a few generations.

**Monte Carlo methods:** Methods that rely on random sampling to arrive at numerical results.

**Morphology:** The robot body plan or physical description of a robot.

**Mutation:** Unbiased random modifications performed on a single parent to create offspring.

**Numerical explosion:** Phenomenon that causes unreasonably high fitness for evolved robots by accumulating errors during fitness evaluation due to poor virtual simulator accuracy.

**Parent:** Individual robot chosen from a population to create offspring.

**Parent selection:** Methods adopted to select parents, for mating, during evolution.

**Phenotype:** Synonymous to human body where observable properties of the genotype is found.

**Phenotypic plasticity:** Change in phenotype due to environmental feedback.

**Population:** A group of robots that are available to be evolved.

**Reality gap:** The deviation between the behaviour of virtual and real robots.

**Reinforcement Learning:** Class of learning algorithms that learn based on environmental feedback through rewards.

**Speciation:** Process of formation of new species during evolution.

**Supervisory Learning:** Class of learning algorithms that builds a controller from an already available input-output mapping.

**Survivor selection:** Methods used to choose the robots that progress to the next generation.

**Temporal Difference methods:** Similar to Monte Carlo methods but can modify estimates by incorporating the current status.

**Unsupervised Learning:** Class of learning algorithms that find patterns from an unlabelled training data to build a model that fits it.

**Variation operation:** Operations performed on parents to generate an offspring robot.

# REFERENCES


[1] "Autodesk Project Dreamcatcher." [Online]. Available: https://autodeskresearch.com/projects/dreamcatcher. [Accessed: 10-Oct-2018].

[2] "Generative design for lean, additive manufacturing | Frustum," [Online]. Available: https://www.frustum.com. [Accessed: 10-Oct-2018].

[3] R. Volpe, "The CLARAty architecture for robotic autonomy," *Proceedings of the IEEE Aerospace Conference Proceedings*, vol. 1, pp. 1121–1132, 2001.

[4] S. Doncieux, N. Bredeche, J.-B. Mouret, and A. E. Eiben, "Evolutionary Robotics: What, Why, and Where to," *Frontiers in Robotics and AI*, vol. 2, no. 15, p. 1178, 2015.

[5] G. S. Hornby, J. D. Lohn, and D. S. Linden, "Computer-Automated Evolution of an X-Band Antenna for NASA's Space Technology 5 Mission," *Evolutionary Computation*, vol. 19, no. 1, pp. 1–23, 2011.

[6] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. Springer, Berlin, Heidelberg, 2015.

[7] J. Auerbach, D. Aydin, A. Maesani, P. Kornatowski, T. Cieslewski, G. Heitz, P. Fernando, I. Loshchilov, L. Daler, and D. Floreano, "RoboGen: Robot Generation through Artificial Evolution," *Proceedings of the International Conference on the Synthesis and Simulation of Living Systems*, pp. 136–137, 2014.

[8] A. Diaz-Calderon, I. A. D. Nesnas, H. D. Nayar, and W. S. Kim, "Towards a Unified Representation of Mechanisms for Robotic Control Software," *International Journal of Advanced Robotic Systems*, vol. 3, no. 1, pp. 61-66. 2006.

[9] D. Brugali, "Software abstractions for modeling robot mechanisms," *Proceedings of the IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pp. 1–6, 2007.

[10] L. Jingtao and W. Tianmiao, "The modular approach based on functional components division for modular reconfigurable walking robot," *Proceedings of the Conference on Genetic and Evolutionary Computation*, pp. 540–544, 2009.

[11] I. Saha and N. Shankar, "ModelRob: A Simulink Library for Model-Based Development of robot manipulators," *Proceedings of the IEEE International Conference on Robotics and Automation,* pp. 2654–2659, 2012.

[12] J. Kim, K. Kang, and J. Lee, "Survey on automated LEGO assembly construction," *Proceedings of the International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, pp. 89–96, 2014.

[13] T. Kozaki, H. Tedenuma, and T. Maekawa, "Automatic generation of LEGO building instructions from multiple photographic images of real objects," *Computer-Aided Design*, vol. 70, pp. 13–22, 2016.

[14] J. Treanor and G. M. P. OHare, "Lowering the Bar for Robotic Development: Driver Generation for Ubiquitous Robotic Systems," *Proceedings of the 22nd Irish Conference on Artificial Intelligence and Cognitive Science,* pp. 1-10, 2011.

[15] M. Kjaergaard, *Modular Platform for Commercial Mobile Robots*, Technical University of Denmark*,* 2013.

[16] P. Chedmail and E. Ramstein, “Robot mechanism synthesis and genetic algorithms,” *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 4, pp. 3466–3471, 1996.
[17] R. Cruz, and J. M. Ibarra Zannatha, “Efficient mechanical design and limit cycle stability for a humanoid robot: An application of genetic algorithms,” *Neurocomputing*, vol. 233, pp. 72–80, 2017.
[18] K. Sims, “Evolving virtual creatures,” *Proceedings of the Conference on Computer Graphics*, pp. 15–22, 1994.
[19] A. Faiña, F. Bellas, F. Orjales, D. Souto, and R. J. Duro, “An evolution friendly modular architecture to produce feasible robots,” *Robotics and Autonomous Systems*, vol. 63, pp. 195–205, 2015.
[20] C. Darwin, *The origin of species by means of natural selection: or, the preservation of favored races in the struggle for life*. John Murray London, 1859.
[21] G. Mendel, “Experiments in plant hybridization (1865),” in *Classic papers in genetics*, no. 1, J. Peters, Ed. Englewood Cliffs, New Jersey, pp. 1–19, 1959.
[22] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*, MIT Press, Cambridge, MA, USA, 1992.
[23] S. Nolfi, H. Lipson, and D. Floreano, *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines*. MIT Press, Cambridge, MA, USA, 2000.
[24] J. C. Bongard, “Evolutionary robotics,” *Communications of the ACM.*, vol. 56, no. 8, pp. 74–83, 2013.
[25] W. Weimer, T. V. Nguyen, C. Le Goues, and S. Forrest, “Automatically finding patches using genetic programming,” *Proceedings of the International Conference on Software Engineering,* pp. 364–374, 2009.
[26] A. Supady, V. Blum, and C. Baldauf, “First-Principles Molecular Structure Search with a Genetic Algorithm,” *Journal of Chemical Information and Modeling*, vol. 55, no. 11, pp. 2338–2348, 2015.
[27] S. Schulenburg and P. Ross, “Strength and Money: An LCS Approach to Increasing Returns,” in *Advances in Learning Classifier Systems*, vol. 1996, no. 8, Springer, Berlin, Heidelberg, pp. 114–137, 2000.
[28] J. Arifovic, “Genetic algorithm learning and the cobweb model,” *Journal of Economic Dynamics and Control*, vol. 18, no. 1, pp. 3–28, 1994.
[29] T. Baeck, D. B. Fogel, and Z. Michalewicz, *Handbook of Evolutionary Computation*. CRC Press, 1997.
[30] E. L. Lawler, *The Travelling Salesman Problem*. John Wiley & Sons, 1985.
[31] R. S. K. Kwan and P. Mistry, “A co-evolutionary algorithm for train timetabling,” *Proceedings of the Congress on Evolutionary Computation*, vol. 3, pp. 2142–2148, 2003.
[32] Ç. Koç, T. Bektaş, O. Jabali, and G. Laporte, “A hybrid evolutionary algorithm for heterogeneous fleet vehicle routing problems with time windows,” *Computers & Operations Research*, vol. 64, pp. 11–27, 2015.
[33] S. Reddy, “Multi-Objective based Congestion Management using Generation Rescheduling and Load Shedding,” *IEEE Transactions on Power Systems*, pp. 852–863, 2016.
[34] S. K. Goudos, D. Plets, N. Liu, L. Martens, and W. Joseph, “A multi-objective approach to indoor wireless heterogeneous networks planning based on

biogeography-based optimization," *Computer Networks*, vol. 91, pp. 564–576, 2015.

[35] P. Myszkowski and M. Norberciak, "Evolutionary algorithms for timetable problems," *Annales Universitatis Mariae Curie-Sklodowska, sectio AI – Informatica*, vol. 1, no. 1, pp. 1–9, 2015.

[36] L. T. Bui and Z. Michalewicz, "An evolutionary multi-objective approach for dynamic mission planning," *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 1–8, 2010.

[37] A. Ponsich, A. L. Jaimes, and C. A. C. Coello, "A Survey on Multiobjective Evolutionary Algorithms for the Solution of the Portfolio Optimization Problem and Other Finance and Economics Applications," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 3, pp. 321–344, 2013.

[38] A. I. Marqués Marzal, V. García Jiménez, and J. S. Sánchez Garreta, "A literature review on the application of evolutionary computing to credit scoring," *Journal of the Operational Research Society*, vol. 64, no. 9, pp. 1384–1399, 2013.

[39] A. Mukerjee, R. Biswas, K. Deb, and A. P. Mathur, "Multi–objective Evolutionary Algorithms for the Risk–return Trade–off in Bank Loan Management," *International Transactions in Operational Research*, vol. 9, no. 5, pp. 583–597, 2002.

[40] P. P. Bonisone, R. Subbu, and K. S. Aggour, "Evolutionary optimization of fuzzy decision systems for automated insurance underwriting," *Proceedings of the IEEE World Congress on Computational Intelligence,* vol. 2, pp. 1003–1008, 2002.

[41] Y.-S. Hong and R. Bhamidimarri, "Evolutionary self-organising modelling of a municipal wastewater treatment plant," *Water Research*, vol. 37, no. 6, pp. 1199–1212, 2003.

[42] P. J. Fleming and R. C. Purshouse, "Evolutionary algorithms in control systems engineering: a survey," *Control Engineering Practice*, vol. 10, no. 11, pp. 1223–1241, 2002.

[43] R. K. Ursem, *Models for evolutionary algorithms and their applications in system identification and control optimization*. University of Aarhus, 2003.

[44] E. Bilotta, A. Cerasa, Pietro S Pantano, A. Quattrone, A. Staino, and F. Stramandinoli, "Evolving Cellular Neural Networks for the Automated Segmentation of Multiple Sclerosis Lesions.," *Variants of Evolutionary Algorithms for Real-World Applications*, no. 12, pp. 377–412, 2012.

[45] M. Zhang and W. N. Browne, "A Survey on Evolutionary Computation Approaches to Feature Selection," *IEEE Transactions of Evolutionary Computation*, vol. 20, no. 4, pp. 606–626, 2016.

[46] R. Kicinger, T. Arciszewski, and K. De Jong, "Evolutionary computation and structural design: A survey of the state-of-the-art," *Computers and Structures*, vol. 83, no. 23, pp. 1943–1978, 2005.

[47] J. M. Moore and P. K. McKinley, "Evolution of an amphibious robot with passive joints," *Proceedings of the IEEE Congress on Evolutionary Computation,* pp. 1443–1450, 2013.

[48] O. Chocron and P. Bidaud, "Genetic design of 3D modular manipulators," *Proceedings of the International Symposium on Computational Intelligence in Robotics and Automation*, vol. 1, pp. 223–228, 1997.

[49] B. Weel, E. Crosato, J. Heinerman, E. Haasdijk, and A. E. Eiben, “A robotic ecosystem with evolvable minds and bodies,” *Proceedings of the IEEE Conference on Evolvable Systems*, pp. 165–172, 2014.

[50] K. Larpin, S. Pouya, J. van den Kieboom, and A. J. Ijspeert, “Co-evolution of morphology and control of virtual legged robots for a steering task,” *Proceedings of the International Conference on Robotics and Biomimetics*, pp. 2799–2804, 2011.

[51] G. Buason, N. Bergfeldt, and T. Ziemke, “Brains, Bodies, and Beyond: Competitive Co-Evolution of Robot Controllers, Morphologies and Environments,” *Genetic Programming and Evolvable Machines,* vol. 6, no. 1, pp. 25–51. 2005.

[52] K. Endo, T. Maeno, and H. Kitano, “Co-evolution of morphology and walking pattern of biped humanoid robot using evolutionary computation: designing the real robot,” *Proceedings of the IEEE International Conference on Robotics and Automation,* pp. 1362–1367, 2003.

[53] C. Mautner and R. K. Belew, “Evolving robot morphology and control,” *Artificial Life and Robotics,* vol. 4, no. 3, pp. 130–136, 2000.

[54] J. Bongard, “The Utility of Evolving Simulated Robot Morphology Increases with Task Complexity for Object Manipulation,” *Artificial Life,* vol. 16, no. 3, pp. 201–223, 2010.

[55] P. S. Shiakolas, D. Koladiya, and J. Kebrle, “Optimum Robot Design Based on Task Specifications Using Evolutionary Techniques and Kinematic, Dynamic, and Structural Constraints,” *Proceedings of the ASME International Mechnical Engineering Conference and Exposition*, vol. 2002, pp. 825–832, 2002.

[56] M. Rommerman, D. Kuhn, and F. Kirchner, “Robot design for space missions using evolutionary computation,” *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 2098–2105, 2009.

[57] E. Samuelsen, K. Glette, and J. Torresen, “A hox gene inspired generative approach to evolving robot morphology,” *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 751–758, 2013.

[58] D. Lessin, D. Fussell and R. Miikkulainen, “Adopting Morphology to Multiple Tasks in Evolved Virtual Creatures,” *Proceedings of the International Conference on Simulation and Synthesis of Living Systems*, pp. 247–254, 2014.

[59] M. L. Pilat, T. Ito, R. Suzuki, and T. Arita, “Evolution of virtual creature foraging in a physical environment,” *Proceedings of the International Conference on Simulation and Synthesis of Living Systems*, pp. 423–430, 2012.

[60] A. Azarbadegan, F. Broz, and C. L. Nehaniv, “Evolving Sims's creatures for bipedal gait,” *Proceedings of the IEEE Symposium on Artificial Life,* pp. 218–224, 2011.

[61] N. Chaumont, R. Egli, and C. Adami, “Evolving virtual creatures and catapults,” *Artificial Life,* vol. 13, no. 2, pp. 139–157, 2007.

[62] M. O'kelly and K. Hsiao, “Evolving Simulated Mutually Perceptive Creatures for Combat,” *Proceedings of the International Conference on Simulation and Synthesis of Living Systems*, pp. 113–118, 2004.

[63] H. Lipson, “Evolutionary synthesis of kinematic mechanisms,” *Artificial Intelligence for Engineering Design, Analysis and Manufacturing,* vol. 22, no. 3, pp. 195–205, 2008.

[64] M. Gregor, J. Spalek, and J. Capák, “Use of context blocks in genetic programming for evolution of robot morphology,” *Proceedings of the International Conference ELEKTRO*, pp. 286–291, 2012.

[65] I. Macinnes and E. Di Paolo, “Crawling out of the simulation: Evolving real robot morphologies using cheap reusable modules,” *Proceedings of the International Conference on Simulation and Synthesis of Living Systems*, pp. 94–99, 2004.

[66] W. P. Lee, J. Hallam, and H. H. Lund, “A hybrid gp/ga approach for co-evolving controllers and robot bodies to achieve fitness-specified tasks,” *Proceedings of the IEEE International Conference on Evolutionary Computation*, pp. 384–389, 1996.

[67] W. P. Lee, “Evolving Autonomous Robot: From Controller to Morphology,” *IEICE Transactions on Information and Systems*, vol. 83, no. 2, pp. 200–210, 2000.

[68] D. Lessin, D. Fussell, and R. Miikkulainen, “Open-ended behavioral complexity for evolved virtual creatures,” *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 335–342, 2013.

[69] H. H. Lund, “Co-evolving Control and Morphology with LEGO Robots,” in *Morpho-functional Machines: The New Species*, no. 4, Springer, Tokyo, Japan, pp. 59–79, 2003.

[70] S. Risi, D. Cellucci, and H. Lipson, “Ribosomal robots: Evolved designs inspired by protein folding,” *Proceedings of the Annual Conference on Genetic and Evolutionary Computation,* pp. 263–270, 2013.

[71] G. S. Hornby and J. B. Pollack, “Body-brain co-evolution using L-systems as a generative encoding,” *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 868–875, 2001.

[72] A. De Beir and B. Vanderborght, “Evolutionary method for robot morphology: Case study of social robot probo,” *Proceedings of the ACM/IEEE International Conference on Human-Robot Interaction*, pp. 609–610, 2016.

[73] S. H. Lim and J. Teo, “Design, Optimization and Fabrication of a Climbing Six Articulated-Wheeled Robot Using Artificial Evolution and 3D Printing,” *British Journal of Mathematics & Computer Science*, vol. 10, no. 2, pp. 1–21, 2015.

[74] A. J. Clark, J. M. Moore, J. Wang, and X. Tan, “Evolutionary design and experimental validation of a flexible caudal fin for robotic fish,” *Proceedings of the Conference on Simulation and Synthesis of Living Systems*, pp. 325–332, 2012.

[75] Y. S. Shim, S. J. Kim, and C. H. Kim, “Evolving flying creatures with path-following behavior,” *Proceedings of the International Symposium on Computational Intelligence in Robotics and Automation*, pp. 125–132, 2004.

[76] F. Corucci, M. Calisti, H. Hauser, and C. Laschi, “Novelty-Based Evolutionary Design of Morphing Underwater Robots,” *Proceedings of the Annual Conference on Genetic and Evolutionary Computation*, pp. 145–152, 2015.

[77] G. S. Hornby, *Generative representations for evolutionary design automation*. Brandeis University, 2003.

[78] L. Brodbeck, S. Hauser, and F. Iida, “Morphological Evolution of Physical Robots through Model-Free Phenotype Development.,” *PLoS ONE*, vol. 10, no. 6, p. e0128444, 2015.

[79] J. B. Pollack and H. Lipson, “The GOLEM project: Evolving hardware bodies and brains,” *Proceedings of the NASA/DoD Workshop on Evolvable Hardware*, pp. 37–42, 2000.

[80] B. Smith, C. M. Saaj, and E. Allouis, “Evolving legged robots using biologically inspired optimization strategies,” *Proceedings of the IEEE International Conference on Robotics and Biomimetics*, pp. 1335–1340, 2010.

[81] M. Komosiński and S. Ulatowski, “Framsticks: Towards a Simulation of a Nature-Like World, Creatures and Evolution,” in *Applications of Evolutionary Computation*, vol. 1674, no. 33, A. M. Mora and G. Squillero, Eds., Springer, Berlin, Heidelberg, pp. 261–265, 1999.

[82] K. Digumarti, “Concurrent optimization of mechanical design and locomotion control of a legged robot,” *Proceedings of the International Conference on Climbing and Walking Robots and Support Technologies for Mobile Machines*, pp. 315–323, 2014.

[83] J. E. Auerbach and J. C. Bongard, “Evolving complete robots with CPPN-NEAT,” *Proceedings of the Annual Conference on Genetic And Evolutionary Computation,* pp. 1475–1482, 2011.

[84] K. Endo and T. Maeno, “Simultaneous design of morphology of body, neural systems and adaptability to environment of multi-link-type locomotive robots using genetic programming,” *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 4, pp. 2282–2287, 2001.

[85] K. Endo, T. Maeno, and H. Kitano, “Co-evolution of morphology and walking pattern of biped humanoid robot using evolutionary computation. Consideration of characteristic of the servomotors,” *Proceedings of the International Conference on Intelligent Robots and Systems*, pp. 2678–2683, 2002.

[86] S. Rubrecht, E. Singla, V. Padois, P. Bidaud, and M. de Broissia, “Evolutionary Design of a Robotic Manipulator for a Highly Constrained Environment,” in *New Horizons in Evolutionary Robotics*, vol. 341, no. 7, Springer, Berlin, Heidelberg, 2011, pp. 109–121.

[87] M. Mazzapioda, A. Cangelosi, and S. Nolfi, “Evolving morphology and control: A distributed approach,” *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 2217–2224, 2009.

[88] M. R. Heinen, and F. S. Osório, “Evolving morphologies and gaits of physically realistic simulated robots,” *Proceedings of the ACM symposium on Applied Computing,* pp. 1161–1165, 2009.

[89] G. B. Parker, D. Duzevik, A. S. Anev, and R. Georgescu, “Morphological Evolution of Dynamic Structures in a 3-Dimensional Simulated Environment,” *Proceedings of the International Symposium on Computational Intelligence in Robotics and Automation*, pp. 534–540, 2007.

[90] S. Farritor and S. Dubowsky, “On Modular Design of Field Robotic Systems,” *Autonomous Robots,* vol. 10, no. 1, pp. 57–65, 2001.

[91] O. Chocron, “Evolutionary design of modular robotic arms,” *Robotica,* vol. 26, no. 3, pp. 323–330, 2007.

[92] O. Chocron, “Evolving modular robots for rough terrain exploration,” in *Studies in Computational Intelligence*, vol. 50, no. 4, Springer, Berlin, Heidelberg, pp. 23–46, 2007.

[93] W. K. Chung, Jeongheon Han, Y. Youm, and S. H. Kim, “Task based design of modular robot manipulator using efficient genetic algorithm,” *Proceedings of the International Conference on Robotics and Automation*, vol. 1, pp. 507–512, 1997.

[94] T. Miconi and A. Channon, “A virtual creatures model for studies in artificial evolution,” *Proceedings of the IEEE Congress on Evolutionary Computation*, vol. 1, pp. 565–572, 2005.

[95] T. Miconi, “In Silicon No One Can Hear You Scream: Evolving Fighting Creatures,” in *Genetic Programming*, vol. 4971, no. 5802, Springer, Berlin, Heidelberg, pp. 25–36, 2008.

[96] J. B. Pollack, G. S. Hornby, H. Lipson, and P. Funes, “Computer creativity in the automatic design of robots,” *Leonardo*, vol. 36, no. 2, pp. 115–121, 2003.

[97] K. Endo, T. Maeno, and H. Kitano, “Co-evolution of morphology and walking pattern of biped humanoid robot using evolutionary computation - evolutionary designing method and its evaluation,” *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems,* vol. 1, pp. 340–345, 2003.

[98] K. Sims, “Evolving 3D Morphology and Behavior by Competition,” *Artificial Life,* vol. 1, no. 4, pp. 353–372, 1994.

[99] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: NSGA-II,” *IEEE Transactions of Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[100] T. Back, *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, New York, 1996.

[101] K. O. Stanley and R. Miikkulainen, “Evolving Neural Networks through Augmenting Topologies,” *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.

[102] N. Hansen, “The CMA Evolution Strategy: A Tutorial,” *arXiv.org*, vol. 1604. p. arXiv:1604.00772, 2016.

[103] P. Caamaño, R. Tedín, A. Paz-Lopez, and J. A. Becerra, “JEAF: A Java Evolutionary Algorithm Framework,” *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 1–8, 2010.

[104] N. Lassabe, H. Luga, and Y. Duthen, “A new step for artificial creatures,” *Proceedings of the IEEE Symposium on Artificial Life*, pp. 243–250, 2007.

[105] S. Koos, J. B. Mouret, and S. Doncieux, “The Transferability Approach: Crossing the Reality Gap in Evolutionary Robotics,” *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 1, pp. 122–145, 2012.

[106] G. B. Parker and P. J. Nathan, “Co-evolution of sensor morphology and control on a simulated legged robot,” *Proceedings of the International Symposium on Computational Intelligence in Robotics and Automation*, pp. 516–521, 2007.

[107] J.-A. Leal-Naranjo, M. Ceccarelli, C.-R. Torres-San-Miguel, L.-A. Aguilar-Perez, G. Urriolagoitia-Sosa, and G. Urriolagoitia-Calderón, “Multi-objective optimization of a parallel manipulator for the design of a prosthetic arm using genetic algorithms,” *Latin American Journal of Solids and Structures,* vol. 15, no. 3, pp. 1-15, 2018.

[108] G. A. Simon, J. M. Moore, A. J. Clark, and P. K. McKinley, “Evo-ROS,” *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 1386–1393, 2018.

[109] T. Taylor and C. Massey, “Recent Developments in the Evolution of Morphologies and Controllers for Physically Simulated Creatures,” *Artificial Life,* vol. 7, no. 1, pp. 77–87, 2001.

[110] W. P. Lee, “Evolving robot brains and bodies together: An experimental investigation,” *Journal of the Chinese Institute of Engineers*, vol. 26, no. 2, pp. 125–132, 2003.

[111] J. M. Moore and P. K. McKinley, “Evolving flexible joint morphologies,” *Proceedings of the Annual Conference on Genetic and Evolutionary Computation*, pp. 145–152, 2012.

[112] T. F. Nygaard, E. Samuelsen, and K. Glette, “Overcoming Initial Convergence in Multi-objective Evolution of Robot Control and Morphology Using a Two-Phase Approach,” in *Applications of Evolutionary Computation*, vol. 10199, no. 8, Springer, Cham, pp. 825–836, 2017.

[113] P. Krcah, “Effects of morphological plasticity on evolution of virtual robots,” *Adaptive Behaviour*, vol. 25, no. 2, pp. 44-59, 2017.

[114] M. Georgiev, I. Tanev, and K. Shimohara, “Coevolving behavior and morphology of simple agents that model small-scale robots,” *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 1576–1583, 2018.

[115] T. F. Nygaard, C. P. Martin, E. Samuelsen, J. Torresen, and K. Glette, “Real-world evolution adapts robot morphology and control to hardware limitations,” *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 125–132, 2018.

[116] H. Lipson and J. Pollack, “Evolving physical creatures,” *Proceedings of the International Conference on Artificial Life*, pp. 282–287, 2006.

[117] A. Faíña, F. Bellas, F. López-Peña, and R. J. Duro, “EDHMoR: Evolutionary designer of heterogeneous modular robots,” *Engineering Applications of Artificial Intelligence*, vol. 26, no. 10, pp. 2408–2423, 2013.

[118] G. Lan, M. Jelisavcic, D. M. Roijers, E. Haasdijk, and A. E. Eiben, “Directed Locomotion for Modular Robots with Evolvable Morphologies,” in *Parallel Problem Solving from Nature – PPSN XV*, vol. 11101, no. 5802, Springer, Cham, pp. 476–487, 2018.

[119] E. Hupkes, M. Jelisavcic, and A. E. Eiben, “Revolve: A Versatile Simulator for Online Robot Evolution,” in *Applications of Evolutionary Computation*, vol. 10784, no. 7553, Springer, Cham, pp. 687–702, 2018.

[120] N. Jakobi, P. Husbands, and I. Harvey, “Noise and the reality gap: The use of simulation in evolutionary robotics,” in *Advances in Artificial Life*, vol. 929, no. 53, Springer, Berlin, Heidelberg, pp. 704–720, 1995.

[121] R. Konsella, F. Chiarulli, J. Peterson, and J. Rieffel, “A baseline-realistic objective open-ended kinematics simulator for evolutionary robotics,” *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 1113–1116, 2017.

[122] S. Nichele, “The coevolution of robot controllers (” brains”) and morphologies (” bodies”)–challenges and opportunities,” 2015.

[123] A. L. Nelson, G. J. Barlow, and L. Doitsidis, “Fitness functions in evolutionary robotics: A survey and analysis,” *Robotics and Autonomous Systems*, vol. 57, no. 4, pp. 345–370, 2009.

[124] J. E. Auerbach, G. Heitz, P. M. Kornatowski, and D. Floreano, “Rapid Evolution of Robot Gaits,” *Proceedings of the Companion Publication of the Annual Conference on Genetic and Evolutionary Computation*, pp. 743–744, 2015.

[125] “Google Protocol Buffer,” [Online]. Available: *https://developers.google.com/protocol-buffers/* [Accessed: 10-Oct-2018].

[126] L. Yang, J. Qi, D. Song, J. Xiao, J. Han, and Y. Xia, "Survey of Robot 3D Path Planning Algorithms," *Journal of Control Science and Engineering*, vol. 2016, no. 1, pp. 1–22, 2016.

[127] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Iowa State University, 1998.

[128] A. Yershova, L. Jaillet, T. Siméon, and S. M. LaValle, "Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain," *Proceedings of the* IEEE *International Conference on Robotics and Automation*, pp. 3856–3861, 2005.

[129] S. Karaman and E. Frazzoli, "Optimal kinodynamic motion planning using incremental sampling-based methods," *Proceedings of the IEEE Conference on Decision and Control*, pp. 7681–7687, 2010.

[130] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[131] M. I. Shamos and D. Hoey, "Closest-point problems," *Proceedings of the Annual Symposium on Foundations of Computer Science,* pp. 151–162, 1975.

[132] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *Proceedings of the IEEE International Conference on Robotics and Automation,* vol. 2, pp. 500–505, 1985.

[133] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik,* vol. 1, no. 1, pp. 269–271, 1959.

[134] P. Hart, N. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[135] A. Stentz, "Optimal and efficient path planning for partially-known environments," *Proceedings of the IEEE International Conference on Robotics and Automation,* pp. 3310–3317, 1994.

[136] K. Culligan, M. Valenti, Y. Kuwata, and J. P. How, "Three-Dimensional Flight Experiments Using On-Line Mixed-Integer Linear Programming Trajectory Optimization," *Proceedings of the American Control Conference,* pp. 5322–5327, 2007.

[137] J. Tisdale, Z. Kim, and J. Hedrick, "Autonomous UAV path planning and estimation," *IEEE Robotics and Automation Magazine*, vol. 16, no. 2, pp. 35–42, 2009.

[138] Y. Pehlivanoglu, "A new vibrational genetic algorithm enhanced with a Voronoi diagram for path planning of autonomous UAV," *Aerospace Science and Technology*, vol. 16, no. 1, pp. 47–55, 2012.

[139] R. Glasius, A. Komoda, and S. C. A. M. Gielen, "Neural Network Dynamics for Path Planning and Obstacle Avoidance," *Neural Networks*, vol. 8, no. 1, pp. 125–133, 1995.

[140] E. Masehian and M. R. A. Naseri, "A voronoi diagram-visibility graph-potential field compound algorithm for robot path planning," *Journal of Robotic Systems*, vol. 21, no. 6, pp. 275–300, 2004.

[141] X. Zhang, H. Duan, and Y. Yu, "Receding horizon control for multi-UAVs close formation control based on differential evolution," *Science China Information Sciences*, vol. 53, no. 2, pp. 223–235, 2010.

[142] F. Yan, Y.-S. Liu, and J.-Z. Xiao, "Path Planning in Complex 3D Environments Using a Probabilistic Roadmap Method," *International Journal of Automation and Computing*, vol. 10, no. 6, pp. 525–533, 2014.

[143] G. Erinc and S. Carpin, “A genetic algorithm for nonholonomic motion planning,” *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 1843–1849, 2007.
[144] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. The MIT Press, 2018.
[145] J. Kober, J. A. Bagnell, and J. Peters, “Reinforcement learning in robotics: A survey,” *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.
[146] M. P. Deisenroth, “A Survey on Policy Search for Robotics,” *FNT in Robotics*, vol. 2, no. 1, pp. 1–142, 2013.
[147] J. Peters and S. Schaal, “Policy Gradient Methods for Robotics,” *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2219–2225, 2006.
[148] R. J. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” *Machine Learning*, vol. 8, no. 3, pp. 229–256, 1992.
[149] J. Baxter, P. L. Bartlett, and L. Weaver, “Experiments with Infinite-Horizon, Policy-Gradient Estimation,” *Journal of Artificial Intelligence Research*, vol. 15, pp. 351–381, 2001.
[150] P. L. Bartlett and J. Baxter, “Stochastic optimization of controlled partially observable Markov decision processes,” *Proceedings of the IEEE Conference on Decision and Control,* vol. 1, pp. 124–129, 2000.
[151] J. Sorg, R. L. Lewis, and S. P. Singh, “Reward Design via Online Gradient Ascent,” *Proceedings of the International Conference on Neural Information Processing Systems*, pp. 2190–2198, 2010.
[152] D. Aberdeen, *Policy-Gradient Algorithms for Partially Observable Markov Decision Processes,* The Australian National University*,* 2003.
[153] M. Ghavamzadeh and S. Mahadevan, “Hierarchical policy gradient algorithms,” *Proceedings of the Conference on Machine Learning*, vol. 13, no. 2, pp. 197–229, 2006.
[154] F. Sehnke, C. Osendorfer, T. Rückstieß, A. Graves, J. Peters, and J. Schmidhuber, “Parameter-exploring policy gradients,” *Neural Networks*, vol. 23, no. 4, pp. 551–559, 2010.
[155] M. Grüttner, F. Sehnke, T. Schaul, and J. Schmidhuber, “Multi-Dimensional Deep Memory Atari-Go Players for Parameter Exploring Policy Gradients,” in *Artificial Neural Networks – ICANN 2010*, vol. 6353, no. 14, Berlin, Heidelberg, pp. 114–123, 2010.
[156] N. Heess, D. Silver, and Y. T. EWRL, “Actor-Critic Reinforcement Learning with Energy-Based Policies,” *Proceedings of the European Workshop on Reinforcement Learning,* pp. 43–57, 2012.
[157] N. Kohl and P. Stone, “Policy gradient reinforcement learning for fast quadrupedal locomotion,” *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 2619–2624, 2004.
[158] T. Rückstieß, M. Felder, and J. Schmidhuber, “State-Dependent Exploration for Policy Gradient Methods,” in *Machine Learning and Knowledge Discovery in Databases*, vol. 5212, no. 16, Springer, Berlin, Heidelberg, pp. 234–249, 2008.
[159] D. Wierstra, A. Forster, J. Peters, and J. Schmidhuber, “Recurrent policy gradients,” *Logic Journal of IGPL*, vol. 18, no. 5, pp. 620–634, 2009.
[160] D. Wierstra, A. Foerster, J. Peters, and J. Schmidhuber, “Solving Deep Memory POMDPs with Recurrent Policy Gradients,” in *Artificial Neural Networks –*

*ICANN 2007*, vol. 4668, no. 71, Springer, Berlin, Heidelberg, pp. 697–706, 2007.

[161] D. Silver, G. Lever, N. Heess and T. Degris, "Deterministic policy gradient algorithms," *Proceedings of the International Conference on Neural Information Processing Systems*, pp. 387–394, 2014.

[162] G. Endo, J. Morimoto, T. Matsubara, J. Nakanishi, and G. Cheng, "Learning CPG-based Biped Locomotion with a Policy Gradient Method: Application to a Humanoid Robot," *The International Journal of Robotics Research*, vol. 27, no. 2, pp. 213–228, 2008.

[163] K. Chatzilygeroudis, V. Vassiliades, F. Stulp, S. Calinon, and J.-B. Mouret, "A survey on policy search algorithms for learning robot controllers in a handful of trials," *arXiv.org*. 06-Jul-2018.

[164] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[165] R. M. Neal and G. E. Hinton, "A View of the Em Algorithm that Justifies Incremental, Sparse, and other Variants," in *Learning in Graphical Models*, vol. 39, no. 12, Springer, Dordrecht, pp. 355–368, 1998.

[166] G. Neumann, "Variational inference for policy search in changing situations," *Proceedings of the International Conference on Machine Learning*, pp. 817-824, 2011.

[167] J. Peters, K. Mülling, and Y. Altun, "Relative Entropy Policy Search.," *Proceedings of the Conference on Machine Learning*, pp. 1607–1612, 2010.

[168] A. S. Polydoros and L. Nalpantidis, "Survey of Model-Based Reinforcement Learning: Applications on Robotics," *Journal of Intelligent and Robotic Systems*, vol. 86, no. 2, pp. 1–21, 2017.

[169] A. El-Fakdi, M. Carreras, and P. Ridao, "Direct Gradient-Based Reinforcement Learning for Robot Behavior Learning," in *Informatics in Control, Automation and Robotics II*, vol. 42, no. 21, Springer, Dordrecht, pp. 175–182, 2007.

[170] M. R. Shaker, S. Yue, and T. Duckett, "Vision-based reinforcement learning using approximate policy iteration," *Proceedings of the International Conference on Advanced Robotics*, pp 1-8. 2009.

[171] S. Russell, "Learning agents for uncertain environments (extended abstract)," *Proceedings of the Annual Conference on Computational Learning Theory*, pp. 101–103, 1998.

[172] D. Hadfield-Menell, S. Milli, P. Abbeel, S. J. Russell, and A. Dragan, "Inverse Reward Design," *Proceedings of the International Conference on Neural Information Processing Systems*, pp. 6765–6774, 2017.

[173] M. M. Drugan, "Reinforcement learning versus evolutionary computation: A survey on hybrid algorithms," *Swarm and Evolutionary Computation*, vol. 44, pp. 228–246, 2019.

[174] W. D. Hillis, "Co-evolving parasites improve simulated evolution as an optimization procedure," *Physica D: Nonlinear Phenomena*, vol. 42, no. 1, pp. 228–234, 1990.

[175] A. Beigi and N. Mozayani, "A simple interaction model for learner agents: An evolutionary approach," *Journal of Intelligent & Fuzzy System*, vol. 30, no. 5, pp. 2713–2726, 2016.

[176] D. E. Moriarty, A. C. Schultz, and J. J. Grefenstette, “Evolutionary Algorithms for Reinforcement Learning,” *Journal of Artificial Intelligence Research*, vol. 11, pp. 241–276, 1999.

[177] S. Girgin and P. Preux, “Feature Discovery in Reinforcement Learning Using Genetic Programming,” in *Genetic Programming*, vol. 4971, no. 19, Springer, Berlin, Heidelberg, pp. 218–229, 2008.

[178] S. Whiteson and P. Stone, “Evolutionary Function Approximation for Reinforcement Learning,” *Journal of Machine Learning Research*, vol. 7, pp. 877–917, 2006.

[179] F. Stulp and O. Sigaud, “Path Integral Policy Improvement with Covariance Matrix Adaptation,” *Proceedings of the International Conference on Machine Learning,* pp. 1547–1554, 2012.

[180] A. Buzdalova, A. Matveeva, and G. Korneev, “Selection of Auxiliary Objectives with Multi-Objective Reinforcement Learning,” *Proceedings of the Companion Publication of the Annual Conference on Genetic and Evolutionary Computation*, pp. 1177–1180, 2015.

[181] K. L. Downing, “Reinforced Genetic Programming,” *Genetic Programming and Evolvable Machines*, vol. 2, no. 3, pp. 259–288, 2001.

[182] G. Karafotias, A. E. Eiben, and M. Hoogendoorn, “Generic parameter control with reinforcement learning,” *Proceedings of the Annual Conference on Genetic and Evolutionary Computation*, pp. 1319–1326, 2014.

[183] D. Thierens, “An adaptive pursuit strategy for allocating operator probabilities.” *Proceedings of the Annual Conference on Genetic and Evolutionary Computation*, pp. 1539–1546, 2005.

[184] Á. Fialho, L. Da Costa, M. Schoenauer, and M. Sebag, “Extreme Value Based Adaptive Operator Selection,” in *Applications of Evolutionary Computation*, vol. 5199, no. 18, A. M. Mora and G. Squillero, Eds. Springer, Berlin, Heidelberg, pp. 175–184, 2008.

[185] J. E. Pettinger and R. Everson, “Controlling genetic algorithms with reinforcement learning,” *Proceedings of the Annual Conference on Genetic and Evolutionary Computation*, pp. 692–692, 2002.

[186] V. V. Miagkikh and W. F. Punch III, “An approach to solving combinatorial optimization problems using a population of reinforcement learning agents,” *Proceedings of the Annual Conference on Genetic and Evolutionary Computation*, pp. 1358–1365, 1999.

[187] S. Mabu, K. Hirasawa, and J. Hu, “A Graph-Based Evolutionary Algorithm: Genetic Network Programming (GNP) and Its Extension Using Reinforcement Learning,” *Evolutionary Computation*, vol. 15, no. 3, pp. 369–398, 2007.

[188] L. Bull, “A brief history of learning classifier systems: from CS-1 to XCS and its variants,” *Evolutionary Intelligence*, vol. 8, no. 2, pp. 55–70, 2015.

[189] S. Khadka and K. Tumer, “Evolution-Guided Policy Gradient in Reinforcement Learning,” *Proceedings of the* Conference on Neural Information Processing Systems, pp. 1196–1208, 2018.

[190] P. A. Vargas, E. A. Di Paolo, I. Harvey, and P. Husbands, *The horizons of evolutionary robotics*, MIT Press, Cambridge, MA, USA, 2014.

[191] G. Boeing, “Visual Analysis of Nonlinear Dynamical Systems: Chaos, Fractals, Self-Similarity and the Limits of Prediction,” *arXiv.org*, vol. nlin.CD, no. 4. Multidisciplinary Digital Publishing Institute, p. 37, 15-Aug-2016.

[192] R. Plomin, *Blueprint: How DNA Makes Us Who We Are*. The MIT Press, 2018.

[193] A. E.-F. Sencianes and M. C. Pérez, “Policy gradient based Reinforcement Learning for real autonomous underwater cable tracking,” *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3635–3640, 2008.

[194] J. M. Baldwin, “A New Factor in Evolution,” *The American Naturalist*, vol. 30, no. 354, pp. 441–451, 1896.

[195] E. Heard and R. A. Martienssen, “Transgenerational Epigenetic Inheritance: Myths and Mechanisms,” *Cell*, vol. 157, no. 1, pp. 95–109, 2014.

[196] K. Rogers, “What’s the Difference Between a Gene and an Allele?,” [Online] Available: *https://www.britannica.com/story/whats-the-difference-between-a-gene-and-an-allele*. [Accessed: 10-Oct-2018].

[197] R. J. Safran and P. Nosil, “Speciation: the origin of new species,” *Nature Education*, vol. 3, no. 10, 2012.

[198] “Evolution at different scales: micro to macro,” [Online] Available: *https://evolution.berkeley.edu/evolibrary/article/evoscales_01*. [Accessed: 10-Oct-2018].

[199] L. J. Harmon and S. Braude, “Conservation of small populations: Effective population sizes, inbreeding, and the 50/500 rule,” in *An Introduction to Methods and Models in Ecology, Evolution, and Conservation Biology*, pp. 125–138, 2010.

[200] J.-B. Mouret and S. Doncieux, “Overcoming the bootstrap problem in evolutionary robotics using behavioral diversity,” *Proceedings of the IEEE Congress on Evolutionary Computation,* pp. 1161–1168, 2009.

[201] A. E. Eiben and J. E. Smith, “Towards the evolution of things,” *SIGEVOlution*, vol. 8, no. 3, pp. 3–6, 2016.

[202] A. Rosendo, M. von Atzigen, and F. Iida, “The trade-off between morphology and control in the co-optimized design of robots,” *PLoS ONE*, vol. 12, no. 10, pp. e0186107–14, 2017.