The University of Southern Mississippi

GUIDED GENETIC EVOLUTION:

A FRAMEWORK FOR THE EVOLUTION OF AUTONOMOUS ROBOTIC CONTROLLERS

by

Khaled El-Sawi

A Dissertation
Submitted to the Graduate Studies Office
of The University of Southern Mississippi
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy

August 2006

ABSTRACT

GUIDED GENETIC EVOLUTION:

A FRAMEWORK FOR THE EVOLUTION OF AUTONOMOUS ROBOTIC CONTROLLERS

by

Khaled El-Sawi

August 2006

The development of autonomous robotic agents capable of complex navigation, control and planning has always been an intriguing area of research. The benefits associated with the successful implementation of such systems are enormous. However, the creation of robotic controllers for the efficient manipulation of autonomous agents in real-time is a very computationally complex task. Such complexity increases exponentially as the structure of the robot or its surrounding environment increase in sophistication. We propose a new genetic framework labeled *Guided Genetic Evolution*, or GGE. The guided genetic evolution platform encapsulates a connectionist model, labeled *Trigger Networks*, for the representation of articulated robotic structures as well as the behavioral capabilities of robotic agents. The evolution of trigger networks is based upon genetic programming methodologies with the inclusion of specialized algorithms for the evolution of articulated robotic controllers. Evolutionary guidance constructs are also introduced as means for minimizing the search space associated with the control problem and achieving successful evolution of agents in a shorter time duration. A simulation environment based on rigid body dynamics is utilized for the functional modeling of system interactions. The simulation environment allows for the utilization of minimal agent representation in order to achieve reliable fitness allowing for the further expansion of the research into the real domain.

The University of Southern Mississippi

GUIDED GENETIC EVOLUTION:

A FRAMEWORK FOR THE EVOLUTION OF AUTONOMOUS

ROBOTIC CONTROLLERS

by

Khaled El-Sawi

A Dissertation
Submitted to the Graduate Studies Office
of The University of Southern Mississippi
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy

Approved:

_____

Director

_____

_____

_____

_____

_____

_____

University Coordinator, Graduate Studies

August 2006

TO MY FAMILY

ACKNOWLEDGMENTS

I would like to extend my gratitude to the thesis director, Dr. Adel Ali, and other committee members, Dr. Dia Ali, Dr. Beddhu Murali, Dr. Ray Seyfarth, Dr. Andrew Strelzoff, and Dr. Ras Pandey for their support, insight and assistance which have been of extreme value to me throughout the duration of this thesis.

I would like to specially thank Dr. Adel Ali for his support over the many years I have known him. The attention, friendship and continuous support he has given me over the years are things I am proud of and will always continue to treasure. I would also like to deeply thank Dr. Dia Ali who has always exemplified a rare image of caring and selfless giving. Dr. Dia's continuous support and positive influence have been beyond words or description. I am also extremely grateful for Dr. Murali's insight and critical view which have helped me in continuously improving my research. Dr. Seyfarth has been of tremendous assistance to me, and he has contributed greatly to the editing of the manuscript. I am quite grateful for his recommendations and effort. Dr. Pandey has been a wonderful force of knowledge and encouragement over the past many months. I am quite appreciative of his belief in me and his constant support. I am also very thankful for Dr. Strelzoff. His recommendations have helped me greatly in guiding my work and formulating ideas for future research.

I would specially like to thank Dr. Rex Gandy and the School of Computing. Dr. Gandy's continued support over the past several years has meant a great deal to me. I am quite appreciative of all his assistance over the years. I would also like to thank Dr. Joseph Kolibal for his valuable assistance with Latex and with the formatting of this manuscript.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure

LIST OF TABLES

Table

Chapter 1

INTRODUCTION

1.1 Autonomous Robotic Control

The creation of autonomous robots capable of complex navigation, control and planning has always been an intriguing area of research. Today, various classes of applications based on autonomous robotic control are being investigated, including those relating to hazardous environments, transportation, service robotics, and so forth [82]. The benefits associated with the successful implementation of such systems are enormous. However, the design of robotic controllers for the efficient manipulation of autonomous robotic agents in real-time can be a very computationally complex task. Such complexity increases exponentially as the structure of the robot or its surrounding environment increase in sophistication. The analysis of an agent's interaction with the environment is still largely unexplored due to the difficulty involved in designing systems that exploit sensory-motor coordination [97]. Each action performed by the a robotic agent forces changes to its internal and external state vectors, which in turn affects its decisions regarding successive actions. Hence, the ability to predict the consequences of a particular decision can be an extremely complex yet crucial component in the design of autonomous robots.

In behavior-based robotics, the modular *divide and conquer* approach is usually utilized to reduce the complexity of robotic controller design by partitioning the control problem into manageable sub-parts. This method allows the designer to design each module independently solving a single problem at a time. The control system is implemented layer by layer with each layer taking the responsibility for carrying out a particular basic task [105]. However, several problems exist with this approach [53]:

- The decomposition method for the robotic control system as a whole might not be apparent. Hence, the division lines chosen by the designer may or may not be the most efficient.
- Interactivity among the different controller sub-parts provide an incomplete view of the controller state as the interactivity with the environment must also be considered.

- As the number of controller sub-parts increases, the number of potential interactions grows exponentially possibly going beyond the designer's capabilities to define the correlations between the different system layers.

1.2 Evolutionary Robotics

Evolutionary robotics [60] is a powerful framework used for the creation of self-organizing robotic controllers capable of learning new behaviors based on their own interactions with the environment. This approach relieves the designer from the need to partition the agent's behavior space or the need to map the interactions between the different system components. The controller learning process is based on a genetic approach where an agent population is artificially evolved based on each individual's ability to perform a given task. Genetic algorithms (GA) are mostly used as the evolution mechanism utilizing a fitness function as a measure of each agent's performance. In its general form, GA aims to produce solutions to optimization problems relating to large search spaces of high dimensionality [53]. Learning takes place through the construction of new generations of individuals utilizing genetic selection, crossover and random mutation. This evolutionary cycle continues until the overall population fitness ceases to increase.

1.2.1 Limitations of Evolutionary Techniques

Evolutionary techniques can be an essential part in the design of self-organized intelligent behavior. However, some problems do exist that can greatly limit the potential of evolutionary robotics. Although the evolutionary process strives to reach a level of convergence in performance, once system equilibrium has been reached, the further evolvability of the system may not be determined with any level of certainty. Harvey argues that after the initial system convergence has been reached, only then can the true evolutionary work begin [51]. Jakobi and Quinn acknowledge the same problem focusing on the importance of the crossover and mutation parameters as tools for continued evolution [68]. However, ascertaining the most appropriate values for the genetic parameters can still be a significantly difficult task, specially when utilizing genomes that vary in size, which is usually the case when the representative architecture itself is also evolving.

Another critical limitation of current evolutionary robotics methods stems from the agent's lack of awareness of its immediate environment and its active role in it. The agent usually follows a trial-and-error approach based solely on actions taken and the consequences of those actions determined primarily by the resulting values of the fitness func-

tion. As the search space increases in size, the likelihood of converging to sub-optimal solutions also increases. The existence of an extremely large state space, also called *state explosion*, remains a fundamental problem in model optimization [46]. The inverted pendulum problem, for example, as described by Sutten [127], has a search space of 2^{162}. Due to the size of the search space as well as the lack of a particular evolutionary strategy, experimentation shows that using evolutionary techniques alone can yield sub-optimal results that fall short of solving the problem (Section 2.5). Hence, the development of methods for reducing the search space size or guiding the evolutionary process using loosely pre-defined strategies can increase the probability of accurate convergence.

1.3 Imitation-based Learning

Robotics research has recently gained more interest in imitation-based learning, also called ”learning by watching” or ”learning by example” [110]. Researchers now feel that the study of imitation-based learning could be the route to the creation of fully autonomous robots [116] and could possibly revolutionize robot-environment interactions by providing new and flexible methods for robot programming [21]. Meltzoff suggests the partitioning of the imitative progression into four stages [92]:

- *Body babbling*: This is an essential element which facilitates the connection between muscle movements and different body configurations. Usually, a trial-and-error approach is utilized where random muscle triggers take place while the resulting configurations are observed and recorded. Eventually, a mapping, or schema, is created linking body movements to potential resultant states.

- *Imitation of body movements*: The body schema is utilized to try and imitate an observed movement through the usage of a probabilistic method of determining the muscle groups that could contribute to a successful imitation.

- *Imitation of actions on objects*: A more advanced form of imitation where body movements are utilized at a higher level to interact with environment objects.

- *Imitation based on inferring intentions of actions*: This stage involves an understanding of not only the surface actions, but also the embedded intention associated with performing those action.

From a human perspective, research has proven imitation to be a very significant contributor to social learning at many levels. “Mirror” neurons have been discovered whose

sole purpose is to fire when movement observance takes place or when similar movements are executed by the observer [100]. From a robotic perspective, imitation can allow for interaction biasing in relation to the agent and the environment; in addition, it can be a crucial tool for constraining the search space for learning [21]. Imitation could also be utilized as a tool for acquiring new behaviors as well as adapting existing behaviors using new contexts [24].

1.3.1 Problems in Robot Imitation

Despite the potential advantages associated with imitation-based learning in robots, many hurdles still face researchers and the research community has only begun to address such issues[21]. We focus on four main imitation-related problems:

- *The Correspondence Problem*: In order for imitation to be successful, an explicit correlation must exist between the learner and the demonstrator [91]. This can be a difficult problem, specially when the body representations of the learner and the demonstrator differ.

- *The When Problem*: At what instance in time should the learner be imitating? The learner must be able to determine the appropriateness of imitation at a specific time based on the current context as well as the learner's internal goals and motivations [21].

- *The What problem*: The learner should be able to selectively utilize parts of its sensory input streams as the basis for its imitative process. This requires a level of relevancy determination.

- *The Inference problem*: How can the robot infer the intentions, perceptions and emotions of the demonstrator that initiate the visible actions observed? The ability to perceive beyond the surface behavior to infer the underlying intentions of the demonstrator is considered the most sophisticated form of imitative learning [110].

The problems mentioned constitute formidable hurdles in the path of imitation-based learning. Current research, as in [21, 55, 116, 117], utilizes saliency as well as system simplification to abridge the imitation problem. However, in order to fully achieve imitation-based learning in robots, solutions must exist to these problems, or different formulations must evolve that render such problems irrelevant.

1.4 Genetic Programming

A variation of direct imitation-based learning could rely on a programmatic approach for dictating rules that govern the learning environment. Such rules could contribute to the learning process by surpassing some of the critical problems associated with robot imitation. This rule-based approach could still benefit from evolutionary methods in order to evolve the most optimal set of governing rules. Genetic programming (GP) [74] could be utilized as the evolution vehicle for this approach.

Genetic programming is an extension of genetic algorithms. Instead of evolving chromosomes of individuals, as in GA, GP works on evolving a program that efficiently solve a given problem. In GP, a program is represented using a tree structure where the internal nodes of the tree represent the set of functions upon which the program is based, and the external nodes represent variables and constants used as function parameters [75]. The main benefit of GP lies in the fact that once the evolutionary phase is complete, a method is produced instead of just a point solution[30]. As the system output is in the form of a program, it can better adapt to situational variance by following the resultant algorithm produced. In essence, GP strives to find an appropriate representation of the problem, which is critical to the solution [134], through an evolutionary process.

Genetic programming offers a more flexible approach to evolution than genetic algorithms. However, GP follows the same GA combined representation of the genome (chromosome) and phenome (individual) as a single entity. Such representation as well as the main structure of GP-based evolution results in several limitations:

- GP evolved structures tend to drift towards large and slow solutions on average [114], so even if the solution is correct, it might not be the most efficient.
- If the genetic code is easy to manipulate, it loses its functional complexity [37].
- If functional complexity does exist, the nature of the genetic code manipulation makes the results extremely difficult to reproduce with modification.
- GP suffers from the same GA problems relating to insufficient diversity and the possibility of reaching sub-optimal solutions [30].

Gene expression programming (GEP) was invented by Ferreira [37] to overcome some of the limitations of GP. The main contribution of GEP is the separation of the genome from its representation. The genome is structured as a linear symbolic string of fixed

length and is converted to its expression tree (ET) representation utilizing a specialized language known as Karva. Although GEP solves the representational problem associated with GP, it still suffers from the problem of insufficient diversity and the possibility for sub-optimal convergence. In addition, both GP and GEP are generalized genetic methods. The presence of specific genetic constructs for the development of intelligent robotic controllers in particular is an essential yet missing element. For example, using the current formulation, it is not feasible to define a specific sequence for function execution. From a robot imitation perspective, the specification of execution sequences would be an essential component of the ”learning by example” approach, however, this element is missing from both GP and GEP methodologies.

1.5 Situation and State Awareness

In addition to an agent's ability to learn and execute primitive behaviors, an essential part of a robot's ability to strategize lies in its own awareness of its current state in relation to the surrounding environment. Situation awareness (SA) relates to an agent's ability to analyze and understand the different parameters of both its internal and external environments in order to make informed decisions. An intelligent agent may rely on sensory inputs alone in order to decide on its next course of action; however, awareness of the meaning of such sensory states adds to the agent's ability to plan and strategize effectively. SA essentially revolves around the understanding of information and the meaning of such information in relation to the present and future of an agent's life cycle [125]. Situation awareness is also a key element in the formulation of a genetic approach for agent planning. In order to achieve its main goal, an agent must build a strategy for transporting itself from one state to the next, until the final objective is reached. Endsley [36] defines SA as consisting of two main partitions:

- Comprehension of the agent's current state (both internal and external) in relation to time and space.
- Projection of the agent's near future status.

Several formulations currently exist for formally describing the state of an agent and its environment. All existing formulations deal with the situation object from a general sense by describing both the states of objects in the environment as well as actions that could be executed within the environment. However, none of the existing methods expand

their constructs to include an agent's ability to transition from its current known state to a future desired state.

1.6 Thesis Overview

The aim of this research is to formulate a new framework for the successful evolution of robotic controllers for the goal-based manipulation of autonomous robotic agents in real-time. The framework introduces a new genetic approach labeled *Guided Genetic Evolution*, or GGE. The guided genetic evolution platform encapsulates a connectionist model, labeled *Trigger Networks*, for the representation of articulated robotic structures as well as the behavioral capabilities of robotic agents. The evolution of trigger networks is based upon genetic programming methodologies with the inclusion of specialized algorithms for the evolution of articulated robotic controllers. Evolutionary guidance constructs are also introduced as means for minimizing the search space associated with the control problem and achieving successful evolution of agents in a shorter time duration.

A simulation environment based on rigid body dynamics is utilized for the functional modeling of system interactions. The simulation environment allows for the utilization of minimal agent representation in order to achieve reliable fitness allowing for the further expansion of the research into the real domain.

1.7 Thesis Contribution

The proposed guided genetic evolution platform adds unique elements to current known evolutionary techniques. Those elements have not been used in any existing genetic evolution framework, to the author's knowledge. GGE is unique in several respects:

1. A new connectionist model, labeled *Trigger Networks*, is created for the encoding of agent attributes and control capabilities. The model offers a high level descriptive structure for the representation of control strategies of any level of sophistication for the control of articulated robots. Trigger networks offer a time-based model for the description of execution sequencing as well as control urgency associated with each of the robotic joints.

2. A genetic evolution algorithm is formulated for the evolution of trigger networks based on one or more fitness functions associated with the desired behaviors. The algorithms presented as part of the evolution framework allows for the processing of trigger networks through genetic selection, crossover, and mutation operators

over multiple generations in an effort to achieve successful fulfillment of the preset behavioral goals.

3. Mechanisms for guiding the genetic process are formulated in order to reduce the network convergence time and increase the quality of the convergence results.

4. The framework allows for the inclusion of *learning by example* techniques in robotic evolution while circumventing the current existing limitations that render such techniques unachievable in a practical sense.

5. The framework is successfully utilized for the control of biped robot balancing and walking behaviors in addition to other classes of robotic control. Although successful biped mobility has been achieved utilizing different types of control strategies, the genetic approach presented offers a high level of flexibility and expandability.

1.8 Summary

The successfully evolution of complex robotic controllers for the manipulation of autonomous robots could revolutionize the design and implementation of intelligent robotic agents. Until today, the complexity of the behavioral interaction models of robots have been prohibitive from a practical sense hindering any significant advancement in the design of autonomous articulated robots. The approach offered by guided genetic evolution aims to circumvent many problems associated with current methodologies in order to advance the fields of autonomous agent design and implementation.

Chapter 2

EVOLUTIONARY ROBOTICS

2.1 Introduction

Autonomous robotic motion control is a very intriguing problem that has prompted exploration in many areas of research. An autonomous robot is an independent entity capable of making intelligent decisions about its environment without any explicit human intervention. Such a robot should be capable of successfully navigating its environment while traversing its decision space and executing planned strategies that would allow it to achieve its goals, both immediate and long term. The complexity associated with creating such systems lies in the complexity of modelling the interactivity that takes place within the robotic agent as well as between the agent and its environment.

Most current research exploring the area of autonomous robot design ignores the complex problem of dynamic motion control relying heavily on the utilization of wheel-based robots. Such robots mainly require an evolved decision-making mechanism capable of controlling their basic locomotion tasks without any need for articulated control at any level. The utilization of wheel-based locomotion also reduces the complexity of the interactivity model between the agent and its environment by reducing the number of variables associated with the control problem.

The creation of robotic controllers capable of efficient decision making based on articulated structures requires the existence of a mechanism for managing and reducing the complexity of the control system. Behavior-based robotics rely on a divide and conquer approach in order to partition the problem space into more manageable sub-parts. The system is then structured as layers with each layer responsible for controlling a single basic task. However, the divide and conquer approach has some significant limitations[53]. Mainly, the system decomposition task is limited by the abilities of the designer. As the number of partitions increase, so will the number of interactions that exist among the system sub-parts possibly going beyond the capabilities of the designer.

Evolutionary robotics is a methodology for the design of self-organizing robotic controllers that operate autonomously in real environments. Utilizing this approach, the designer plays a less active role in the organization of system divisions as the basic system behaviors emerge dynamically as a result of the interactions between the agent and the

environment [105]. This method relies on the artificial evolution of an agent population whose characteristics are encoded as artificial chromosomes. Each member of the population is tested to determine its success in performing a particular given task. Agent performance is then evaluated based on a fitness function that measures the agent's ability to produce the desired results. Only individuals scoring the highest performance levels are allowed to further participate in the evolutionary process. In the case of genetic algorithms, a new population of chromosomes is produced through selective reproduction, crossover and random mutation. This evolutionary process continues until the overall performance of the population seizes to increase.

2.2 Evolving in Simulation

The evolution of robotic agents is usually performed in simulation due to the large number of iterations required to produce successful results. Also, the unexpected behavior associated with the initial population of agents renders them potentially harmful to themselves and to their surrounding environment. However, the effectiveness of evolution in simulation is a largely debated topic. Brooks [22] was skeptical in regards to the problems that might exist due to the use of simulators and the difficulty of accurately simulating real world dynamics. Miglino [95] lists some of the factors that contribute to the difficulties involved in developing control systems for real robots through the use of computer models. He argues that numerical simulations do not cover all the physical laws that govern the interactions between the agent and the environment. Also, physical sensors usually retain uncertain values and approximations while computer models usually return perfect sensory information. Finally, Miglino argues that different physical sensors frequently perform differently due to slight differences in their physical makeup, while this fact is usually ignored when building simulated environments.

2.2.1 Bridging the Gap

Although the problems resulting from the discrepancies present between a simulated environment and the real world must be acknowledged and considered, the careful study of such problems could introduce solutions for bridging the gap between the two environments making simulation-based training more effective. In [62] and [63], arguments are made on how to reduce the problems associated with simulations in order to produce more accurate results. The following are some of the methods through which more precise simulated training environments may be achieved.

- The design of the simulation should be based largely on appropriate quantities of real world data. The data should be regularly validated making the appropriate adjustments to the environment.
- The introduction of noise should be considered at all levels of the simulation allowing for the simulated environment to better represent real world inconsistencies and imprecision.
- The utilization of adaptive noise tolerant units as part of the design will allow the final controller to adapt to the differences between the simulation and the real world.

In order for the evolutionary process to be reliably fit, sufficient conditions must be set forth for the transfer of evolved controllers from simulation to reality. If evolving controllers are forced to satisfy such transfer constraints, then despite the inaccuracy or incompleteness present in the simulated environment, the evolved controller should still transfer into reality [67].

2.2.2 System Modeling

The functional modeling of the relationships between the agent, its goals, and its environment must be present in order to successfully model the constraints needed to achieve reliable fitness. A comprehensive model of the agent's internal state vector, external state vector, as well as the agent's goal priority vector is needed. As Figure 2.1 shows, the core system components are tightly connected based on the given causality model. As the agent changes its internal state, it forces changes to the external environment vector, which might or might not cause further change in the state of the agent. Similarly, the agent's current goal priority vector will be re-prioritized as the state of the agent changes. Different goal priorities affects the controller's subsequent decision patterns.

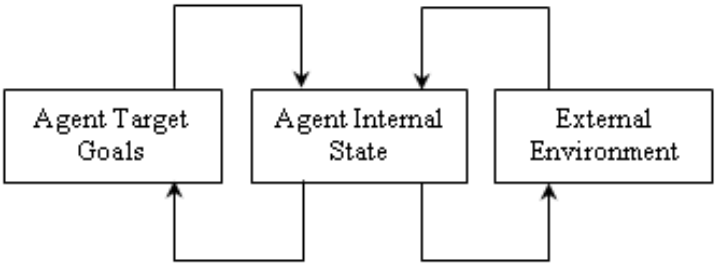


Figure 2.1: Agent-environment causality diagram.

In [65], a formulation is given for the accurate representation of the way in which the internal state of an agent-environment system changes over time. We consider $\vec{s}_t$ to be a state vector representing the agent's various internal state variables s_i at time t. The value $\vec{s}_{t+1}$ is a function of $\vec{s}_t$, the sensory input at time t, represented as $\vec{i}_t$ and the agent's goal priority vector given by $\vec{g}_t$. Hence, given the function S_1 which defines the state transformation of the agent's internal state system over time, $\vec{s}_{t+1}$ is given by

$$\vec{s}_{t+1} = S_1(\vec{i}_t, \vec{s}_t, \vec{g}_t) \tag{2.1}$$

Similarly, the external state vector of the environment at time $t+1$, given by $\vec{e}_{t+1}$, is a function of the state of the environment at time t and the state of the agent at time $t+1$. The state of the agent's external environment might or might not be modified by a new agent state. The function E_1 defines the state transformation function for the environment over time given the state of the agent. $\vec{e}_{t+1}$ is given by

$$\vec{e}_{t+1} = E_1(\vec{e}_t, \vec{s}_{t+1}) \tag{2.2}$$

The agent's sensory input is clearly a function of the external environment. Whether working in simulation or in the real world, the presence of noise (either real or simulated) would cause the agent's sensory input to be only an approximation of the external environment state and not an exact match. Consequently, the sensory input vector $\vec{i}_t$ is a function of the current state of the environment $\vec{e}_t$. We define the function I_1 to define the translation between the external environment state and what is being sensed by the agent.

$$\vec{i}_t = I_1(\vec{e}_t) \tag{2.3}$$

Also, given the function S_2 which defines the way in which motor signals are generated by the controller, the vector $\vec{o}_t$ representing the generation of motor signals is given by

$$\vec{o}_t = S_2(\vec{s}_t) \tag{2.4}$$

In order to simulate a real world environment, noise is added to the motor manipulation signals within the environment. Consequently, the generation of motor signals might or might not succeed due to various conditions. A guarantee constraint must be built into the simulated environment to guarantee the realistic application of control signals. For example, if an agent tries to transition to state $\vec{s}_{target}$ given state $\vec{s}_{initial}$ and the current state of the environment $\vec{i}_{initial}$, the control signal vector $\vec{o}_1$ will be produced. If the agent

fails to achieve the desired state by applying the control signals decided upon, a new set of signals must be generated to gracefully return the agent to the previous state $\vec{s}_{initial}$. Alternatively, the controller might decide not to return to a previous state and instead apply control signal vector $\vec{o}_2$ to transition to a new state other than $\vec{s}_{initial}$ or $\vec{s}_{target}$. The possible transition scenarios are shown in Figure 2.2.

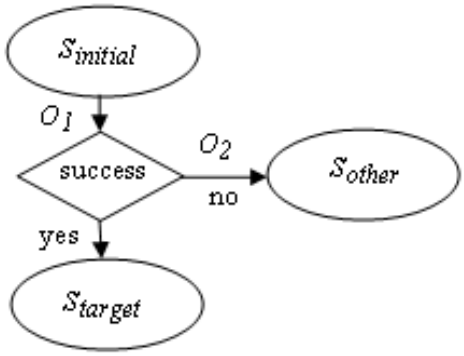


Figure 2.2: State transitions due to motor signal application where an alternative state is chosen given the failure to achieve a target state.

The agent's goal vector $\vec{g}_t$ is dependant on the internal state of the agent $\vec{s}_t$ as well as the state of the environment $\vec{e}_t$. Given the function G_1 that defines the agent goal transformations, the goal state vector is given by

$$\vec{g}_t = G_1(\vec{s}_t, \vec{e}_t) \tag{2.5}$$

The goal vector will need re-prioritization in relation to new agent states reached. For example, the existence of a scenario where the agent is not balanced will prompt an immediate goal to correct the imbalance situation before proceeding to fulfill other goals on the agenda. The goal vector will need to follow continuous revisions and adjustments as the system progresses. To deal with such needs, the simulator will have to offer a dynamic model for the presentation of goals as well as an intelligent re-organization of goals with every time step. The evolutionary process plays an important role in the creation of a dynamic decision making mechanism capable of learning and adapting to rapid system flux.

The progression of the agent-environment system can be described by the following five equations:

$$\begin{array}{ll} \vec{s}_{t+1} = S_1(\vec{i}_t, \vec{s}_t, \vec{g}_t) & \vec{e}_{t+1} = E_1(\vec{e}_t, \vec{s}_{t+1}) \\ \vec{i}_t = I_1(\vec{e}_t) & \vec{o}_t = S_2(\vec{s}_t) \\ \vec{g}_t = G_1(\vec{s}_t, \vec{e}_t) & \end{array} \tag{2.6}$$

The interdependency present between the different modules calls for a systematic approach for system transitioning considering all the relationships present. Intelligence and learning must also be core elements of the decision making mechanism in order to evolve populations that are reliably fit.

2.2.3 Minimal Simulation

The comprehensive modeling of interdependent system components can produce accurate evolutionary results in simulation. However, in order to guarantee the reliable translation of those results into real robots, Jakobi proposes the design of minimal simulations using specific guidelines to ease the transfer of evolutionary results [65, 68]. The core design principles proposed by Jakobi are as follows:

1. A limited base set of agent-environment interactions involved in the execution of a particular behavior should be identified. The simulation should be designed around the base set leaving other interactions to be rooted in the real world. This approach would allow for the mixing of simulated and real environment parameters yielding a smoother transition into physical agents.

2. Different implementation aspects of the simulation must be randomly varied during the evolutionary process allowing the evolving population to develop a level of adaptability to a changing environment. Enough variation must be included so that the agents will evolve without dependence on specific implementation aspects.

3. The base set parameters must also be randomly varied from generation to generation and from trial to trial. This variance will increase the presence of reliably fit agents within the evolved population as agents will be able to cope with changing environment parameters.

The minimal simulation approach increases the success rate of evolving real world controllers. The alternative would be to process a significantly higher number of fitness evaluations, which can be very time-consuming causing all the speed advantages of simulation-based evolution to be lost.

2.3 Genetic Algorithms

Evolutionary robotics [60] aim to develop an agent controller based on an adaptive artificial neural network [105]. Genetic algorithms (GA) are usually used as a teaching vehicle through which the neural network can be trained. In its general form, GA methods can be seen as a solution to optimization problems relating to a large search space of high dimensionality [53]. Genetic algorithms are probabilistic search algorithms where N potential solutions of an optimization problem sample the search space [16]. A genetic algorithm uses a selective reproduction approach operating on a population of abstract representations, or artificial chromosomes. In most cases, a chromosome (genome or genotype) is structured as a string that represents a set of parameters relating to the evolutionary problem under consideration. A binary representation of the value of function variables to be optimized, or the connection weights of an artificial neural network, are examples of the type of encoding a chromosome could hold. Figure 2.3 shows an example of such an encoding [97]. In a typical robotics application, a genotype would represent a parameter of the agent controller in need of optimization. In order to evolve a controller neural network, the floating point values defining the weights of the network nodes can be encoded as integer values to be represented in the chromosome.

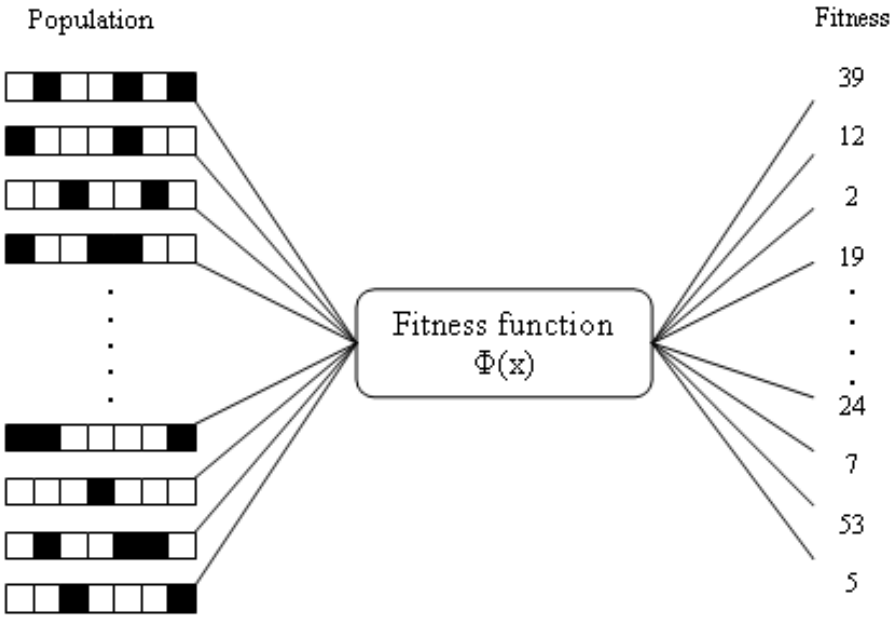


Figure 2.3: The parameters encoded within the chromosomes are represented as binary 0's (white) or 1's (black) and combined to form the value of the variable x to be fed into the fitness function for evaluation.

The evolutionary process typically starts with a population of randomly encoded agents effectively sampling the entire search space associated with the control problem. The evaluation of individuals takes place based on a well defined fitness function which represents a performance measure upon which selection decisions are made. Individuals scoring highest are allowed to reproduce sexually or asexually while others are eliminated from the mix. The genetic algorithm evaluation and selection process is represented in Figure 2.4 [53].

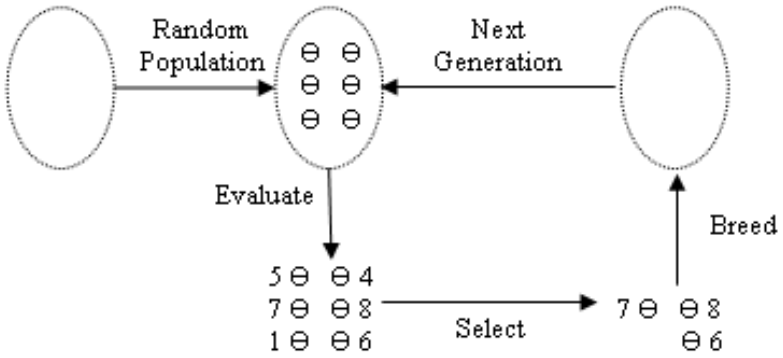


Figure 2.4: The genetic algorithm cycle of evaluation and selection.

2.3.1 Initialization

The initial population of individuals must be carefully initialized to best suit the nature of the problem being investigated. An initialization that is most suitable to the problem at hand would allow for faster population convergence. On the other hand, an inappropriate initial selection could result in a lack of diversity causing premature convergence to a solution that is possibly sub-optimal. Several methods could be utilized to generate the initial population of individuals [41]:

- Random Initialization: A popular method where the population is chosen randomly covering the entire search space with uniform distribution.
- Grid Initialization: The search space is divided into multiple intervals of a specific size depending on the nature of the problem. The population is seeded using independent selection from the defined intervals.
- Non-clustering Initialization: This method guarantees an even distribution by placing a restriction on the initialization process where each individual placed must be

a predefined distance away from individuals who have already been placed.

2.3.2 Selective Reproduction

Lets consider a population of individuals whose chromosomes c_i are encoded as fixed length binary strings from the set

$$C = \{0,1\}^n$$

where n is the length of the string encoding. Given a population of size m, the entire generation G at time t could be represented as [17]

$$G_t = (c_{1t}, c_{2t}, ..., c_{mt})$$

Selective reproduction is based on selecting individuals with the best performance record and making copies of their chromosomes. The next generation will include a higher number of copies of chromosomes belonging to individuals whose performance was superior in previous generations. A selection operator is utilized to improve the performance quality of a population by allowing individuals of higher quality a higher probability of advancing to the next generation [16]. The *roulette wheel* is a genetic selection operator used to implement selective reproduction. The concept behind the roulette wheel selection method is that each individual in the population has a chance to become a member in the next generation of individuals, and that chance is proportional to the performance of this individual. Each slot in the wheel corresponds to an individual in the population, and the size of each slot is representative of the individual's fitness. More precisely, given an individual denoted as c_j whose fitness at time t is defined as $f(c_{j,t})$, the size of the wheel slot $P[c_{j,t}]$ corresponds to the fitness value of the individual normalized by the total fitness of m individuals in the population.

$$P[c_{j,t}] = \frac{f(c_{j,t})}{\sum_{k=1}^{m} f(c_{k,t})} \tag{2.7}$$

$P[c_{j,t}]$ represents the probability of an individual for being chosen for reproduction. After spinning the wheel N times, the expected number of children fathered by individual j is $NP[c_{j,t}]$. There are two main drawbacks associated with using the roulette wheel method. First, there are instances where the fitness results must be sorted in order to allow for the proper distribution of probabilities, which is a computationally expensive task and might not be practical for large population sizes. Second, the fitness function utilized must yield positive results. If that is not the case, a non-decreasing transformation

$\varphi : \mathbb{R} \rightarrow \mathbb{R}^+$ must be applied to shift the values to a usable range [17]. The probabilities would then be defined as

$$P[c_{j,t}] = \frac{\varphi(f(c_{j,t}))}{\sum_{k=1}^{m} \varphi(f(c_{k,t}))} \tag{2.8}$$

Tournament selection is another selection method that is widely used. This method is based upon the selection of the fittest individuals based on a tournament among a randomly selected group of individuals. The evaluation of two competing individuals takes place by choosing a random number r between 0 and 1. If r is less than a predefined value T then the individual with the higher fitness is chosen to be a parent. Otherwise, the other individual is chosen [97]. Depending on the type of tournament selection being utilized, the selected individual may or may not be placed back into the population for future re-selection.

Another selection method that exhibits extremely fast convergence behavior is *deterministic selection*. In this method, only individuals with the best fitness survive an evolutionary round. Usually, the selection is done by selecting a specific number of top-most individuals after sorting the population according to the fitness values. However, this type of selection may produce poor long term results as low performers are entirely removed from the population, while they could exhibit certain attributes that could produce high future performance.

2.3.3 Crossover Operator

As part of the evolutionary process, genetic operators are utilized to apply changes to the genetic encoding of an individual. The crossover operator exchanges genetic material between two parent individuals producing hybrid offspring. The application of the crossover operation on individuals plays a central role in genetic evolution and could be considered one of the main characteristics of the algorithm. The crossover points are chosen randomly determining the section of genetic code to be transferred. Several crossover methods may be utilized, each using a different formula for determining the nature of how chromosomes are transferred between individuals.

- One-point crossover utilizes only a single random splitting point for the chromosomes of the individuals, then the two tails to the right or to the left of the crossover line are swapped.

- In two-point crossover, two crossover points are randomly selected, and the genes that reside between the two lines are swapped between the individuals.

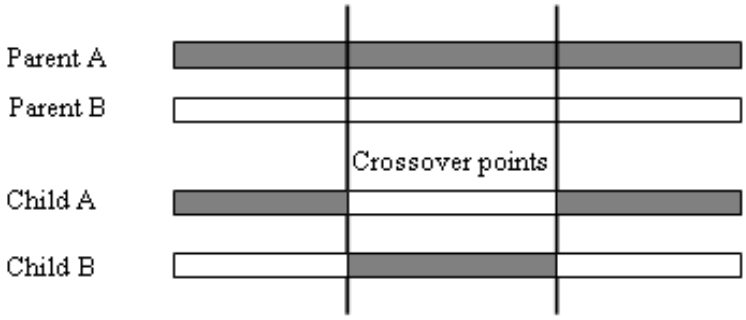


Figure 2.5: Two-point genetic crossover operator. The genes residing between the two crossover points are swapped between the two individuals.

- N-point crossover utilizes N breaking crossover lines where every second section is swapped. A variation of this method is the Shuffle crossover where a random permutation is applied to the parents before the N-point crossover is carried out. Once the crossover has been performed, an inverse permutation is performed on the children.

2.3.4 Mutation Operator

The mutation genetic operator is a process where changes are made to an individual's genes relying on a predefined probability. The process is analogous to biological mutation as it maintains genetic diversity from one generation to the next. For each of the individual's genes, the predefined probability p_m is used to determine if the gene is to be altered or left unchanged. The role of the mutation operator is to allow for exploratory moves within the search space preventing any specific point from becoming out of reach. It also helps prevent the convergence of the evolutionary process to a suboptimal solution. However, the value of p_m must be small and chosen carefully so as not to result in the chaotic changing of the genetic structure causing the process to become more like a random search.

Given n genes, the gene g_i is mutated with the probability p_m. Usually, a random number r is generated between 0 and 1, and the mutation takes place if $r < p_m$. Similar to the crossover operator, several mutation methods may be utilized [17]:

- **Single-bit inversion:** A single randomly chosen bit is negated with probability p_m.

- **Bitwise inversion:** Each bit in the genetic string is inverted with probability p_m
- **Random selection:** With probability p_m the entire string is replaced by a randomly generated string.

2.3.5 Core Components

Based on the principles discussed, we identify several components as core elements of the genetic algorithm. These core components must be used in unison in order to produce the evolutionary results desired. Different variations of each component exist; however, the principles governing their usage are standard, and experimentation may be used to determine the best variation for a specific problem at hand. The following are the core components of the genetic algorithm:

- **Generation of initial population:** A random initialization process may be used, however, in robotic control problems, special constraints may be placed on the initialization process so as not to produce individuals whose behavior may be harmful to themselves or the environment.
- **Evaluation of individual performance:** A fitness function is used to evaluate the performance of each member of the population. The results are stored and used to determine the probabilities of individual selection.
- **Individual selection for reproduction:** Based on each member's performance in relation to the fitness function, one of the selection methods (roulette wheel, tournament or deterministic) is used to choose the set of individuals to proceed to the next generation. The higher an individual's performance, the higher the probability this individual will be selected.
- **Generation of offspring through crossover:** The next generation of offspring are generated by choosing and applying one of the crossover methods to the parent population. This would involve the swapping of genes between parents producing the offspring.
- **Mutation of selected offspring:** Individual genes are mutated using a predefined probability p_m. The mutation method utilized is chosen depending on the problem at hand.

- **Repeat until terminating condition is met:** Any of the following conditions may be chosen to terminate the evolutionary process:
  - A target generation number is reached,
  - A specific average fitness is reached, or
  - A specific maximum fitness is reached.

The following algorithm describes the evolutionary process:

Procedure Genetic Algorithm begin(1)
t := 0;
initialize G_t;
evaluate G_t;
While *Not termination-condition* **do**
begin(2)
$t = t + 1$;
select G_t *from* G_{t-1};
crossover G_t
mutate G_t
evaluate G_t
end(2)
end(1)

2.4 Genetic Encoding

In order to successfully carry out the genetic process, means are needed for encoding the different attributes of the agent being evolved. Two main encoding schemes are mostly used: Binary-Coded Genetic Algorithms (BCGA) and Real-Code Genetic Algorithms (RCGA). The following sections discuss the main characteristics of both encoding schemes.

2.4.1 Binary Coding (BCGA)

Binary coding utilizes a string of binary bits of length *n* to represent each chromosome in the population. The following case study demonstrates the usage of BCGA as well as the application of the different genetic operators on a binary coded structure.

Our study will utilize a roulette wheel selection method along with two-point crossover without mutation. We consider the simple problem of finding the maximum of a polynomial function[17]. We define the polynomial function f as

$$\begin{array}{ccc} f_1 : \{0,....,63\} & \rightarrow & \mathbb{R} \\ x & \mapsto & 3x^2+2x+1 \end{array}$$

We choose a binary string $C = \{0,1\}^6$ where a value from $\{0,....,63\}$ is used to encode the chromosome of each individual within the population. Each individual will be represented by a bit sequence to indicate a value corresponding to x. The fitness function for each individual is then calculated by evaluating the function $3x^2+2x+1$. Given the number of bits n to be encoded, we choose an initial population G of size s to be initialized such that

$$\forall g_i \in G, g_{(i,k)} = Random[0,1], i \in \{1,...,s\}, k \in \{1,...,n\}$$

The initial population is chosen of size 10 yielding the random distribution shown in Table 2.1. The last column shows the probability of choosing the individual for reproduction based on the roulette wheel selection method.

Individual	Chromosome *genotype*	x value *phenotype*	f(x) *fitness*	p_i *selection*
1	1 1 1 0 0 1	57	9,862	0.17
2	0 1 1 1 0 0	28	2,409	0.04
3	1 1 0 1 1 0	54	8,857	0.16
4	1 0 1 1 0 1	45	6,166	0.11
5	0 0 1 1 0 0	12	457	0.01
6	1 1 1 1 1 0	62	11,657	0.21
7	1 1 0 1 0 1	53	8,534	0.15
8	1 0 1 0 0 1	41	5,126	0.09
9	0 0 0 0 0 1	1	6	0.00
10	1 0 0 0 0 1	33	3,334	0.06

Table 2.1: Initial random distribution of genetic code. The roulette wheel selection method is used to produce the reproduction probability p_i shown in the last column.

In order to evaluate the fitness of each individual, the encoded chromosomes must be decoded to produce a performance value. In this particular scenario, where a bit-string is used, each chromosome is decoded by evaluating the decimal equivalent of the binary value stored. Given the encoded string $s = \{0,1\}^n$, the chromosome c_i is decoded as

$$c_i = \sum_{k=0}^{n-1} s[n-k] \cdot 2^k$$

The computed results exhibit an average fitness of 5,640, while the maximum fitness achieved is 11,657. The selection probability is computed based on the formula

$$p_i = \frac{f_i}{\sum_{k=1}^{m} f_k}$$

For example, individual number 6 scored the highest on the fitness evaluation with a score of 11,657 yielding the highest selection probability of 21%. On the other hand, individual number 9 scored the lowest yielding a probability very close to zero for reproduction. The fitness statistics of the initial population is shown in Table 2.2.

Total Fitness	Average Fitness	Max Fitness
56,408	5,640	11,657

Table 2.2: Fitness statistics evaluating the performance of the initial random population.

The selection operator is then applied based on the reproduction probability of each individual. The results of the application of genetic selection is shown in Table 2.3, while the associated fitness statistics are shown in Table 2.4.

Individual	Chromosome *genotype*	*x* value *phenotype*	f(x) *fitness*
1	1 0 1 1 0 1	45	6,166
2	1 0 1 1 0 1	45	6,166
3	1 1 0 1 1 1	55	9,186
4	1 1 0 1 0 0	52	8,217
5	1 1 1 1 0 0	60	10,921
6	0 1 1 1 1 0	30	2,761
7	0 1 1 1 0 0	28	2,409
8	1 1 1 1 1 0	62	11,657
9	1 0 1 0 0 1	41	5,126
10	1 1 1 1 0 1	61	11,286

Table 2.3: Second generation of individuals after applying the selection operator.

The overall fitness of the second generation is clearly higher than that of the first. The probabilistic selection of the best individuals of the first generation produced an elevation in the average fitness achieved by the population. The two-point crossover genetic operator is then applied to the second generation of individuals. The method relies on

Total Fitness	Average Fitness	Max Fitness
73,883	7,388	11,657

Table 2.4: Fitness statistics evaluating the performance of the second generation produced by the selection operator.

the random selection of two crossover point for the transfer of genetic material between two individuals, as shown in Figure 2.5. In the bit-string representation, the bits residing between the two crossover points are swapped. Table 2.5 demonstrates the application of the two-point crossover method. The second column shows the individuals pre-crossover, while the fifth column shows the individuals after the crossover has been performed based on the two random points chosen.

Individual	pre-crossover	Point 1	Point 2	post-crossover
1	1 0 1 1 0 1	2	6	1 0 1 1 0 1
2	1 0 1 1 0 1	2	6	1 0 1 1 0 1
3	1 1 0 1 0 1	3	5	1 1 0 1 1 1
4	1 1 0 1 1 0	3	5	1 1 0 1 0 0
5	1 1 1 1 1 0	2	6	1 1 1 1 0 0
6	0 1 1 1 0 0	2	6	0 1 1 1 1 0
7	0 1 1 1 0 0	4	4	0 1 1 1 0 0
8	1 1 1 1 1 0	4	4	1 1 1 1 1 0
9	1 1 1 0 0 1	1	3	1 0 1 0 0 1
10	1 0 1 1 0 1	1	3	1 1 1 1 0 1

Table 2.5: Application of the two-point crossover operator to the second generation of individuals.

After 20 generations of selection and crossover, we can see the average fitness of each generation increase gradually over the previous as shown in Figure 2.6. For this simple problem, the optimal average fitness is reached by the 11^{th} generation, which demonstrates a relatively rapid convergence. However, other more complex problems may require hundreds or thousands of generations for the results to converge.

2.4.2 Discretized Search

When dealing with discrete values for x, the chromosome binary encoding is direct. However, when dealing with a range of continuous values, discretization of the search space is

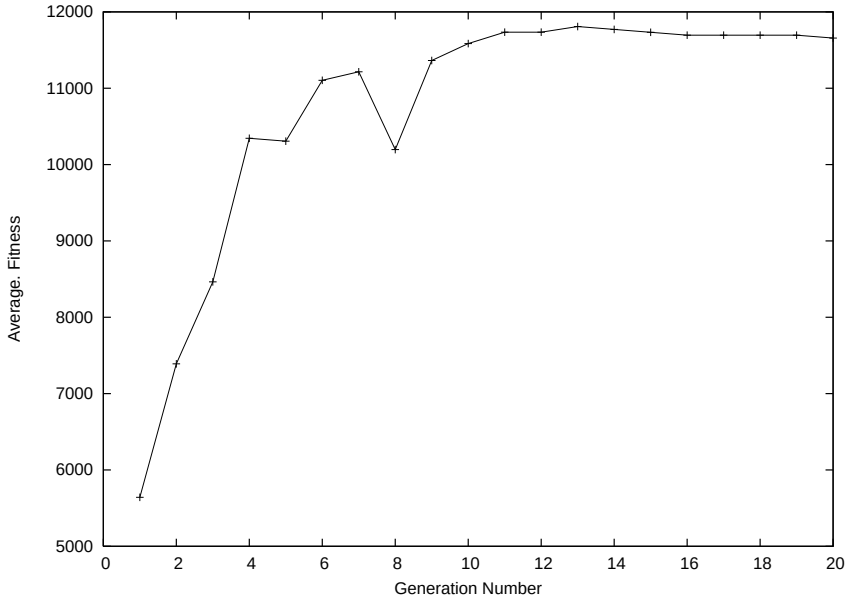


Figure 2.6: Evolution over 20 generations of individuals.

needed. One technique for achieving discrete values for the encoding of agent attributes is to divide the search space into 2^n intervals and represent each interval by a point that can be enumerated. This strategy would yield 2^n points to be encoded using a binary string. In the general form, given the interval $[a,b]$, the encoding function is described as [17]

$$\begin{array}{rcl} c_{n,[a,b]} : [a,b] & \rightarrow & \{0,1\}^n \\ x & \mapsto & b_n(rnd((2^n-1)\cdot\frac{x-a}{b-a})) \end{array}$$

where b_n is a function which converts a number from $\{0,\ldots,2^{n-1}\}$ to its binary representation. The decoding function can be defined as

$$\begin{array}{rcl} \tilde{c}_{n,[a,b]} : \{0,1\}^n & \rightarrow & [a,b] \\ s & \mapsto & a + bin_n^{-1}(s)\cdot\frac{b-a}{2^n-1} \end{array}$$

Lets consider the problem of finding the maximum of the function:

$$\begin{array}{rcl} f_2 : [0,15] & \rightarrow & \mathbb{R} \\ x & \mapsto & \sqrt{x}\cdot\cos(x) \end{array}$$

The plot for the function is shown in Figure 2.7. We will choose $n = 16$ for the discretization of the search space yielding a solution accuracy of $1.14E^{-4}$. We will now apply the evolutionary algorithm to a population of 100 individuals using the roulette wheel selection method, two-point crossover and random mutation with a probability of 0.001.

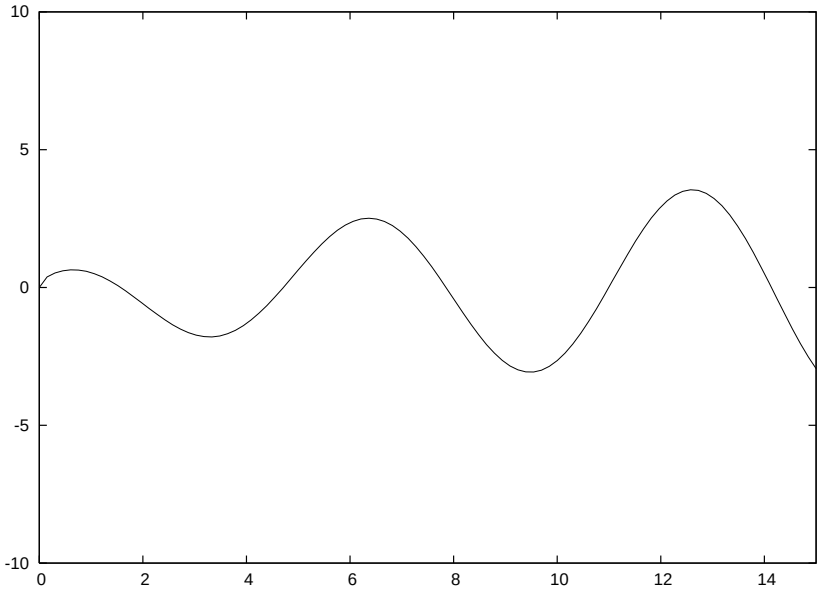


Figure 2.7: Plot of function x = $\sqrt{x} \cdot \cos(x)$

The results of the evolutionary process are shown in Figure 2.8. An optimal approximate solution was reached by the tenth generation, For this experiment, the results show how quickly an evolutionary algorithm can reach an approximate solution for a particular problem compared to an exhaustive search which scans the entire search space. An exhaustive search would require $2^{16} = 65,536$ evaluations, while the optimal solution was reached using $10 \times 100 = 1000$ evaluations.

2.4.3 Schema Theorem

The schema theorem was formulated by Holland [60] in 1975, and it provides theoretical expectations of a GA over the evolutionary process. The theorem represents the first attempt to explain why GAs work, as it describes the propagation of schemata from one generation to the next under the influence of selection, crossover and mutation. Some criticism does exist over the schema theorem; however, Holland's work does effectively describe the way searches take place using GAs.

A schema describes a pattern present among a subset of chromosomes. For example, the schema $\varepsilon = 1*1*00$ represents the chromosomes:

$$\{(101000), (101100), (111000), (111100)\}$$

Two features of ε are described as follows [57]:

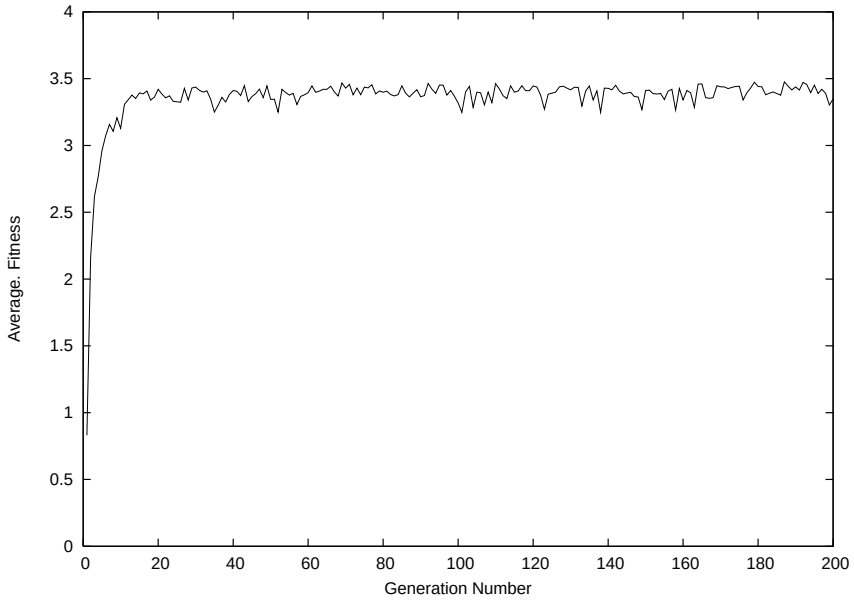


Figure 2.8: Discretized search evolution results. Optimal solution is reached within the first 20 generations of evolution.

- The order of ε, denoted by $o(\varepsilon)$, represents the number of fixed symbols present in ε.
- The defining length of ε, denoted by $\delta(\varepsilon)$, represents the difference between the first and the last fixed symbol in ε.

The informal statement of the Schema Theorem is that *short, low-order schema with high average fitness will increase in number in the following generation*. We now consider a binary-coded chromosome of length L. The function $f(\varepsilon)$ represents the average fitness of the instances of ε in the population, while $\bar{f}$ denotes the average fitness of all individuals in the population. The number of instances of ε in the population at generation t is defined as $m(\varepsilon,t)$. After the application of the selection, crossover and mutation operators, the expected number of instances of ε in generation $t+1$ is given by [60]

$$m(\varepsilon,t+1) \geq m(\varepsilon,t) \cdot \frac{f(\varepsilon)}{f} \cdot (1 - p_c \cdot \frac{\delta(\varepsilon)}{L-1}) \cdot (1-p_m)^{o(\varepsilon)} \tag{2.9}$$

After applying the selection operator, the expected number of ε present is $m(\varepsilon,t) \cdot \frac{f(\varepsilon)}{f}$. The probability of ε being present after applying the crossover operator is approximated by$(1 - p_c \cdot \frac{\delta(\varepsilon)}{L-1})$. We notice that the probability is inversely related to $\delta(\varepsilon)$. The probability

of ε being present after mutation is approximated by $(1-p_m)^{o(\varepsilon)}$ and it is inversely related to $o(\varepsilon)$.

The condition for a schema to increase its fitness in the next generation is given by

$$\frac{f(\varepsilon)}{f} > (1 - p_c \cdot \frac{\delta(\varepsilon)}{L-1}) \cdot (1-p_m)^{o(\varepsilon)}$$

The *Building Block Hypothesis* [47] is closely related to the Schema Theorem. It describes the behavior of GAs in an effort to discover and exploit collections of closely interacting genes. The collections are further combined to create successively larger blocks that eventually solve the problem [34].

The main criticism of the Schema Theorem is in the fact that the theorem does not take into consideration the effects of crossover and mutation on the evolving populations. Such effects change the structure of the chromosome as well as the successive effects of the genetic operators. A more sophisticated presentation of the theorem will have to take into account the affects of mutation as it allows for the creation of a child whose schema belongs to none of the parents, also called schema creation. Schema disruption is also an important phenomenon which must be considered. Disruption occurs when the schema of the child differs from that of its parents.

2.4.4 Arguments for BCGA

Two main arguments exist for using binary-coded genetic algorithms. The first argument is that the use of the binary alphabet maximizes the implicit parallelism in the evolutionary process. A binary-coded genetic algorithm processes a very large amount of information in parallel, and that is partly due to the nature of the binary alphabet where each part of the chromosome is a separate entity.

For a given information content, strings coded with smaller alphabets are representatives of larger numbers of similarity subsets (schemata) than strings coded with larger alphabets [57].

The second argument relates to the number of fitness evaluations feasible in relation to the problem being solved. This problem may be managed through the choice of smaller population sizes as well as smaller number of genes within each chromosome. This would reduce the computational expense of the evolutionary process.

The binary alphabet offers the maximum number of schemata per bit of information [48].

Despite the advantages of using BCGA, some drawbacks do exist due to the fact that a large portion of optimization problems utilize real-valued parameters. The first disadvantage is that the interval for value discretization must be specified in advance. Classical BCGA methods do not allow for an unbounded search of the solution space, and a very large interval would require a massive number of partitions to cover it, or the precision of the results will have to be sacrificed. In addition, the accuracy of the solution produced is limited by the width of the discretization interval width given by

$$\frac{1}{2^n - 1}$$

Due to some of the limitations of BCGA, other coding schemes have been developed to deal with specific types of problem parameters. The next section discusses *Real-coded Genetic Algorithms (RCGA)*, which was developed specifically to deal with real-valued parameters in a more practical manner.

2.4.5 Real Coding (RCGA)

Coding the chromosomes of individuals as real numbers allows for the direct representation of problem parameters in the genetic code. An N-dimensional vector of floating point numbers may then be used to represent each individual in the population. The size of the chromosome vector will be the same as the size of the vector which represents a solution to the problem, so each gene in the chromosome represents a variable of the problem [57]. The use of real-coded genetic algorithms (RCGA) offers many advantages over the use of BCGA.

- Real coding allows for encoding the different chromosomes (genotype) without the need for any translation of the problem parameters. The genotype and phenotype become the same. This allows for a much simpler genetic representation of the problem.
- Encoding parameters as floating point numbers allows for the exploration of very large domains without loss of precision.
- RCGA allows for the utilization of *graduality* in order to achieve the desired solution. With BCGA, changing a single gene can cause a drastic change in the fitness value of the individual. However, RCGA allows for the gradual changing of chromosome values in an effort to achieve gradual enhancement in the fitness value.

The selection operator discussed for BCGA can be used with RCGA without the need to make any modifications. The selection process is identical as it is based on the fitness values of the individuals regardless of the method of encoding being utilized. The crossover and mutation operators, however, will need to undergo some modifications as shown in the following sections.

2.4.6 Crossover Operator for RCGA

The crossover operator for RCGA carries the same principles as that of BCGA. The main purpose of the operator is to swap genetic material between two individuals creating offspring that share the characteristics of the parents. The following are the most common crossover operator types used for RCGA:

- **Simple crossover:** this crossover type is identical to the one-point crossover for BCGA. Instead of swapping bits between the two individuals, floating point elements are swapped. Given the two individuals $C_1 = (c_1^1, c_2^1, ..., c_n^1)$ and $C_2 = (c_1^2, c_2^2, ..., c_n^2)$, the crossover location $k \in \{1, 2, ..., n-1\}$ is chosen at random, then the two offspring b_1 and b_2 are structured as follows:

$$b_1 = (c_{1,1}, c_{2,1}, ..., c_{k,1}, c_{k+1,2}, ..., c_{n,2})$$
$$b_2 = (c_{1,2}, c_{2,2}, ..., c_{k,2}, c_{k+1,1}, ..., c_{n,1})$$

- **Flat crossover:** Given the individual $C_i = (c_1^i, c_2^i, ..., c_n^i)$, the offspring $b_i = (x_1, x_2, ..., x_n)$ is created using the vector of random values $(r_1, r_2, ..., r_n)$ where

$$x_i = r_i \cdot c_i^1 + (1 - r_i) \cdot c_i^2$$

- **BLX-α crossover:** This method is an expansion of the flat cross over method. In order to allow values outside of the interval $[min(x_i^1, x_i^2), max(x_i^1, x_i^2)]$ to be included in the offspring generation, this method expands the range by the percentage α. Each element of the offspring chromosome vector is chosen as a random value from the interval [17]

$$[min(x_i^1, x_i^2) - I \cdot \alpha, max(x_i^1, x_i^2) + I \cdot \alpha]$$

where

$$I = max(x_i^1, x_i^2) - min(x_i^1, x_i^2)$$

and the parameter α has to be chosen in advance to control the amount of expansion taking place.

2.4.7 Mutation Operator for RCGA

The mutation operators for RCGA operate on individual chromosomes changing their genetic structure. Given the chromosome $C = (c_1, ..., c_i, ..., c_n)$, any of the following mutation method may be applied to change C [94].

- **Random mutation:** each gene c_i is replaced by a random value generated from the predefined interval $[a_i, b_i]$.
- **Non-uniform mutation:** this method allows for the impact of the mutation to be less significant as the number of generations increase. Let g_{max} be the maximum number of generations to be evolved, and let g be the current generation number. The gene c_i is then calculated using one of the following two values (selected at random with equal probability)

$$c_i' = x_i + \Delta(t, b_i - x_i)$$
$$c_i'' = x_i - \Delta(t, x_i - a_i)$$

where

$$\Delta t(t,x) = x \cdot \left(1 - r^{\left(1 - \frac{g}{g\max}\right)^b}\right)$$

The value b is chosen by the user to determine the significance of the iteration number on the mutation result.

2.4.8 BCGA-RCGA Comparison

Figure 2.9 shows two comparative graphs for the maximization problem initially introduced in section 2.4.2. The upper graph shows the evolution results demonstrated earlier using the BCGA techniques discussed. The lower graph, however, shows the results for the RCGA implementation using chromosomes based on real-coded parameters. A population of 100 individuals was chosen, and the chromosome of each individual was coded using the x value to be optimized, thus the problem parameter was in fact the genotype to be evolved. The BLX-α crossover method was used to allow for expanding the search target area in an exploratory manner. An expansive crossover α value of 0.1 was chosen as an intermediate value to limit the deviation from any good results reached. A mutation probability $p_m = 0.005$ was used to promote stability while keeping the mutation factor still present. As shown in the figure, the results are almost identical. Such results demonstrate the effectiveness of RCGA encoding methods eliminating the need for discretizing the parameter search space.

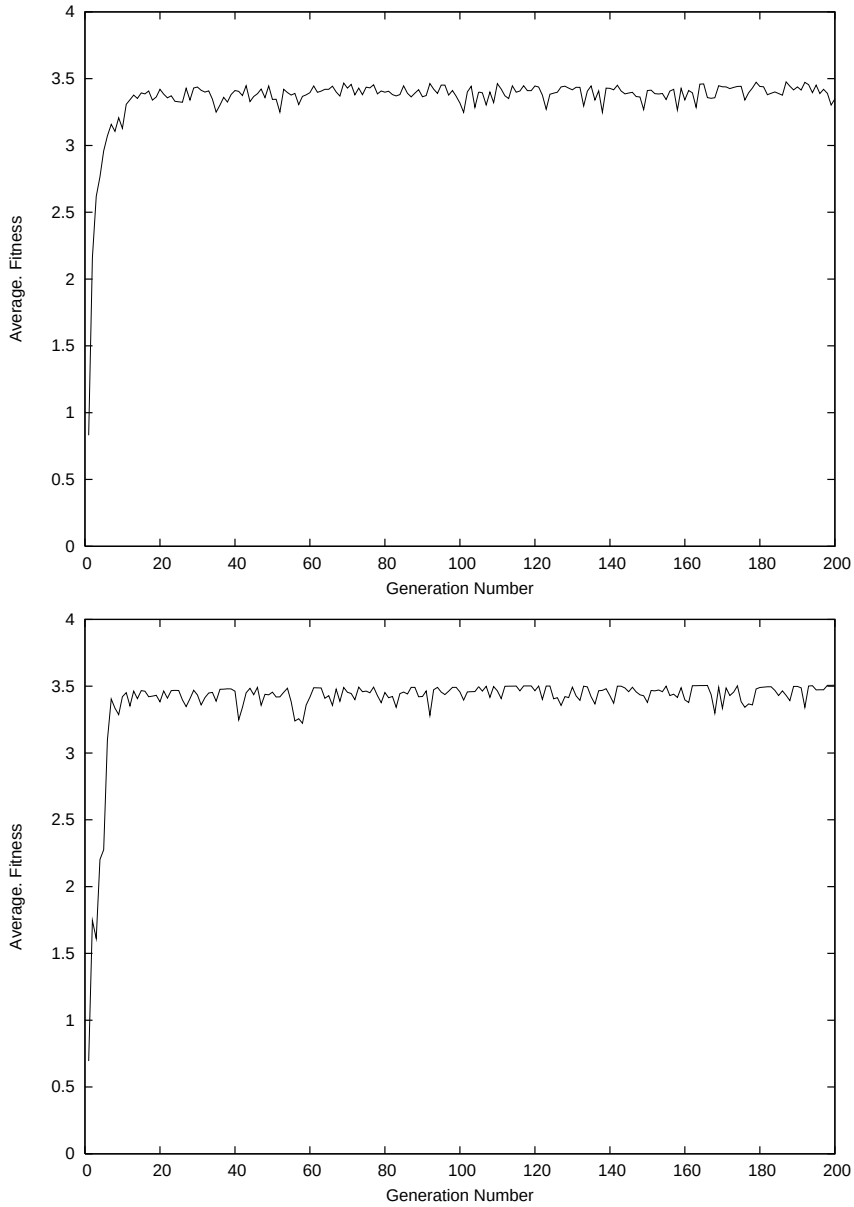


Figure 2.9: Two graphs showing the comparative performance of BCGA and RCGA. The top graph represents the BCGA solution covered in section 2.4.2, while the bottom graph shows the RCGA solution to the same problem.

2.5 Evolving a Robotic Controller

In this section, we discuss the utilization of evolutionary techniques in the creation of a robotic controller capable of making real-time control decisions. The controller we will demonstrate handles the inverted pendulum problem, which is often utilized as an example of an unstable dynamic system with multiple parameters. The evolutionary pro-

cess will allow for the controller to optimize its own performance through the knowledge gained while performing the task. This is accomplished through the evaluation of the outcome of individuals after each experiment, followed by evolving the producers of the best results.

The inverted pendulum problems deals with the task of keeping a rigid pole, which is hinged to a moveable wheeled cart, from falling to the ground. The pole is free to move about the hinge axis within the vertical plane and would fall under the force of gravity unless the cart is moved in an appropriate fashion to counter any falling potential. The cart is also constrained to a maximum distance from its initial starting point. In order to successfully handle the balancing task, the controller must be capable of applying corrective left and right forces to the cart to compensate for the pole's rotation, yet without having the cart exceed the maximum distance allowed. Figure 2.10 demonstrates the overall characteristics of the inverted pendulum environment [1].

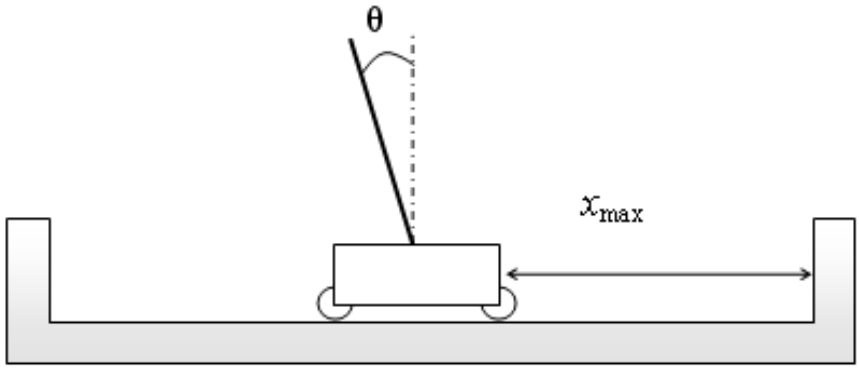


Figure 2.10: The inverted pendulum environment.

The following four parameters are available to the controller at each time step t:

- x_t: the horizontal distance of the cart along the x-axis measured from the cart's initial starting position. The value is given in meters and is constrained to a maximum value of x_{max},
- v_t: the horizontal velocity of the cart along the x-axis. The value is given in meters per second,
- θ_t: the pole's clockwise angle measure to the z-axis. The angle is given in degrees, and the maximum angle bounds allowed to maintain successful balancing is $\pm\theta_{max}$,
- ω_t: the pole's angular velocity measured in degrees per second.

For this particular system, a failure state is reached if the pole falls past the given angle ($|\theta| > \theta_{max}$), or if the cart reaches the maximum distance allowed. In formal terms, at time t, the system state s_t is described as

$$s_t = \begin{cases} 1, if\ |\theta_t| < \theta_{max} or\ |x_t| < x_{max}; \\ 0, otherwise \end{cases}$$

In order to evolve individuals capable of balancing the pole successfully, a fitness function formulation is needed to the gauge the performance of each individual. The fitness function accumulates a value $+1$ for each time step during which failure did not occur. For example, if the pole was successfully balanced for 100 time steps, then the resultant value of the fitness function is 100. Once failure occurs, the pole is reset to the vertical position and the cycle repeats for the next individual.

2.5.1 Physics-based Simulated Environment

A physics-based simulation environment based on Rigid Body Dynamics was utilized for evolving individuals to perform the inverted pendulum task successfully. The environment accurately simulates gravitational forces as well as the friction forces between the tires and the ground. Figure 2.11 shows the simulated environment being utilized.

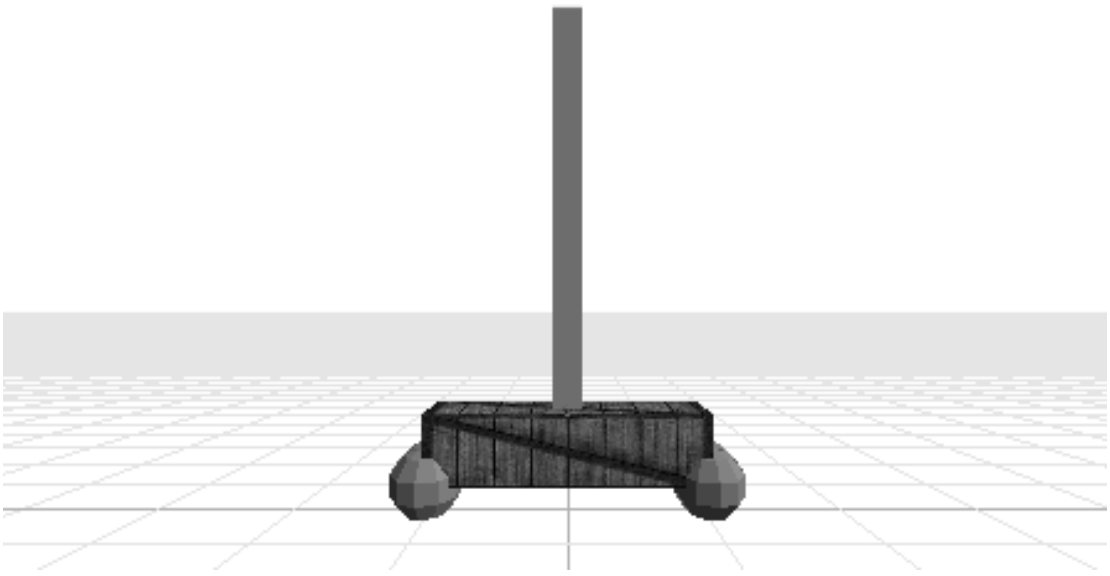


Figure 2.11: Physics-based simulation environment utilized for the application of the evolutionary algorithms.

A fixed time step of $t = 1/30$ is used as the basis for the dynamics engine stepping.

Although the physics-based simulation environment requires more processing time per time step than using direct mathematical methods for calculating the different parameters of the environment, the added elements of collision detection and response adds to the reality of the simulation as many parameters interact together to produce a single result. Such interactions increase the level of noise in the simulation environment causing the results to be more reliably fit. The training takes place without rendering the scene after each time step in order to speed up the evolutionary process. After each failure, or if the maximum time step is reached, the pole is reset back to the vertical position, and the next individual takes control of it. Once the population has evolved, then the best individual is chosen for a rendered run to visually show the results of the evolution.

2.5.2 Fixed-magnitude Force Application

The original inverted pendulum experiment utilizes corrective forces of a predefined fixed magnitude in order to compensate for the motion of the pole; a force of zero magnitude is not permitted at any time step. The inverted pendulum problem definition follows the principles presented by Barto, Sutton, and Anderson [13] in that the search space is divided up into partitions (boxes) which produce specific target ranges for each of the problem parameters.

Each of the four input parameters is partitioned into multiple sectors of interest. This approach yields a discrete number of states for which the controller could adapt its signals. The symmetry between the positive and negative parameters was not taken into account in order to maintain the realism of the control environment. The four parameters are partitioned according to the ranges shown in Figure 2.12. This method yields 162 distinct states to which the controller must adapt its decision making. As this experiment will utilize forces of a fixed magnitude, and zero force application is not allowed, the resultant control signal will either be a command for left-force application or right-force application. Hence, a binary value would be sufficient to describe the control signal needed to compensate for the motion of the pole. For each of the 162 states, a single binary value can yield the application of the appropriate force.

An RCGA encoding scheme is used to structure the chromosomes of the individuals to be evolved. Each chromosome will contain 162 genes (floating point numbers) each representing a particular state of the problem. The RCGA encoding was chosen instead of BCGA to allow for a gradual learning curve that has a better chance of approaching better results. The individuals will be evolved so that the command g_i associated with each state ($g_i < 0.5$ for left and $g_i >= 0.5$ for right) would be optimized from one generation to the

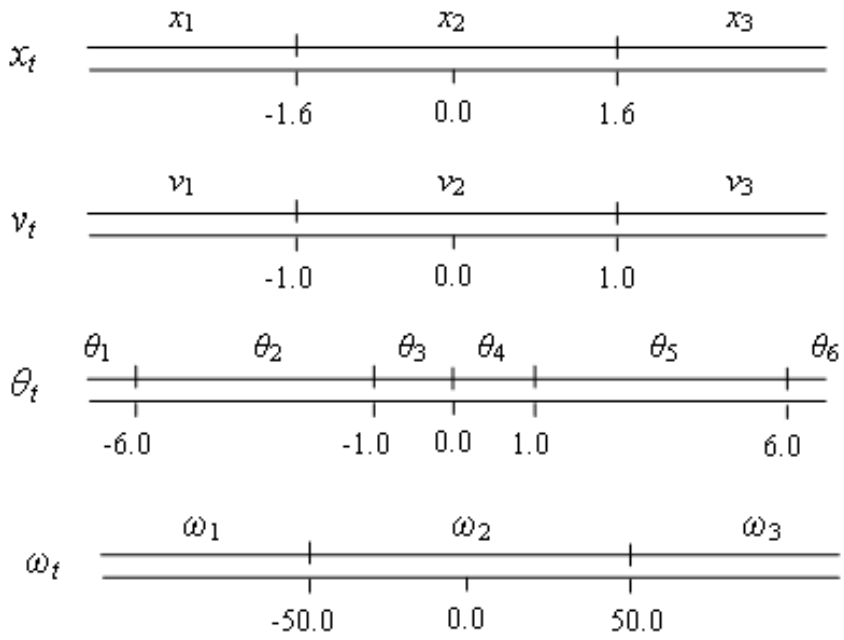


Figure 2.12: Search space partitioning for the inverted pendulum problem. The partitioning yields 162 distinct states.

next. The following parameters were used in the evolutionary process:

- Number of individuals: 50
- Number of generations: 100
- Crossover expansion range: 0.1
- Mutation probability p_m: 0.01

2.5.3 Results

The evolution results (Figure 2.13) show a steady increase in performance that took place over the first 40 generations. The last 60 generations produced fitness results that were confined between 2,300 and 2,800 showing a slowing down in the learning process. Such a halt in learning is attributed to the complexity of the problem as well as the delayed reward or penalty associated with each decision. A wrong control decision would cause a failure to occur several hundred steps later or more making it difficult to trace back the source of failure. Other methods, like reinforcement learning for example, which is beyond the scope of this discussion, allow for such backward propagation of reward or

penalty to further enhance the learning curve. However, from an optimization point of view, the evolutionary process is still the most effective tool for searching the problem space and yielding significant results using a small number of iterations.

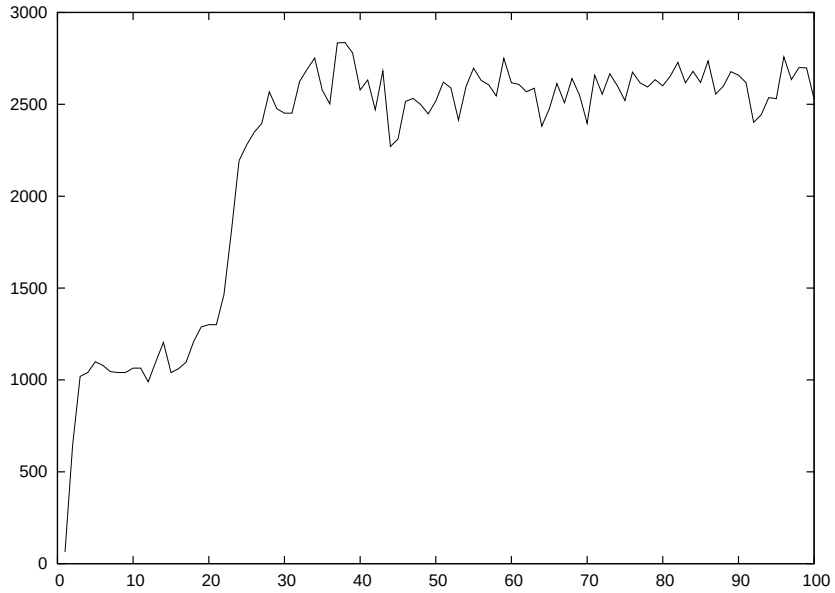


Figure 2.13: Evolution results for the inverted pendulum problem utilizing physics-based rigid body dynamics engine. The graph shows the average fitness over 100 generations of evolution; each generation included 50 individuals.

The experiment was repeated using the code written by Sutton [127] which utilizes Euler's method for applying the equations of motion. The only difference between the environment used in the first experiment and the second is in the utilization of rigid body dynamics. The first experiment took into account the frictional and interactive forces between the environment bodies producing an approximation that includes a level of noise which better resembles real world results. However, the second experiment used only numerical approximations applying the equations of motion based on the action decided upon by the controller.

As seen from the results shown in Figure 2.14, the overall fitness reached significantly higher values than in the previous experiment. Over the first 90 generations, the results were similar to the results based on rigid body dynamics, however, a drastic improvement took place over the last 10 generations. Such significant differences in results show the importance in building simulations that better resembles real world environments. A simulation based solely on numerical methods without regard to the complex interac-

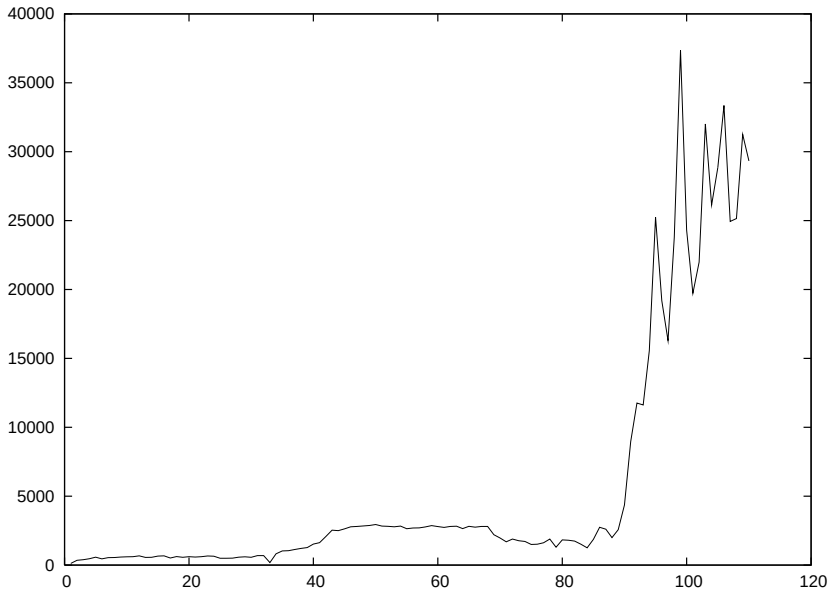


Figure 2.14: Evolution results for the inverted pendulum problem utilizing only numerical approximations of the equations of motion.

tions between bodies might show great results only in simulation, but those results do not translate well into real robots operating in a more complex environment which includes higher levels of noise. Section 2.2 offers a more detailed discussion of simulation-based evolution.

2.6 Conclusion

Evolutionary robotics utilizing genetic algorithms offer extremely powerful methods for rapid solution approximation, even when the scope of the problem covers a very large search space. Evolutionary robotics has the following advantages:

- A thorough understanding of the dynamics of the problem is not needed as the evolutionary process allows for the discovery of the best solution to the problem using a self-organization mechanism.
- The fine tuning of evolutionary parameters is possible allowing for a more refined search when needed.
- The evolutionary process is very fast in finding a solution, scanning very large search domains over only a few generations.

- The genetic algorithms are generally easy to design and implement. Usually the genetic chromosomes map directly to the problem parameters.

The following disadvantages are also associated with evolutionary techniques:

- Problems requiring a very large number of parameters might converge to suboptimal solutions.
- Choosing unsuitable parameters for the genetic algorithms can possibly yield inaccurate evolutionary results or longer convergence time.

Despite the disadvantages mentioned, evolutionary robotics provide many possibilities for the creation of complex controllers that evolve based on their own relative performance. Although beyond the scope of this discussion, the evolutionary process may also be combined with other neuro-based methods to provide the flexibility of neural networks as well as the fast optimization possibilities of genetic algorithms.

Chapter 3

GENETIC PROGRAMMING

3.1 Introduction

The automation of algorithm design for solving complex problems is a very intriguing field of study. Such automation could decrease the time required for achieving a solution as well as allow for the automated optimization of existing solutions [30]. However, the formulation of an automated platform for the efficient construction of algorithmic solutions to problems can be a very complex task. Genetic Programming (GP) [70, 71, 72] is a methodology based on genetic evolution and used to evolve algorithmic solutions in the form of computer programs. GP represents a natural evolution from Genetic Algorithms (GA) [60] as it aims to reduce human intervention [131] in solution finding. Instead of evolving chromosomes that represent a point solutions to a problem as in GA, GP evolves a more flexible structure representing a program. In essence, GP produces a method for solving a problem rather than just a point solution [30], and that is one of the main benefits of genetic programming. GP has been successfully utilized for developing solutions in many areas, including data mining and classification [42], symbolic regression [121], system modeling [80] and robotic control [131]. GP has also produced a significant amount of human-competitive results, as documented by Koza [74].

3.2 Components

GP uses many of the same methodologies utilized in GA evolution. The main distinction lies in the nature of the entities being evolved. GP evolves program components structured in a tree-based fashion through the application of genetic operators very similar to the ones used in GA. In GA, the individuals are usually composed of different parameters that represent a solution to a particular problem. GP individuals are composed of different primitive components that form a program or algorithm for solving a particular problem. Those primitive components are usually connected via a tree-structure that represents the program to be evaluated using the associated fitness function. GP operators then perform on the trees within the population by changing their structure in order to achieve better overall fitness. The following main components are utilized in the GP framework:

- *Functions* represent core components of the genetic programming approach. The essence of the process is to combine functions with different problem-specific parameters in order to create a solution to a problem. Functions may represent primitive tasks, like addition, subtraction and multiplication for example, or they may represent problem-specific complex tasks that could be treated as single entities. If a task may not be represented as a single entity, it may be partitioned into smaller tasks for inclusion in the evolutionary process.

- *Terminals* are combined with functions in order to represent the alphabet of the programs being evolved. Terminals are usually composed of variables and constants to be treated as parameters to the various functions defined. By combining terminals and functions using a tree-based structure, a program is created that could be evaluated in relation to its ability to solve the problem being investigated.

- The *fitness function* is a very important component that simply carries the essence of genetic programming. Once a program has been created through combining the different system primitives, the performance of the program has to be evaluated, and this is achieved through the use of the fitness function. The appropriateness of the fitness function is key to the successful and optimal convergence of the genetic process. Hence, the function has to be chosen carefully to act as an appropriate measure of performance. The output of the fitness function may represent a numeric value resulting from the execution of the genetic program, it may represent an error value that measures the difference between the program output and a pre-defined desired output, or it may take any other form to be determined by the designer.

The human designer must supply five main components to the GP framework in order to commence the evolutionary process. The five components needed are described as:

1. The set of terminals representing problem-specific variables, zero-argument functions, and constants.

2. The set of primitive functions to be utilized in the program solution.

3. The fitness function to be used to measure the performance of each individual.

4. Evolution-specific parameters used to control the evolutionary process.

5. The termination condition used to determine when the evolution terminates in order to present the results.

3.3 Structure

Genetic programming utilizes tree representations to define possible solutions to a problem. Programs are encoded as *syntax trees* that represent a possible solution to a problem. The function parameters define expressions involving variables, constants as well as primitive operations. The specific function $F(x+y*x, x*y)$ would be represented by the syntax tree shown in Figure 3.1. The leaf nodes of the syntax tree represent terminals containing variables and constants. The interpretation[1] of the tree starts at the leaf nodes and moves up the tree in a recursive fashion. This means that in order to evaluate any node, its children must be evaluated first producing results to be propagated up the tree until a final result is achieved.

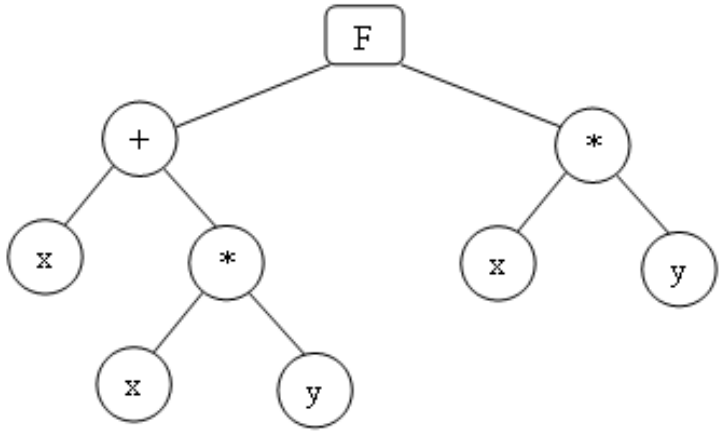


Figure 3.1: The genetic programming representation of the function F.

Similar to higher level programming, a genetic program may also be composed of multiple partitions (modules or subroutines) utilized in reaching the final solution. This is accomplished by grouping multiple sub-trees (branches) together under a single *root* node. Such representation allows for each branch of the tree to be dedicated to solving a specific task, then the different solutions are grouped together to produce a single result. This allows for the utilization of modularity in building the final algorithmic solution. In addition, it allows for the generation of more complex programs utilizing simpler functions residing at a lower level of the solution tree. A representation of a multiple sub-tree program is shown in Figure 3.2[73].

[1] Tree interpretation means traversing the tree using a specific strategy in order to achieve a result. This would be equivalent to executing the program represented by the tree.

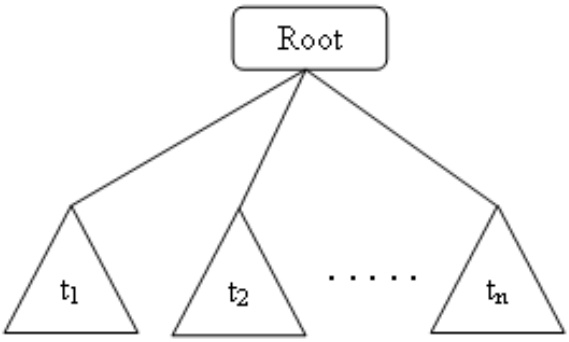


Figure 3.2: Multiple sub-tree program representation.

3.4 Genetic Operators

GP uses genetic operators very similar to those utilized in GA as means for evolving generations of individuals towards higher fitness. Once the fitness of an entire generation is evaluated[2], several genetic operators may be applied in order to produce the next generation to be evolved. The process continues until a maximum number of generations has been reached or a particular fitness value has been recorded. The termination condition may be based on the maximum fitness value achieved by a single individual, or it may be based on the average fitness value for an entire generation. Usually, once the evolutionary process has commenced, no human interaction takes place. The process operates on an automated basis until the termination condition has been met. The only exception would be if the user determines that a need exists for terminating the genetic process manually at a given point in time. The following sections identify the core operators utilized in GP.

3.4.1 Selection Operator

Reproductive selection is usually based on the performance of each individual by assigning a selection probability that is proportional to that individual's fitness score. The selection methods used for GP are very similar to their GA counterparts. The following are some of the most popular selection methods being utilized:

- The *roulette wheel* method performs selection based on the probability for reproduction of an individual. The selection probability of individual c_j is given by the

[2]The fitness of a generation is evaluated by allowing each individual an attempt at solving the problem then evaluating the performance using the problem-specific fitness function.

formula

$$P[c_{j,t}] = \frac{f(c_{j,t})}{\sum_{k=1}^{m} f(c_{k,t})}$$

where the individual's fitness at time t is defined as $f(c_{j,t})$. The size of the wheel slot $P[c_{j,t}]$ is normalized by the total fitness of m individuals in the population. The reader may refer to section 2.3.2 for a detailed discussion of selective reproduction as it relates to genetic algorithms.

- *Tournament selection* is based on selecting two individuals at random from the population and the individual with the higher fitness value proceeds to the next generation. The individuals may or may not be returned back to the general population for the next random selection depending on the method being utilized. The tournament proceeds until a sufficient number of individuals have been selected for the next generation.

- *Rank selection* utilizes the rank of individuals instead of their fitness value as the selection criteria. For example, the top ten performers may be chosen regardless of their specific fitness output.

3.4.2 Crossover Operator

Although the essence of the crossover operator is similar to that of GA, due to the significant representational difference between the two frameworks, the application of crossover in GP is different. Since GP individuals are represented as syntax trees, the crossover operates on the tree structures exchanging parts of the tree representations of individuals. The exchange takes place by selecting a random link in each tree then exchanging the associated sub-trees between the two tree representations. Figure 3.3 demonstrates the crossover operation being performed on two individuals.

One of the main advantages of genetic programming over genetic algorithms is that identical parents may produce children that are different from the parents. The tree structures allow for such crossover to take place as the operation will alter the original trees creating children that are different from the original. An example of this type of crossover is shown in Figure 3.4.

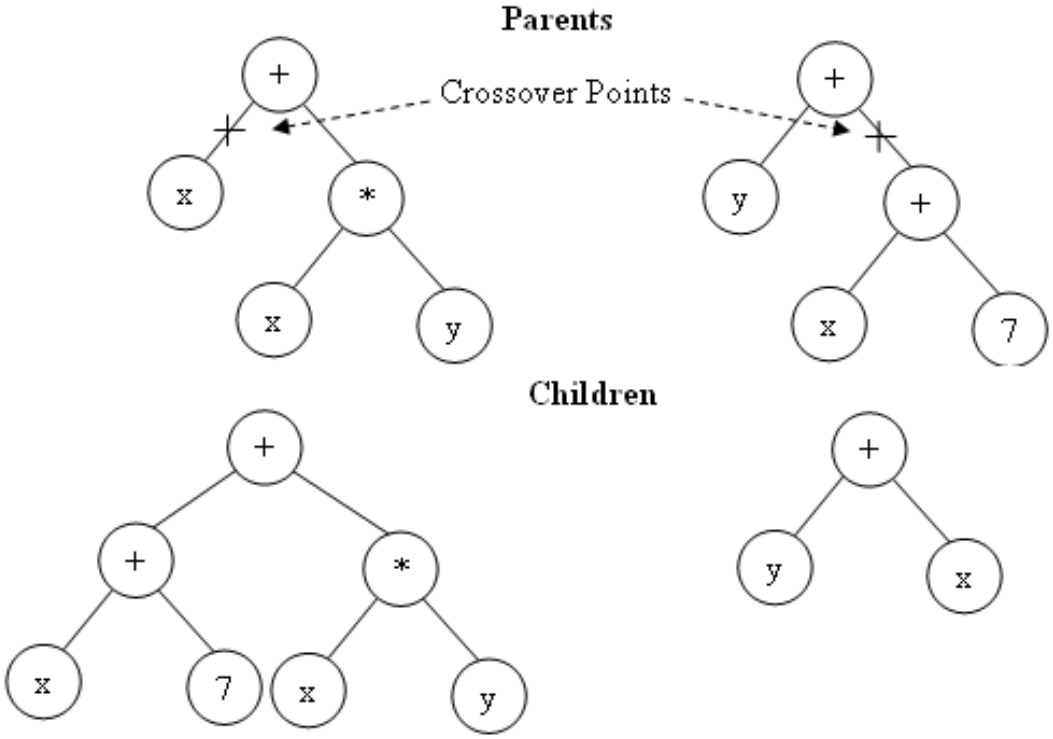


Figure 3.3: The crossover operation performed on two parent trees that are different.

3.4.3 Mutation Operator

Mutation operates on a single individual by altering the structure of its representative tree. This is usually done by selecting a random mutation link then replacing the sub-tree below the link with a randomly generated sub-tree. Genetic mutation allows for exploratory moves into the search space possibly discovering new and more efficient methods for solving the problem. Mutation may also be implemented as a regular crossover between the selected individual and a randomly generated tree. Figure 3.5 shows the random mutation operator being applied to a single individual.

3.5 Implementation

Many programming languages have been utilized to create GP frameworks. Lisp has evolved as one of the prominent languages for GP, since its recursive structure facilitates the use of tree-based organizations. Linear representations has also been utilized to build successful GP Platforms [4]. In general, GP principles are not tied to any single formal language. The GP methodologies may be used to evolve software or hardware solutions

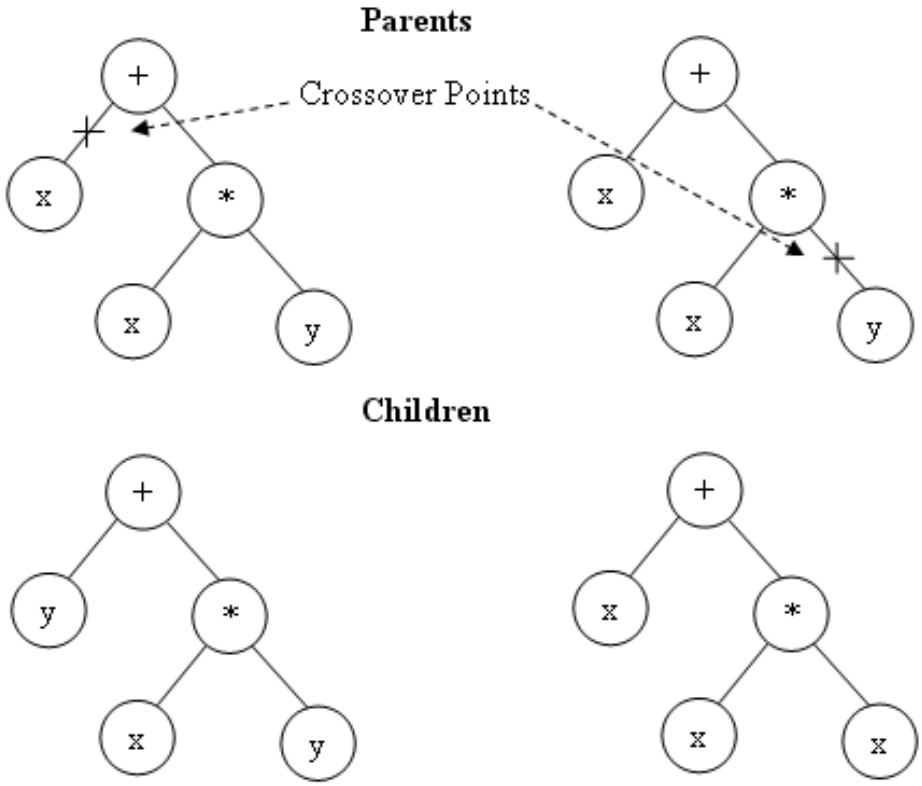


Figure 3.4: The crossover operation performed on two parent trees that are identical.

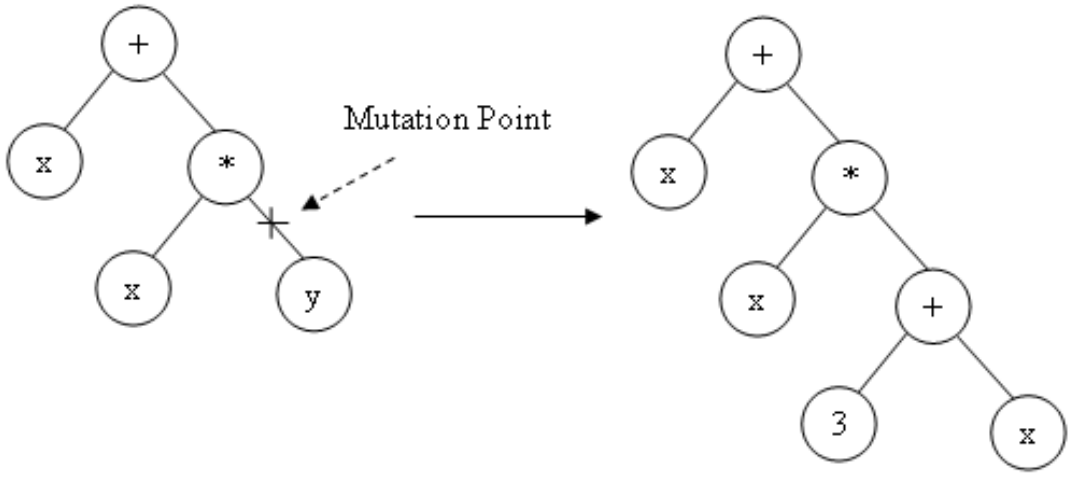


Figure 3.5: The mutation operation performed on a single individual.

using any of many existing languages and frameworks. Figure 3.6 conveys the general structure of the evolutionary process utilized in genetic programming [73].

In order to achieve and maintain the structure of the syntax trees representing the

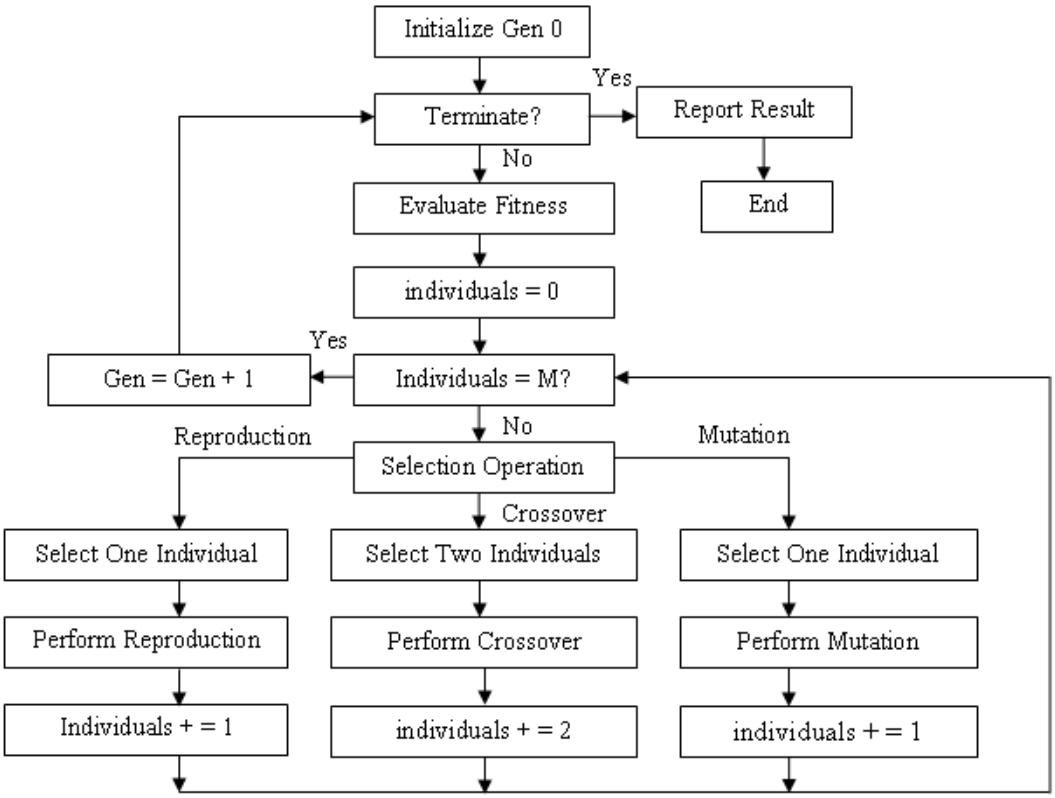


Figure 3.6: Genetic programming flowchart.

individuals being evolved, the GP platform needs to maintain several collections of objects. Those collections allow for the platform to accurately build and maintain the syntax trees. In addition, the evaluation methodologies for the system components are specified for each of the collection types allowing for the execution of evolved programs and the measuring of their performance. The following collections are maintained by the GP platform:

- *Functions:* All the primitive functions to be utilized in the building of the genetic programs must be included in this collection. The definition of each function includes the method through which the function is evaluated as well as the number of parameters that it operates on. Some functions operate on a single parameters, while others may operate on two or more. The specification of the number of parameters allows for the tree-building component of the system to construct the program-trees utilizing correct functional representations.

- *Terminal variables:* This collection includes all variables to be used as parameters

for the specified primitive functions. The variables to be utilized in the evolutionary process can be of any type; hence, the type of the variable as well as the range of values it is capable of storing must be included. The specification of the data type allows the system to properly match variables to randomly selected functions being inserted in the syntax tree.

- *Terminal constants:* Any constant value that is to be used as part of the evolved programs will be added to this collection. As with terminal variables, the data type of each constant must be specified to allow for the appropriate matching of constants to functions.

- *Nodes:* As the syntax tree is being built, the nodes collection keeps track of all entities being inserted in the tree. Each node may represent a function, a terminal variable or a terminal constant. A reference to the node occupant, a link to the parent node, as well as links to children nodes will be included.

- *Links:* Each link present within the syntax tree will be added to this collection and given a unique numeric identifier. Both the crossover and mutation operations rely on the selection of random links in order to perform tree alterations. The existence of this collection facilitates the genetic selection and alteration as each link contains a reference to the parent node which represents the root of the sub-tree to be used in the alteration process.

The first step in the evolutionary process is to create the syntax trees for each individual in the initial population. Trees are generated randomly by choosing elements from the preset function, terminal variable or terminal constant collections. If a function is chosen, the children nodes of the function node are created depending on the number of parameters needed for the function. The creation mechanism then progresses down to each child choosing random elements to satisfy its own processing needs, and so on. A height restriction may be applied to prevent the tree from growing beyond a specific height. For example, if the maximum desired height is 5, then once the tree reaches a height of 4, only a terminal may be chosen in order to halt the vertical tree growth. The tree population process continues until all leaf nodes consist of terminals.

The evaluation of each tree is performed recursively starting at the root then traversing the tree downwards resolving parameter values. For example, given the syntax tree shown in Figure 3.7, the evaluation process starts at the root node (node 1). Since node 1 contains the addition function (+), which requires two parameters, the process moves to node 2 in

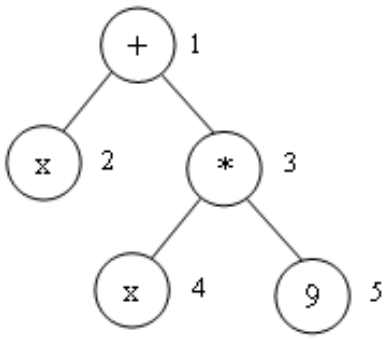


Figure 3.7: Evaluation of the syntax tree.

order to retrieve the value for the first parameter. Node 2 contains the terminal variable *x*, so no further resolution is needed for this parameter. Node 3 is evaluated next to retrieve the value for the second parameter. However, since the node contains the multiplication function (*), the process continues to nodes 4 and 5 to retrieve their values. Since nodes 4 and 5 are terminals then the process terminates. The node values are propagated upward in the tree until a final solution is reached. The evaluation process ultimately formulates the following solution to the tree:

$$x + x * 9 = 10x$$

3.5.1 Data Fitting

In this section, we provide an illustrative genetic programming process that tries to find a data fitting equation given a set of data points. For each syntax tree, the associated fitness is evaluated based on the total error that exists between the set of data points and the tree representation. The set of data points is shown in Table 3.1, and the associated graphical representation is shown in Figure 3.8. The main task for the evolutionary process is to find a syntax tree whose evaluation yields the smallest total error in relation to the original data set. The fitness function f for individual i is given by

$$f_i = \sum_{k=1}^{n} |tree_i(k) - p(k)| \tag{3.1}$$

where n is the number of data points, p(k) is the data value at position $x = k$, and $tree_i(k)$ is the evaluation of the syntax tree of individual i for the value $x = k$. The set of functions F, terminal variables V and terminal constants C are given by

x	$f(x)$
1	0.94
2	-0.98
3	-2.4
4	-1.70
5	1.234
6	3.05
7	1.39
8	-0.71
9	-2.73
10	-2.95
11	0.01
12	2.92
13	3.57
14	0.11
15	-2.44

Table 3.1: Set of data points for GP fitting.

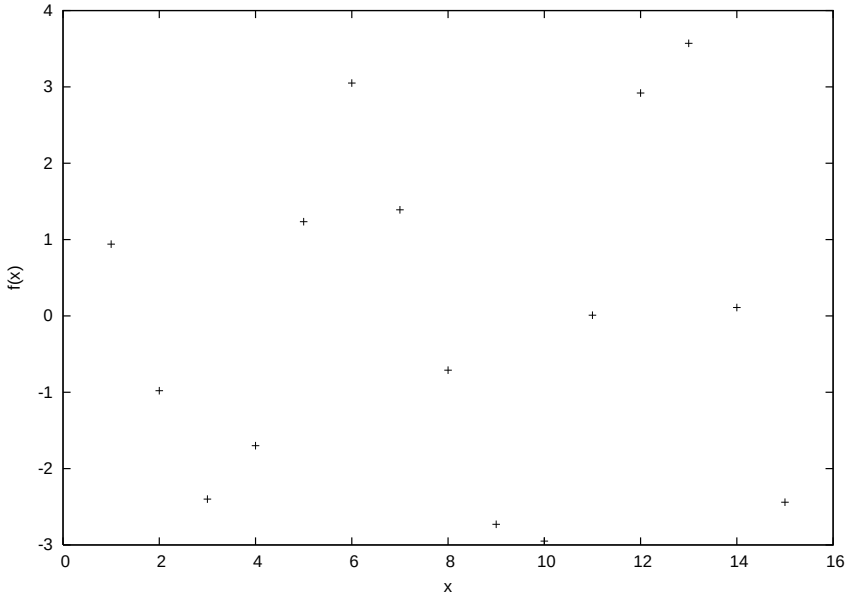


Figure 3.8: Graphical representation of the set of data points for GP fitting.

F={+,-,*,/,SIN,COS,TAN,SQRT}

V={x}

C={$\mathbb{R}$} range = [0,15]

The population size of each generation should be chosen large enough to allow for sufficient diversity in order to achieve reasonable performance and eventual optimal convergence. The population size is usually chosen to include thousands or even millions of individuals. The larger the functional and terminal component set, the larger the population should be. For the purposes of this example, we only use four representative individuals to illustrate the progression of the evolutionary process. Table 3.2 shows the four syntax trees, the plot of the functional representation of each tree along with the plot of the data points. Table 3.3 shows the fitness value of each of the four syntax trees using Equation 3.1.

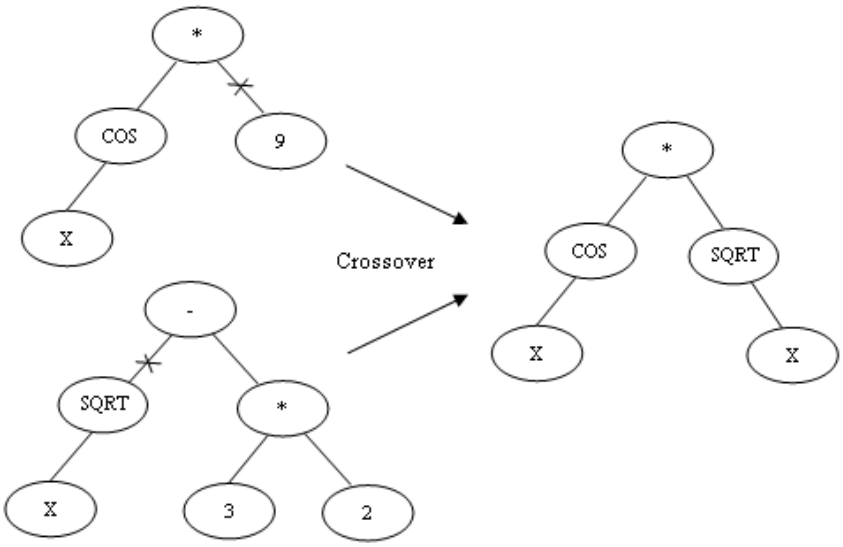
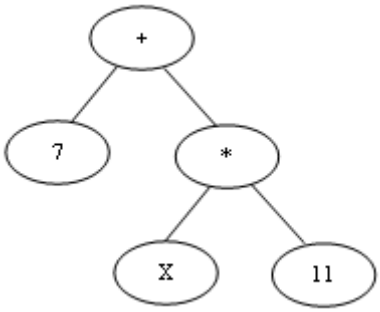


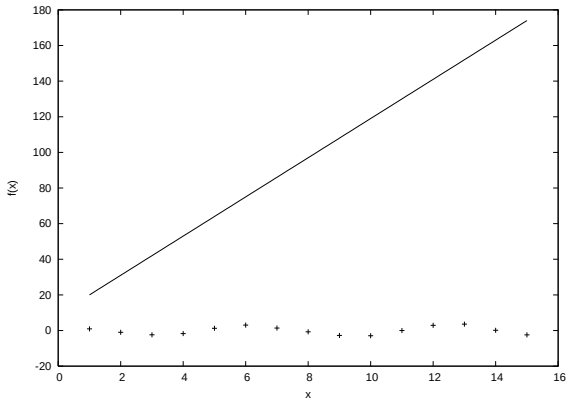
Figure 3.9: The crossover operation performed on syntax trees (b) and (c) yielding the tree representation of equation $cos(x) * \sqrt{x}$.

A crossover operation performed on syntax trees *b* and *c* (Figure 3.9) yields the function $cos(x) * \sqrt{x}$ which represents an accurate data fitting function for the given data. Table 3.4 shows the final fitness value achieved by evaluating the resultant syntax tree. The graphical representation is shown in Figure 3.10.

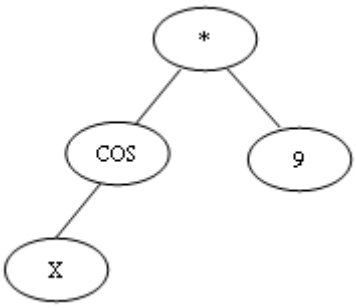
3.6 Limitations of Genetic Programming

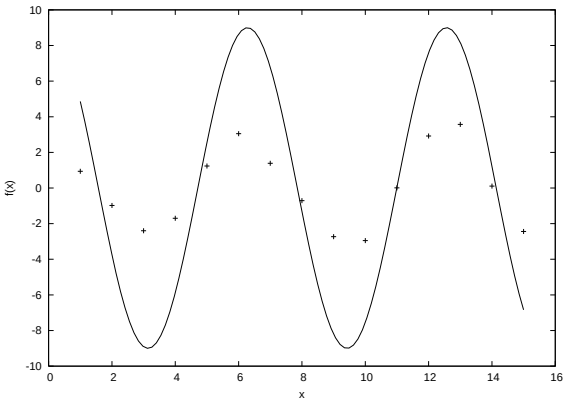
Genetic programming offers a more flexible approach than genetic algorithms due to its ability to develop a method for solving a problem instead of a point solution. However,



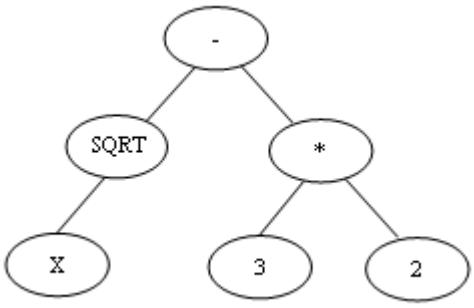


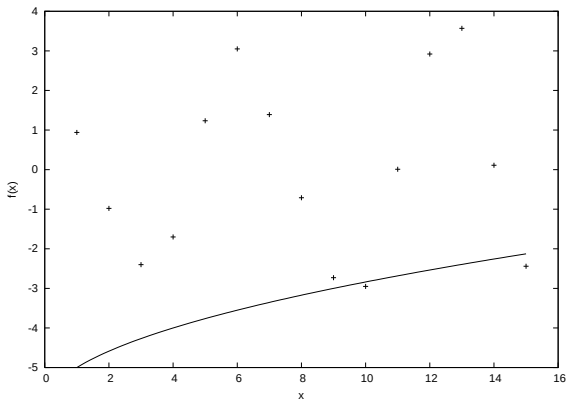
(a) $\{7+11x\}$



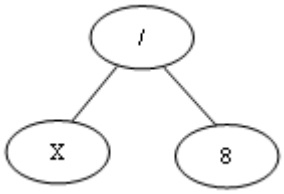


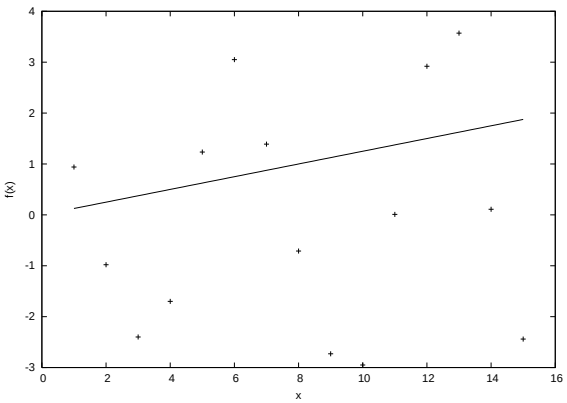
(b) $\{cos(x)*9\}$





(c) $\{\sqrt{x}-6\}$





(d) $\{x/8\}$

Table 3.2: Four representative syntax trees, the plot of the functional representation of each tree, as well as the plot of the original data points.

	Function	$f_i = \sum_{k=1}^{n} \lvert tree_i(k) - p(k) \rvert$
(a)	$7 + 11x$	1,425.69
(b)	$cos(x) * 9$	55.18
(c)	$\sqrt{x} - 6$	49.69
(d)	$x/8$	30.89

Table 3.3: The fitness evaluation of the four representative syntax trees.

Function	$f_i = \sum_{k=1}^{n} \lvert tree_i(k) - p(k) \rvert$
$cos(x) * \sqrt{x}$	5.58

Table 3.4: The fitness evaluation of the syntax tree representation of $cos(x) * \sqrt{x}$.

the unified representation of the genome and phenom utilized by the framework causes significant drawbacks. The main structure of GP-based evolution results in the following limitations:

- The solutions produced by genetic programming tend to drift towards larger and slow solutions on average [114]. Hence, a solution might be ultimately reached, yet no guarantee exists that the presented solutions is the most efficient.
- As the complexity of the solution increases, the reproduction of results becomes very difficult to achieve by applying modifications. Minor changes in the syntax tree could result in major changes in its associated functional representation.
- Genetic programming also suffers from the possible presence of insufficient diversity and the possibility of reaching sub-optimal results [30]. This possibility increases significantly as the size of the search space increases.

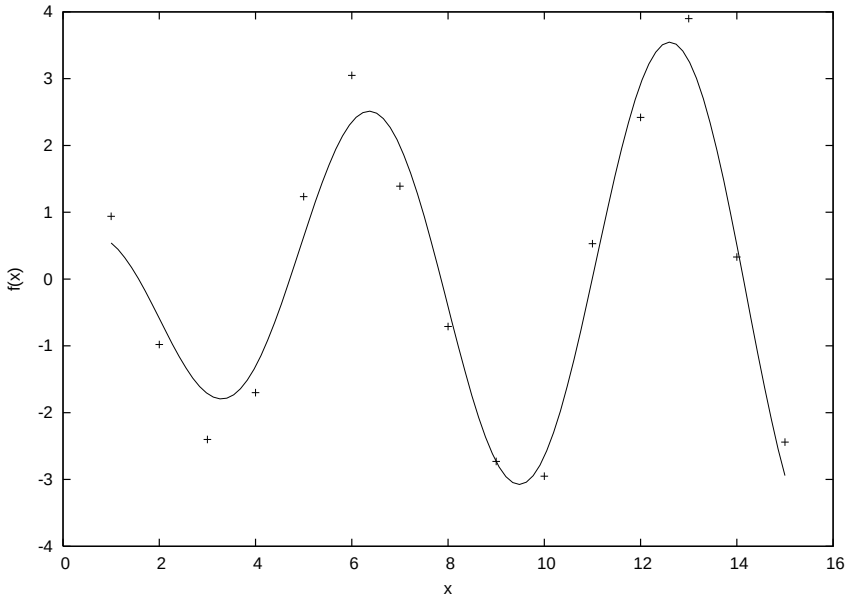


Figure 3.10: Final data fit using function $cos(x) * \sqrt{x}$.

- Most significant results produced by GP frameworks were obtained through the utilization of massive parallelism in order to achieve the processing power necessary for computing solutions within a reasonable time frame. Such requirements make the framework usage possibly prohibitive from a practical sense.

3.7 Gene Expression Programming (GEP)

Ferreira created Gene expression programming (GEP) [37] to overcome some of the limitations of GP. The main contribution of GEP is in its representational separation. GEP allows for the separate representation of the genome and phenom, which facilitates the performance of the genetic operations without losing the flexibility of evolving a method for solving the problem instead of a point solution. The genome is structured as a linear symbolic string of fixed length and is converted to its associated expression tree (ET) representation utilizing a specialized language known as Karva. Despite the fixed length of each symbolic string, different length expression trees could be produced depending on the structure of the string.

Consider the gene representation g, of the function set F, variable set V and constant set C:

F={+,-,*,/}

V={a,b}

C={$\mathbb{R}$} range = [0,5]

The following is an expression string representation of *g*:

+ + * a b 3 + a + 4

The translation from an expression string to an expression tree starts at the left most symbol in the string which represents the root of the tree. The process then moves to the right populating the parameters of functions placed at each level until all function parameters have been populated. Figure 3.11 shows the expression tree for the above expression string.

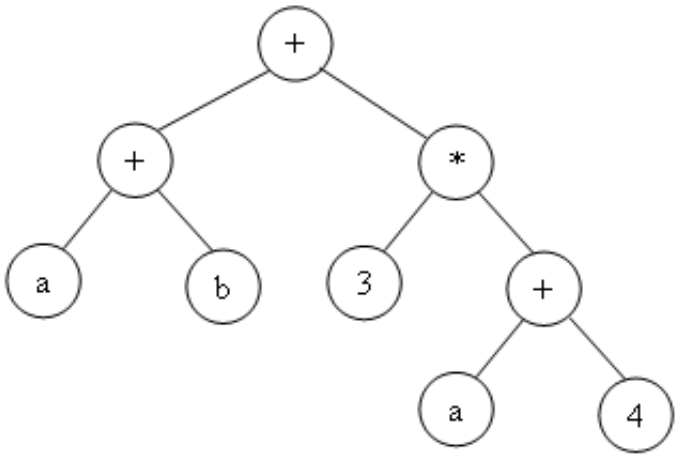


Figure 3.11: Expression tree representation for the expression string + + * a b 3 + a + 4.

The expression string of a gene is divided into two parts. The first part is the head *h*, which is composed of functions and terminals, while the second part is the tail *t* composed only of terminals. The length of *t* is a function of *h* following the formula

$$t = h(n-1)+1 \tag{3.2}$$

where n is the number of arguments of the function of the highest argument count. Using the sets F,V and C, we define another gene *g* described by the string

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6
/	a	*	+	/	*	b	9	*	3	a	a	b	b	3	a	1

In the string listed, the length of $h = 8$. Given $n = 2$, the length of $t = 8(1) + 1 = 9$. Although the gene contains seventeen symbols, only the tree representation determines the last symbol actually utilized. Figure 3.12 shows the representation for the string given. The figure demonstrates how only thirteen symbols are utilized in the expression tree.

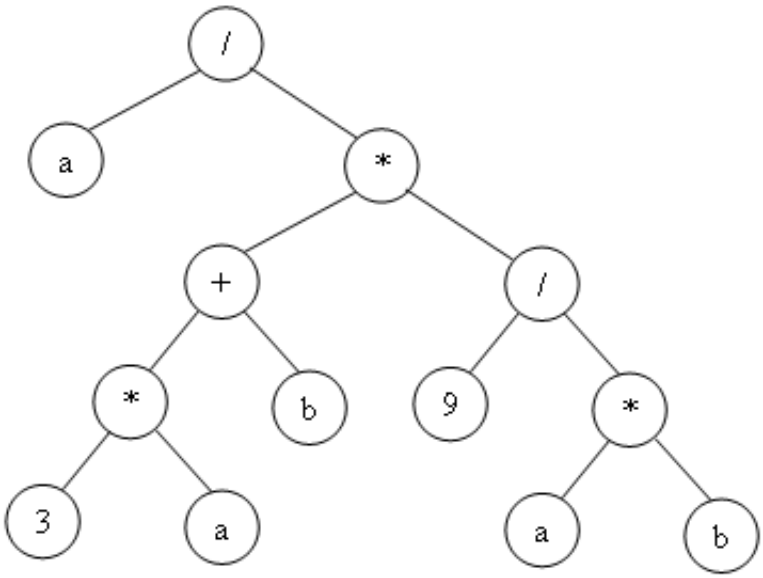


Figure 3.12: Symbol utilization in the expression tree.

As crossover and mutation operations cause alterations to the expression string, the number of symbols being utilized varies depending on the structure of the resultant string. For example, given a mutation operation that changes the tenth symbol from a 3 to a $+$, the resultant string will be structured as follows:

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6
/	a	*	+	/	*	b	9	*	+	a	a	b	b	3	a	1

The expression tree of the modified string is shown in Figure 3.13. We notice that the height of the tree has grown as the $+$ symbol replaces the 3 at the tenth position. We also notice that the number of utilized symbol has grown from 13 to 15.

3.8 GEP Genetic Operators

One of the main advantages of GEP lies in its ability to accept common genetic operators as they are normally performed on linear chromosome strings. The separation of the chromosome from its tree representation allows for genetic operators to be applied directly to

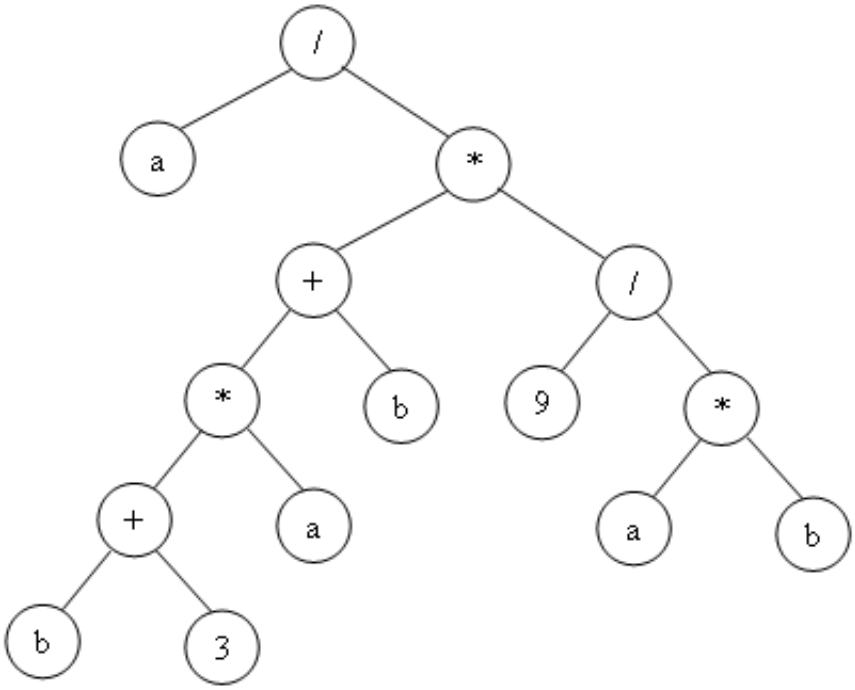


Figure 3.13: Altered expression tree after applying the mutation operator.

the expression strings without having to take the expression tree into consideration. In addition to the regular GEP selection methods, which are identical to the selection methods utilized by GA and GP , the following are the main genetic operators used in GEP:

- Mutation
- Recombination
- Transposition

As long as the expression string rules for the head and tail structuring are followed, GEP guarantees that any alterations done to the expression string by using any of the genetic operators would still yield an expression tree that is structurally correct.

3.8.1 GEP Mutation

The Mutation operator may be applied to any part of the chromosome without violating any of the organizational rules. The symbols in the head may be exchanged with any function or terminal. However, symbols in the tail may only be exchanged with terminals. A mutation parameter p_m is usually used to determine the probability of mutation.

GEP does not impose any restrictions on the number of mutations per chromosome, and any number of mutations should still produce a structurally correct expression tree [37]. Figures 3.12 and 3.13 show the before-mutation and after-mutation tree representations of the string

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6
/	a	*	+	/	*	b	9	*	3	a	a	b	b	3	a	1

Although the mutation only changed a single symbol (at the tenth location), it becomes evident how the mutation operator can cause drastic alterations to an expression tree increasing or decreasing its depth and possibly changing its structure.

3.8.2 GEP Recombination

GEP recombination is comparable to the GA and GP crossover operators where two individuals are selected and the genetic material is swapped between them. A recombination probability parameter p_r may also be used to determine the frequency of recombinations taking place. GEP utilizes three different types of recombinations:

- *One-point recombination* utilizes a single random mutation point and all the genetic material starting at this random points is swapped between the two individuals. For example, given the following two individuals:

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6
/	a	*	+	/	*	b	9	*	3	a	a	b	b	3	a	1
+	*	/	+	b	a	3	a	b	a	a	b	5	1	a	2	a

  A random mutation at location 4 would yield the following two chromosomes:

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6
/	a	*	+	b	a	3	a	b	a	a	b	5	1	a	2	a
+	*	/	+	/	*	b	9	*	3	a	a	b	b	3	a	1

- *Two-point recombination* utilizes two random mutation points instead of only one. The genetic material between the two points is swapped between the individuals. Given the original two individuals shown above and the two recombination points at locations 2 and 9, the resultant individuals would be structured as follows:

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6
/	a	/	+	b	a	3	a	b	a	a	a	b	b	3	a	1
+	*	*	+	/	*	b	9	*	3	a	b	5	1	a	2	a

- *Gene recombination* is used when the chromosome is partitioned into several genes. The genetic material is then swapped along the boundaries of a complete gene chosen at random. Consider the following two individuals divided into two separate genes as follows:

0	1	2	3	4	5	6	7	8	9
/	a	*	+	/	*	b	9	*	3
+	*	/	+	b	a	3	a	b	a

0	1	2	3	4	5	6	7	8	9
a	a	b	b	3	a	1	b	a	2
a	b	5	1	a	2	a	4	0	a

  A gene recombination of the second gene would yield the following resultant individuals:

0	1	2	3	4	5	6	7	8	9
/	a	*	+	/	*	b	9	*	3
+	*	/	+	b	a	3	a	b	a

0	1	2	3	4	5	6	7	8	9
a	b	5	1	a	2	a	4	0	a
a	a	b	b	3	a	1	b	a	2

  Utilizing only gene recombination without using other mutation or recombination methods limits the evolutionary process as no new genes could be created. In this case successful evolution would be dependant on the size of the population as an extremely large population would be the only means for having enough genetic diversity among individuals [38].

3.8.3 GEP Transposition

GEP transposition operates on a single chromosome randomly taking a section of the genetic code and duplicating it to another part within the same chromosome. In essence, the operator duplicates some of the genetic material within the genome, and at the same time, another part is deleted (replaced by the transposed section). For example, given the following chromosome

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9
/	a	*	+	/	*****	b	9	*	3	a	a	b	b	3	a	1	b	a	2

the section starting at positions 3 and ending at position 5 is chosen at random to be transposed. As the section is transposed downstream to position 8, the resultant chromosome is as follows:

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9
/	a	*	+	/	*****	b	9	+	/	*****	b	b	3	a	1	b	a	2	

The transposition application should maintain the head and tail rules. A function should never be transposed to the tail section of the chromosome in order to maintain the correct structure for the resultant expression tree.

3.9 Conclusion

Genetic programing as well as gene expression programing provide a flexible platform for the evolution of programmatic solutions to complex problems. The complexity of developing articulated robotic control, however, would require a much higher level of diversity than what GP and GEP could offer. The use of massive parallelism allows for the utilization of very large populations of possibly millions of individuals, yet such computational power is not readily available in dynamic learning environments. Due to such limitations, the possibility of reaching suboptimal control strategies is quite high due to the very large search space present. In order to utilize genetic methods for the successful development of robotic controllers for complex articulated motion control, constructs are needed for the intelligent bounding of the problem search space. Such constructs would decrease the time needed to reach a solution and also decrease the possibility of reaching suboptimal results.

Chapter 4
GUIDED GENETIC EVOLUTION

4.1 Introduction

In this chapter, we formally define the *Guided Genetic Evolution* (GGE) platform. The platform introduces new concepts for the evolution of autonomous robotic controllers for the real-time control of articulated structures. GGE builds upon genetic programming methodologies with the inclusion of specialized algorithms for the evolution of articulated robotic controllers and guiding the evolutionary process in order to achieve faster convergence time and minimize the possibility for suboptimal convergence. Genetic guidance is achieved through the minimization of the problem search space by reducing the problem parameter count and applying constraints to the evolutionary process while maintaining the genetic diversity within the population.

Guided genetic evolution is motivated by *imitation-based learning* principles, yet it circumvents the current existing limitations that render such principles unachievable in a practical sense. The framework allows the designer to incorporate articulation patterns and constraints to be followed by the agent on an imitative basis. This approach guides the genetic process by biasing exploratory moves towards predefined areas of the search space. Since the evolutionary constraints are presented in an explicit sense, the *correspondence problem* associated with the imitation process does not apply.

The GGE platform allows for the expansion of the evolutionary process into a higher more complex level where comprehensive state determination and transitioning may be achieved. Complex states may be obtained through the genetic evolution of complex action sequences. GGE includes constructs for the specification of the agent's genetic structure as well as the guidelines for the evolutionary path. Constructs for the specifications of the concurrency of execution threads are also present in the genetic process in order to maximize an individual's fitness through the dynamic optimization of execution sequences.

Several types of constraints may be applied to the genetic process in order to achieve the desired evolutionary guidance. The level of commitment to any of the specified constraints maybe predefined or managed dynamically as the agent's environment changes. The evolutionary process aims to optimize the constraint parameters in order to maximize fitness. Such optimizations may be achieved through training before the agent's

life cycle begins, while others maybe developed through the use of dynamic optimization techniques during the agent's life cycle through the interaction between the agent and the environment.

4.2 Genetic Structure

In this section, we discuss the general genetic structure associated with the GGE platform. Further formalization of the different structural constructs will be discussed later in the chapter. The GGE platform is based upon genetic programming principles; hence, the learning methodologies utilized are based upon the genetic evolution of individuals over multiple generations until the system converges to a desired level of performance. In addition to the evolutionary constraints applied to the genetic process, core genetic operators are applied to each generation of individuals allowing the overall fitness to progressively increase over evolutionary time. Figure 4.1 demonstrates the overall genetic process utilized for the structured evolution of individuals.

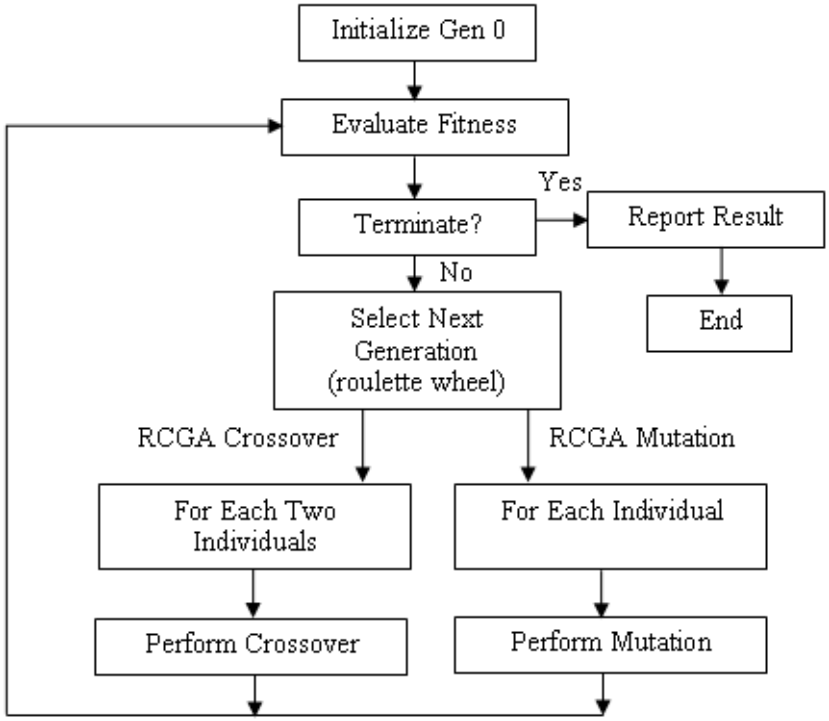


Figure 4.1: Overall genetic process utilized for the structured evolution of individuals.

Real coding (RCGA) is utilized for the representation of chromosomes, so the genetic

operators utilized are geared towards the manipulation of real-valued parameters. Three main reasons motivate the use of RCGA in guided genetic evolution:

- The nature of the robotic control problem requires the use of real-valued parameters. RCGA utilization facilitates the structuring of the genetic process as the genomes represent actual parameters that are applied directly towards a solution.
- The size of the search domain for the control problem is significantly large. The use of RCGA methodologies allows for the exploration of the associated domain without loss of precision.
- The optimization of the different control strategies requires the utilization of *graduality* in order to progressively achieve more optimal results. RCGA methods are most apt for achieving the required gradual fine tuning of problem parameters.

4.2.1 Initialization

The first generation of individuals is initialized using a purely random process to guarantee an even distribution over the target search space, given that the population size is of an appropriate size[1]. On the other hand, if the population size is limited, a non-clustering method is used to promote the needed population diversity. The presence of an appropriate level of diversity within the initial population greatly affects the solution convergence speed as well as the quality of the results. The initialization process is structured to follow the evolutionary constraints placed on the genetic process; hence, prior to the application of any genetic operators, individuals are placed in areas of the search space that maximizes their chances of achieving significantly higher fitness values. Such selective initialization also significantly decreases convergence times as individuals are leaped forward on the evolutionary scale as they are placed in desirable areas of the search space.

4.2.2 Selection

GGE utilizes a *fitness-proportionate selection* method based on the *roulette-wheel* selection principles. Once the fitness of each individual c_j is evaluated, the following formula is used to calculate the selection probability $P(c_j)$:

$$P(c_j) = \frac{\varphi(f(c_j))}{\sum_{k=1}^{m} \varphi(f(c_k))} \tag{4.1}$$

[1]The significance of population size is directly related to the nature of the problem.

where $f(c_j)$ is the fitness value for individual c_j, m is the number of individuals in the generation, and $\varphi : \mathbb{R} \rightarrow \mathbb{R}^+$ is a non-decreasing transformation used to shift the fitness values to $\mathbb{R}^+$. The transformation is performed only for selection purposes without varying the reported fitness values. This guarantees the ability to monitor the unaltered fitness of individuals and the population as a whole from one generation to the next.

In some instances, a need exists for reversing the resultant probabilities. For example, if the goal of an experiment is to keep a robotic agent as close to the point of origin as possible, then either the fitness function would have to be inversely proportionate to the distance from the origin, or it may be proportionate to the distance and a probability reversal may be applied to produce correct selection results. The formulate for probability reversal is defined as

$$P(c_j) = \frac{(1 - P(c_j))}{\sum_{k=1}^{m} (1 - P(c_k))} \tag{4.2}$$

Although *fitness-proportionate selection* tends to favor individuals with higher fitness values, the process does not eliminate the possibility of selecting individuals with lower fitness. This deviation from pure *elitist selection*[2] acknowledges the need for the application of comprehensive randomness in the evolutionary process, as the eventual selection of individuals with low fitness is an essential component in the system exploratory strategy.

4.2.3 Crossover

The GGE crossover operator uses RCGA methodologies to alter the genetic encoding of individuals. The operation promotes the replication of desirable traits among individuals, as it operates on two parent individual swapping genetic material between them in order to produce offspring. GGE utilizes the BLX - α RCGA crossover operator, which allows for the problem parameters to be gradually focused by selecting a random gene value from the expansion of interval $[min(x_i^1, x_i^2), max(x_i^1, x_i^2)]$ where x_i^1 represents gene i for the first parent individual and x_i^2 represents gene i for the second. This method allows for the expansion of the selection interval by an expansion parameter α. The BLX - α crossover interval is defined as

$$[min(x_i^1, x_i^2) - I \cdot \alpha, max(x_i^1, x_i^2) + I \cdot \alpha]$$

[2] Elitist selection guarantees the selection of the most fit individuals of each generation.

Where

$$I = max(x_i^1, x_i^2) - min(x_i^1, x_i^2)$$

The α parameter is problem specific and must be chosen carefully to allow for the gradual expansion of the target domain without deviating from the evolutionary results already achieved. The dynamic management of the α parameter is included in the GGE framework. An initial α value is chosen taking into consideration the overall range of possible values allowed for the specific parameter to be evolved. The parameter is then gradually decreased as evolution proceeds in an effort to further focus the results. Let g_{max} be the maximum generation number desired, g is the current generation number, and α_{min} and α_{max} are the minimum and maximum α values respectively, the dynamically managed BLX - α interval is defined as

$$\left[min(x_i^1, x_i^2) - I \cdot \left(\alpha_{\max} - \frac{g \cdot (\alpha_{\max} - \alpha_{\min})}{g_{\max}} \right), max(x_i^1, x_i^2) + I \cdot \left(\alpha_{\max} - \frac{g \cdot (\alpha_{\max} - \alpha_{\min})}{g_{\max}} \right) \right]$$

The values α_{min} and α_{max} maybe predefined by the designer, or they maybe automatically generated by the GGE platform as a percentage of the search domain size. The α value may still be fixed over all generations by supplying the same identical value for α_{min} and α_{max}.

4.2.4 Mutation

In order to guarantee continued population diversity, GGE uses a mutation operator that alters the genetic encoding of individuals on a singular basis relying on a predefined mutation probability p_m. The mutation operator allows for the evolutionary process to take exploratory moves into areas of the search space that may have not been explored previously. Such moves prevent any area of the domain from being out of evolutionary reach. It also reduces the probability of suboptimal convergence and increases the quality of the results. The p_m value is problem specific and must remain small in order to preserve evolutionary results while still exploring different areas of the search domain. In addition to the mutation probability p_m, the mutation operator requires the two values m_{min} and m_{max} defining the bounds of the target domain for the specific gene. The random value r is chosen from the interval $[m_{min}, m_{max}]$. Non-uniform mutation is then applied to the chosen gene c_i with probability p_m following one of the following two formulas:

$$c_i' = x_i + \Delta(t, b_i - x_i)$$
$$c_i' = x_i - \Delta(t, x_i - a_i)$$

where

$$\Delta t(t,x) = x \cdot \left(1 - r^{\left(1-\frac{g}{g\max}\right)^b}\right)$$

As the generation number g approaches g_{max}, the impact of the random value r on the gene decreases. The value b may be used to increase or decrease the significance of the generation number in the application of the mutation operator.

4.3 Trigger Networks

In this section, we present a new type of connectionist network model labeled *Trigger Networks*, or *T-Nets*, developed as the representational core of guided genetic evolution. This connectionist model is the main representation vehicle for the encoding as well as the evolution of agent control strategies. T-Nets allow for the representation of all aspects of the control problem in an evolvable manner. Genetic evolution methodologies are then applied to the model in order to optimize the control strategy by altering the problem parameters until the predefined fitness criteria has been met.

Trigger networks utilize a hierarchical structure linking multiple levels of subnets using evolvable links. The hierarchical representation allows for the structuring and evolution of complex behaviors based on simpler actions and behaviors represented by other parts of the network. Although guiding constructs are used as a mechanism for reducing the complexity of the problem search space, the essence of the evolutionary process remains intact allowing for the genetic optimization techniques to structure the network according to the performance of individuals without any external intervention beyond the initial design.

Figure 4.2 demonstrates the general evolutionary cycle of trigger networks. The detailed description of a robotic agent is encoded as a trigger network which represents all the actions the agent could execute as well as all the possible relationships that govern the execution of such actions. Each encoded trigger network represents a control strategy to be utilized for agent control. The performance of each trigger network is evaluated through the use of one or more fitness functions which play a crucial role in the selective evolution of network populations.

Several essential building blocks are utilized as constructs to be combined as an evolvable trigger network. A T-Net can be structured as simple as a single action to be performed by an agent or as complex as hundreds or even thousands of subnetworks representing different behavioral strategies and combined together into a single agent con-

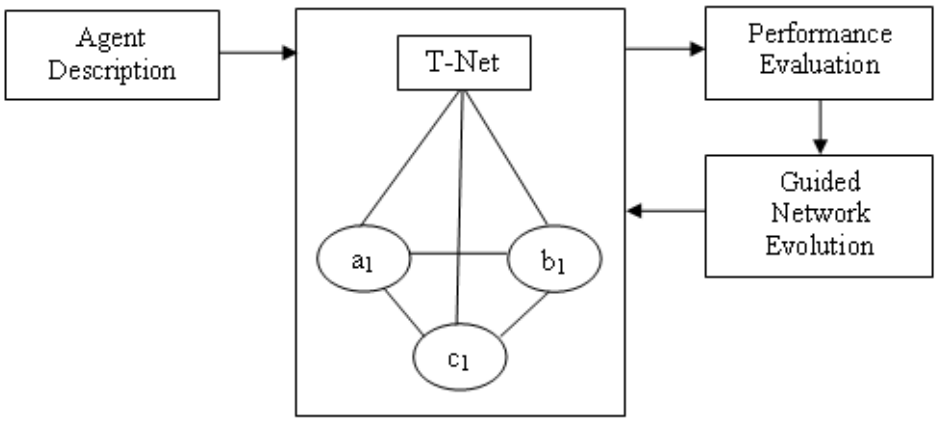


Figure 4.2: Trigger network evolutionary cycle.

troller. The following sections describe the different elements that represent the core building blocks utilized in the construction of trigger networks.

4.3.1 Action Nodes

Action nodes represent the set of primitive behaviors performable by an agent. In an articulated robotic structure, a primitive action represents the actuation of a joint motor setting the joint bodies at a specific relative angle. In T-Nets, an action a_i is represented by the following symbol:

where *i* is the unique ID of the action being referenced.

Each action represented within the network carry several internal evolvable parameters used in the evolutionary process in an aim to optimize performance. For a primitive joint motor actuation action, the internal evolvable parameters are as follows:

- The target angle vector to be reached by the joint. The value for this parameter has to fall between the low and high constraint vectors placed on the joint at design time. The number of vector elements depends on the number of axis utilized by the joint.

- The maximum torque vector to be applied to the joint bodies. This parameter represents the strength of the motor, or *muscle*, in control of the joint.

- The maximum allocated time duration for the completion of the desired action. This parameter represents the urgency associated with action execution.
- The input trigger vector array which determines the input trigger points for the node.
- The output trigger vector array which triggers the execution of other dependant actions.
- The priority value used to determine the node priority level in relation to other simultaneously executed entities within the network.

4.3.2 Trigger Vectors

Trigger vectors are used to represent dependencies as well as execution sequencing within the network structure. Any complex action performed by an agent consists of multiple primitive actions executed using a specific concurrency and sequencing structure. Any action may contain zero or more trigger vectors connected to other action nodes demonstrating execution dependency. A trigger relationship is demonstrated by an arrow connecting two nodes. Figure 4.3 shows multiple possible trigger vector representations.

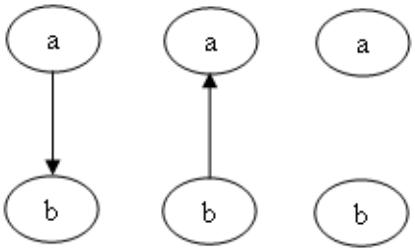


Figure 4.3: Trigger vector representations. Left: trigger dependency of action *b* on action *a*. Middle: trigger dependency of *a* on *b*. Right: no trigger dependency present.

The direction of the trigger vector, determined by the sign of the vector magnitude value, demonstrates the dependency present between actions. The magnitude of the vector determines the trigger point relative to the execution duration of the trigger source action. Given the dependency of action b on action a, if a has a total associated time duration t_a, then the magnitude of the trigger vector $\vec{v}$ would determine the point in time when the trigger becomes active. If $|\vec{v}| = 0.5$, and a is triggered at time t_0, then b is triggered at time $t_0 + 0.5t_a$.

Trigger vectors may also be utilized to commence the executions of actions simultaneously. A vector magnitude of zero would cause b to trigger as soon as a is triggered. Similarly, actions may be executed in succession by setting $|\vec{v}| = 1$. In this case, b would commence immediately after a has terminated. Setting $|\vec{v}| > 1$ would cause a delay after the termination of a before b is executed, and the delay is defined as

$$t_a \cdot (|\vec{v}| - 1)$$

Each action node a contains an input vector array labeled $iVEC$ referencing the source nodes on which a dependency exists. In order for an action to commence execution, all iVEC elements must be triggered by their respective source nodes. Let action a hold an iVEC array of length two containing the two entries $\vec{v}_b$ and $\vec{v}_c$ referencing the action nodes b and c respectively. This relationship means that action a is dependant on both b and c in its execution. In other words, both b and c must trigger a based on the magnitude of their respective trigger vectors, and only then may a start its execution cycle.

We define the fluent EX to state the executability of any action within the environment. The array iVEC holds triggering information denoted by a zero for each element awaiting a trigger signal from the source node and a 1 if the trigger signal has already been received. The fluent EX is defined as

$$\forall a, EX(a) \rightarrow \left[\sum_{k=1}^{|a_{iVEC}|} a_{iVEC}(k) = |a_{iVEC}| \right] \tag{4.3}$$

Each action node a also contains an output vector array labeled $oVEC$ that holds trigger information for other nodes that has dependency on a. Each element of $oVEC$ is an independent entity that holds both a reference to the target node as well as the trigger magnitude indicating the desired trigger point. The elements are continuously processed during execution, and once the trigger point has been reached, the associated target node is immediately triggered. Figure 4.4 shows several possible dependency configurations demonstrating multiple action nodes and trigger vectors.

4.3.3 Root and Relay Nodes

Root and Relay nodes are a special type of action nodes. The nodes do not contain an internal executable action. A root node is unique, as it identifies the entry point for executing the entire trigger network. Each trigger network may contain only a single root node which holds an $oVEC$ array connected to other relay or action nodes which are to be

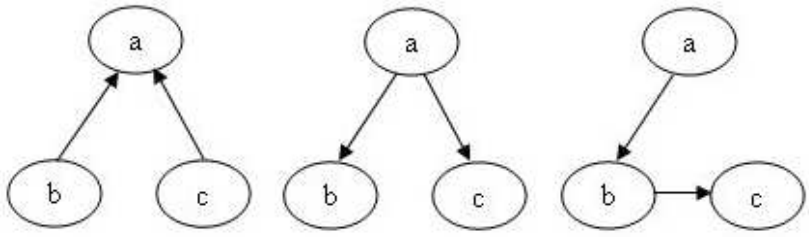


Figure 4.4: Multiple T-Net trigger connections. Left: *a* is triggered by both *b* and *c*. Middle: *a* triggers both *b* and *c*. Right: *a* triggers *b*, and *b* triggers *c*.

triggered at the beginning of the network execution cycle. However, the root node does not hold an *iVEC* array, since it is triggered directly to commence network execution.

Relay nodes hold an *iVEC* array which receives trigger signals from other action or relay nodes. They also hold an *oVEC* array for triggering other actions. Usually, a relay node is utilized as a subnetwork header. When a subnetwork representative relay node is triggered, it initiates the execution of the entire behavior. The *oVEC* element magnitudes are then utilized to determine the trigger points for the execution of individual actions within the subnetwork.

The network root node is represented by the following symbol:

The relay node r_i is represented by the following symbol:

Once a particular behavior has been genetically achieved, the subnetwork associated with the behavior may be represented graphically using only its header (relay node) representation. As the number of behaviors increase, such compact display allows for the visual simplification of the network. The same principles are followed as multiple behaviors are combined into a single complex behavior, as the end result may also be represented using a single relay node.

4.3.4 Reset Nodes

Reset nodes are also a special type of action nodes that do not contain an internal action to be executed. However, a reset node is responsible for returning the state of the network to

an initial state that existed before the network was first triggered. The utilization of reset nodes is crucial to the execution of cyclic behavior within the network. For example, in order for a biped robot to execute a walking behavior, the agent is trained to step with one leg, then the other, then the whole process is repeated to achieve the desired goal of walking.

The network reset node is represented by the following symbol:

A reset node may be used to reset the network then trigger the root node to commence network execution once more, or it may be used to trigger a specific relay node in order to cycle through a specific behavior while the rest of the network follows its normal execution path.

4.3.5 Subnetworks

A subnetwork is a part of a trigger network that represents a grouping of primitive actions combined into a single behavior. A complete subnetwork headered by the relay node r_i is shown in Figure 4.5. An unguided subnetwork representing a desired behavior would initially contain the following elements:

- Action nodes representing all possible primitive actions performable by the agent. Without the presence of genetic guidance, a T-Net has no knowledge of which actions might contribute towards the fulfilment of a particular goal or the execution of a desired behavior. Hence, all possible actions are included in an aim that the self-organization genetic process would eliminate irrelevant nodes.

- Trigger vectors connecting all possible two-node groupings of actions in both directions. Although not all trigger vectors will be needed, the unguided network has no way of knowing which of the vectors would eventually be relevant to the achievement of the desired goals.

The evolutionary process itself aims to eliminate irrelevant trigger vectors and action nodes along the evolutionary path. As particular actions or dependencies within the subnet prove unnecessary or counter productive in relation to the achievement of a desired goal, the nature of the genetic process helps eliminate such entities while maximizing the overall fitness of individuals. As the i_{VEC} elements of a particular action node are all eliminated, the action becomes irrelevant as it will never be executed during the agent's life

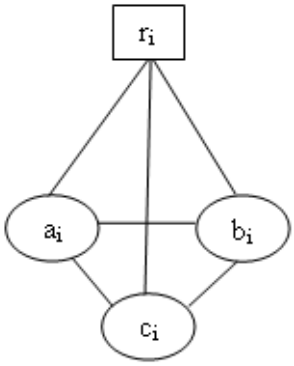


Figure 4.5: Representation of the subnetwork r_i.

cycle. Hence, this particular action may be removed without any impact on the behavior of the network.

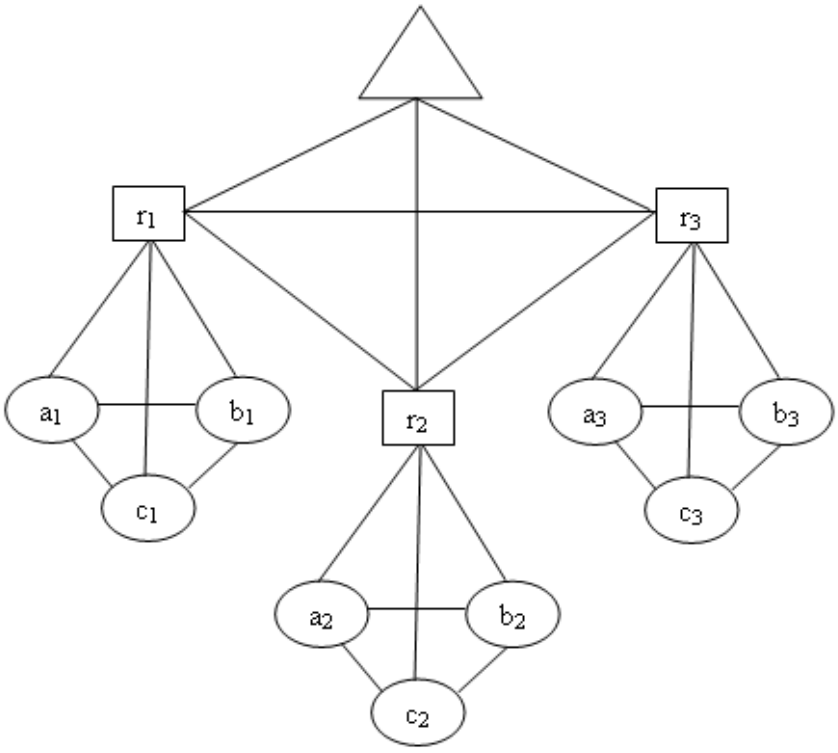


Figure 4.6: Connectivity of multiple subnetworks within a single trigger network.

Figure 4.6 shows a more complex representation of a T-Net consisting of the three subnetworks r_1, r_2, and r_3. The figure demonstrates the structuring of complex behavior based on simpler behaviors that consist of several primitive actions executing simultane-

ously. This hierarchical structuring may be utilized in building more and more complex behaviors based on more primitive subnetworks.

4.3.6 Concurrency

A typical rigger network may have several actions executing simultaneously at any given moment in time. A priority value is given to every action or relay node in order to determine nodes priority level over other actions executing at the same time. For example, given that being balanced is a current goal the agent is pursuing, then no action execution should be allowed that would affect the agent's ability to remain balanced. Hence, all posture alterations has to be performed prior to the balancing behavior being executed in order for the system to correct initiatives that would negatively affect its high priority balancing goal.

The trigger network consisting of the two subnetworks r_1 and r_2 is shown in Figure 4.7. Each relay node contains its priority value p denoting its priority level relative to other concurrent executions.

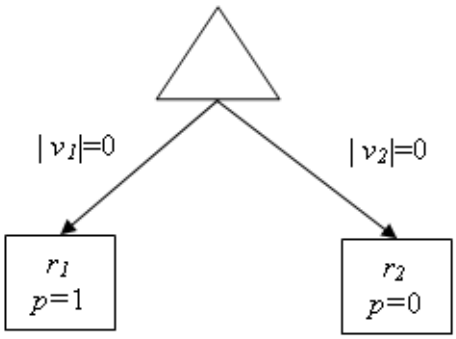


Figure 4.7: Trigger network consisting of the two subnetworks r_1 and r_2.

Once the network root node has been triggered, it will cause the immediate and simultaneous triggering of both subnetworks r_1 and r_2 given that the magnitude of their associated trigger vectors v_1 and v_2 are both zero. A vector magnitude x that is greater than zero causes a delay of x units before the associated action is executed. This delay is useful in achieving execution strategies involving actions which require different starting points.

The two subnetworks execute independently in an effort achieve their respective goals. However, prior to each execution cycle, the subnetworks are sorted according to their

priority values to determine the execution order of each behavior [3]. Subnetworks with the lowest priority execute first giving higher priority behaviors the ability to correct any negative effects caused by previous actions. Similarly, actions within any subnetwork are also given priority values to determine their own execution order.

In the example given, having a priority $p = 0$, subnetwork r_2 executes first by issuing its associated actuation commands in order to achieve its behavioral goal. Subnetwork r_1, having a higher priority, then proceeds to issue its own commands correcting any negative actions that might have been issued by r_2.

4.4 Action Types

Several action types may be utilized as part of the overall control strategy. The types of actions used depend on the specific behavior being implemented and the associated behavioral goals. An action may consist of a primitive command being issued to a specific joint, or it may contain a more complex algorithm for maintaining a certain configuration of joints over a period of time. Actions themselves are also subject to evolutionary alterations in an aim to achieve the most optimal command or algorithm definition to be used for solving the control problem.

4.4.1 Direct Joint Control

The first type of action to be discussed involves sending control signals to a specific joint forcing the joint into a particular configuration. The command is typically given as a combination of the desired joint angles for all axis controlled by the joint as well as the urgency involved in achieving the target angles. The following types of direct joint control signals are available:

- *Hold:* This command prompts the rotation of joint bodies in order to achieve the desired angle given. The torque applied to the bodies would depend on the urgency associated with achieving the desired configuration. The angle vector specified would be held by the joint until a different command is given that requires a different configuration (Figure 4.8). A resistance parameter may also be used to specify the maximum amount of torque allowed for maintaining the target angle.

[3] Although execution ordering exists, it does not imply any sequencing relating to the subnetworks unless direct trigger vectors exist between them. The execution ordering, however, decides on the order for the application of low level action commands or actuation signals

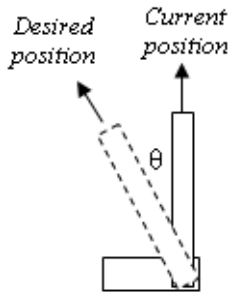


Figure 4.8: Direct angle control of a specific joint axis.

- *Free:* A joint would usually be given low and high stop points depending on the constraints of the articulated structure. A *joint-free* command maybe issued for all axis controlled by the joint or for any single axis. The command prompts the agent to remove any constraints placed on the joints except for the low and high stop points restricting the rotational motion of the bodies.

4.4.2 Joint Control Strategy

As an expansion of joint control, more complex localized goals may be embedded within an action node along with a strategy to achieve it. For example, if the desire is to eliminate the moment due to gravity in relation a particular axis, the associated joint may be invoked to perform continuous joint adjustments until the desired balance is achieved. For example, if an agent is on a slope at the time of behavior execution (Figure 4.10), the exact target angles for keeping steady balance are not known; however, the proper angles may be determined by following an overall strategy to reduce the moment on the body.

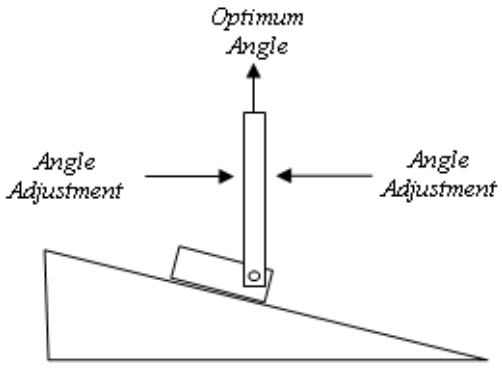


Figure 4.9: Joint control strategy for a body balancing behavior.

Several control strategies may be used in combination to achieve a desired effect. The

following list identifies some of the possible strategies which could be applied as part of a behavior:

- Eliminate moments about one or more axis of a particular body.
- Maintain primary normal vectors of a particular body in a certain configuration.
- Maintain even contact between two bodies; for example, we may desire to maintain even contact between the agent's foot and the ground. As an uneven pressure distribution is sensed, corrective measures may be taken to adjust the position of the foot.

4.4.3 PID Control

PID control is a very effective method for producing control output signals for guiding a control process towards a specific target. PID stands for Proportional, Integral, and Derivative, and the three terms represent the three phases utilized for the calculation of corrective control signal. Each of the three component play an important role in the overall control strategy utilizing and acting on different parts of the control process.

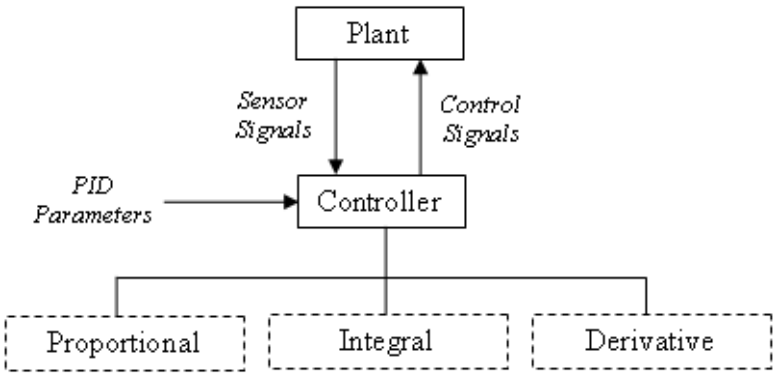


Figure 4.10: Components of a PID controller.

PID control utilizes a continuous loop where measured controller input is compared with a target reference point. The difference, or error, between the two values is the basis for calculating a new output control signal in an effort to reduce the error. In addition to basing the control signals on the error value, the controller also utilizes the accumulation of error values as well as the error rate of change in order to produce the most optimum and stable control strategy. The PID controller loop consists of three main phases:

- Sensory input relays system state,

- Controller calculates output signal based on input values, PID parameters, and control history,
- Controller issues output signals through control channels.

The parts of the PID controller, working in unison, react according to the present, past and future of the control process. The three partitions are defined as follows:

- *Proportional:* Reacts to the current state of the system. The process simply measures the current state and compares it to the desired reference point producing an error value e. The output signal is proportional to the error value using the following equation:

$$u_p = K_p \cdot e \tag{4.4}$$

  where u_p is the proportional output signal, and K_p is a constant parameter representing the *proportional gain*.

- *Integral:* Uses the history of error build-up to alter the output signals in an effort to reach the desired reference point. The integral portion of the controller helps reduce system oscillation that may occur due to overshooting the desired target configuration. The integral output signal u_i is defined as:

$$u_i = K_i \int e \cdot dt \tag{4.5}$$

  where K_i is a constant parameter representing *integral gain* which controls the significance of error buildup in the overall control strategy.

- *Derivative:* Utilizes the first derivative of the error value e to measure the responsiveness of the system to the control signals being issued. The derivative output signal u_d is based on the negative constant parameter K_d which represents *derivative gain* and is defined as:

$$u_d = K_d \frac{de}{dt} \tag{4.6}$$

The PID controller output is calculated as the sum of the three control parts and is defined as:

$$u = K_p \cdot e + K_i \int e \cdot dt + K_d \frac{de}{dt} \tag{4.7}$$

The three PID parameters K_p, K_i, and K_d play a crucial role in determining the overall behavior of the controls strategy. The fine tuning of the parameters may drastically increase the accuracy and stability of the control system. Hence, all three parameters are

genetically evolvable for each action within the trigger network. If an action is chosen to be controlled by a PID controller, the PID parameters are initialized randomly, then the genetic process works on optimizing the output signals based on the performance of the control strategy. The performance is measured based on an evaluation function specific to the type of action being carried out.

4.5 Trigger Network Evolution

The evolution of trigger networks is based on the optimization of overall agent performance through the fine tuning of behavior inclusion, action parameters, and trigger vectors properties in each part of the network. Without the utilization of guidance constructs, all the different network parameters are initialized using randomly chosen values. As the performance of individuals within each population is measured, gradual changes are made to the network in an effort to optimize performance.

4.5.1 Evolution of Trigger Vectors

Trigger vector magnitudes and activation parameters are used to determine the dependency levels and properties among network components. Initially, trigger vectors connect all network components and their magnitudes are initialized randomly. An evolvable *activation* parameter is utilized for each trigger vector to allow the system to eliminate the vector dependency as part of the evolutionary process. The genetic operators utilized may alter the trigger connections in any of the following ways:

- The relevancy of a trigger vector may be altered through the manipulation its activation parameter. If all input trigger vectors of an action become disabled, the action becomes isolated and is not used in the execution of the behavior.

- The trigger vector magnitude may also be altered by the genetic process. The vector magnitude determines the sequencing of action execution as it dictates the point in time at which a dependant action starts its execution cycle. Vector magnitudes may be altered to cause actions to start execution in synch, in succession, or utilizing any other configuration that may be applicable. The direction of the vector could also be altered to reverse the dependency direction between two nodes.

The possibility does exist for the eventual presence of circular dependency within the network, and if such a scenario occurs, behavior progression would simply halt as action nodes wait on a trigger signal that would never materialize. Such a situation takes place

when trigger vectors follow a circular path where an action is indirectly dependant on itself. Figure 4.11 demonstrates the existence of such a scenario. In the presented network, action a_1 is triggered by an outside vector as well as another vector originating from a_3. Since a_3 is triggered by a_2 and a_2 is triggered by $a1$, none of the actions will trigger. The evolutionary process is not affected by the existence of circular referencing within the network. Such an event would cause the representative individual to be rapidly eliminate through the selection process due to the resultant low performance of the agent. Such rapid elimination of the individual prevents the propagation of harmful configurations in following generations.

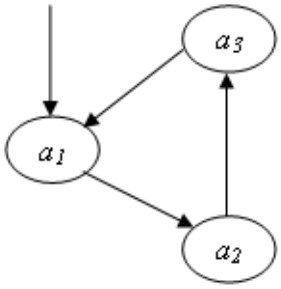


Figure 4.11: Circular trigger vector dependency involving multiple actions.

We present an example based upon the evolution of a robotic arm controller to demonstrate the evolution of trigger vectors. The configuration presented is shown in Figure 4.12. The robotic arm consist of a base, two arm components, and two hinge joints each utilizing a single degree of freedom. The goal of the evolution process is to evolve the arm behavior in order to move towards a target location without any predefined strategy for motion.

The initial trigger network representation of the control problem consists of two subnetworks each controlling the movements of one of the joints. The most optimum strategy should bring the arm closest to the target within predefined time duration. The aim of the genetic optimization process is to accomplish the following:

- To find the best angle values for the joint control commands.
- To find the most appropriate trigger vector magnitudes, which determine the movement execution sequence.

Figure 4.13 shows the trigger network representation of the robotic arm problem. In addition to evolving the parameters utilized within each action, the network layout itself

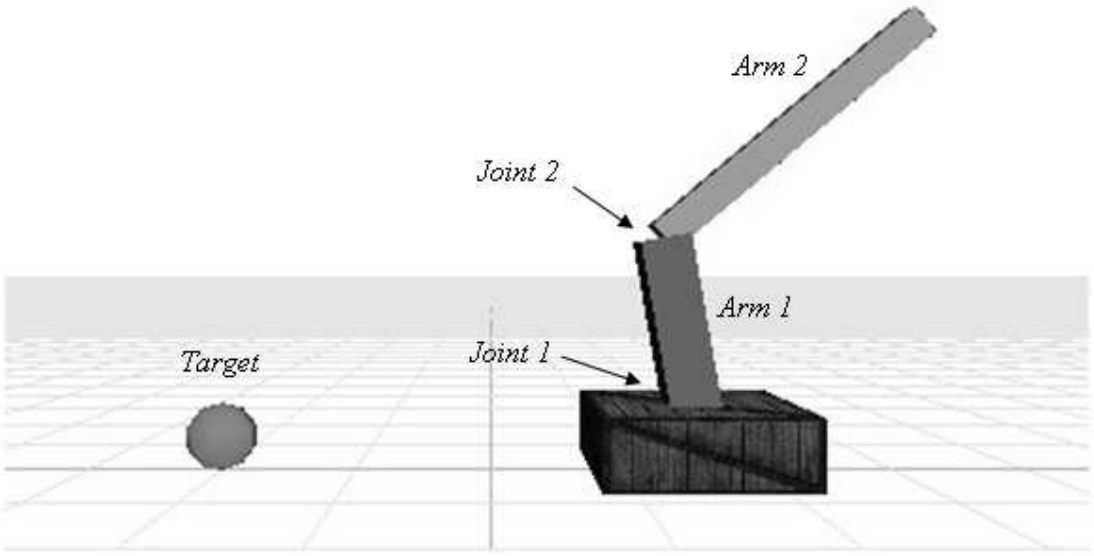


Figure 4.12: Trigger vector evolution of a robotic arm controller.

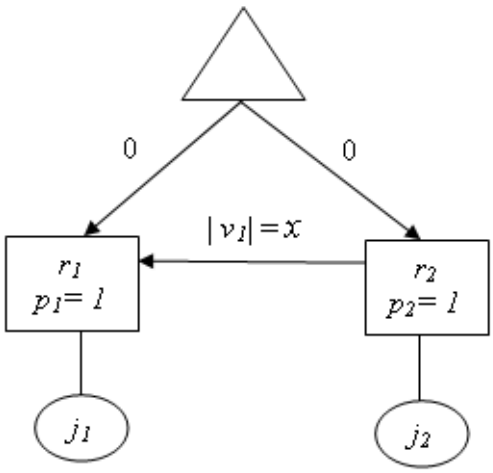


Figure 4.13: Trigger network representation of the robotic arm problem.

is altered through the evolution of the trigger vector magnitudes. The priority values for both subnets are set to 1.0 as the probability for the two actions contradicting each other does not exist. Also, both elements of the *oVEC* array of the root node are set to zero indicating that both behaviors are flagged for execution at simultaneous following the triggering of the root node; however, the ultimate execution starting points are determined by the magnitude and direction of the vector v_1.

The execution sequencing of the two actions is determined by the direction of the trigger vector connecting the two components. The random magnitude initialization of v_1 results in an even distribution of configurations where a percentage p of individuals have the node r_1 trigger r_2, while $1-p$ of individuals have the trigger relationship reversed. If a large enough population is utilized, the value p approaches 0.5 allowing the system to evaluate both trigger scenarios equally.

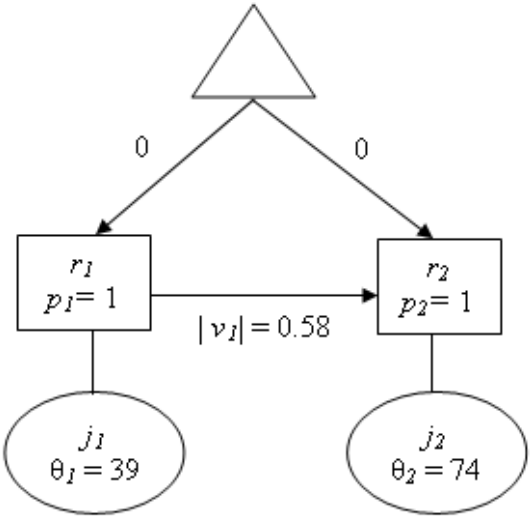


Figure 4.14: Evolved trigger network for robotic arm problem.

The evolved trigger network is demonstrated in Figure 4.14. The process utilized a fitness function that evaluates performance based on the final proximity of the robotic arm to the target. The fitness value f_i for individual i was calculated using the following formula:

$$f_i = 100 - d^3$$

where d represents the final distance from the target. The arm evolution utilized 24 individuals over 100 generations. A crossover $\alpha = 0.1$ and a mutation probability p_m=0.01 were utilized in the evolutionary process.

We notice that ultimately joint j_1 starts the execution of the entire behavior by moving towards the target. The magnitude of vector v_1 evolves to a value of 0.58; hence, the motion of joint j_2 is triggered after j_1 has completed 58% of its execution cycle. An execution duration of 2 seconds was utilized for both actions, so joint j_2 starts its motion 1.16 seconds following joint j_1. The final evolved angle for j_1 and j_2 are 39° and 74° respectively. The final evolved position of the robotic arm is shown in Figure 4.15. The

evolution progression is shown in Figure 4.16 where we notice a very rapid improvement in overall fitness over the first ten generations, then a steady gradual increase in average fitness is demonstrated by the system until the maximum generation number is reached.

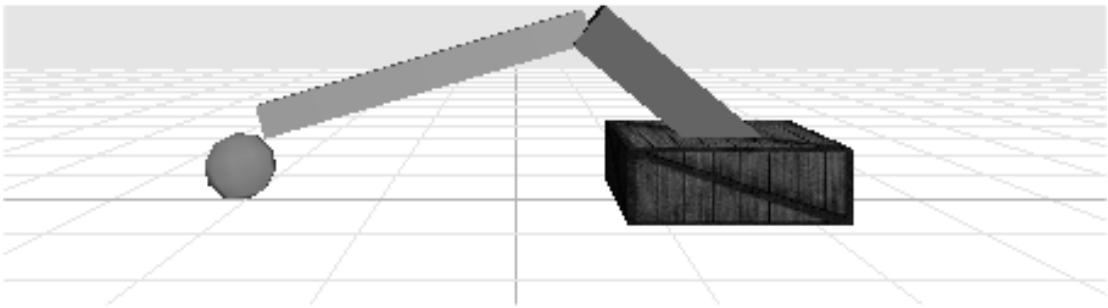


Figure 4.15: Evolved robotic arm controller.

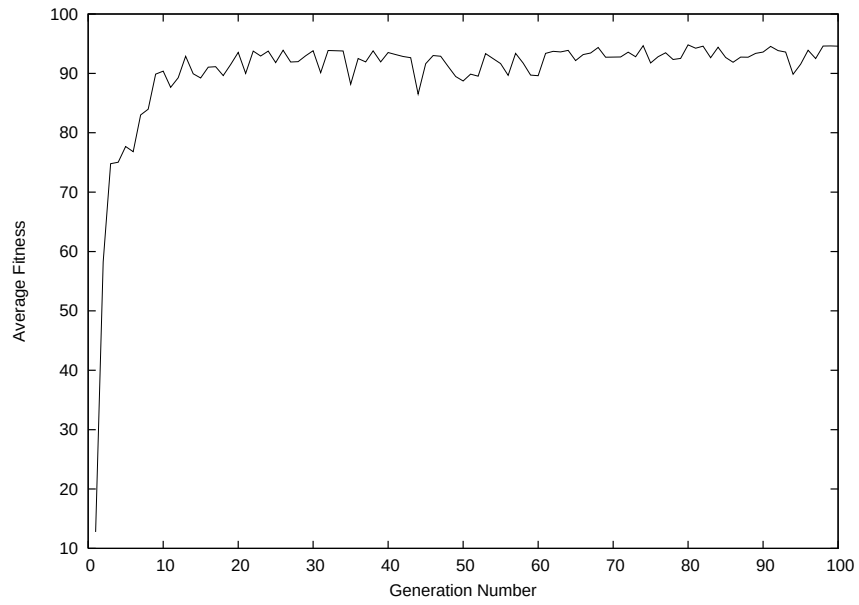


Figure 4.16: Evolution progression of the robotic arm problem.

Although the most optimum arm angles for the presented problem could be numerical calculated using inverse kinematics, the evolutionary methodologies utilized proves useful in other problems where a numerical solution is not reachable, or could be reached, yet not within a reasonable time frame.

4.5.2 Evolution of Actions

Trigger network actions are evolved towards more optimum configurations in an effort to achieve higher fitness. The type of evolution utilized depends on the specific type of action being evolved. Direct joint control actions are evolved based on their associated joint angle as well as the urgency of action execution. For any joint within the articulated structure, each degree of freedom is considered an independent action with its own set of parameters. Different angles and urgencies may be associated with different axis of the same joint.

Joint control strategy evolution is based upon finding the most optimum control algorithm based upon the functional dependency of control parameters. A functional relationship is determined between a desired output signal and an input signal which represents the state of the joint or one of its characteristic functions. Given the input value x, the evolutionary process aims to select the most successful control function $f(x)$ selected from the list presented in Table 4.1. The function selected determines how the input value maps to an executable action that helps the agent achieve its desired goals. The global constant parameter F is scaled by the evolvable scaling parameter s in order to achieve the final form of the function.

1.	$f(x) = s \cdot F \cdot x$
2.	$f(x) = s \cdot F \cdot x^2$
3.	$f(x) = s \cdot F \cdot x^3$
4.	$f(x) = s \cdot F \cdot \sqrt{x}$
5.	$f(x) = s \cdot F \cdot \sqrt[3]{x}$

Table 4.1: Functional dependency list for the evolution of joint control strategies.

The list of functions presented could be expanded to accommodate special types of relationships between sensory inputs and associated actions. In addition, the functional dependency could involve multiple input values combined to produce a more complex mapping strategy.

If a joint is being manipulated through a PID controller, then the evolutionary process aims to optimize the PID gains in an effort to achieve the best results. The three gain parameters are represented by K_p for the proportional gain, K_i for the integral gain, and K_d for the differential gain. Manipulating the PID parameters using the genetic process allows the system to rapidly reach the most optimum control strategy in relation to the fitness function governing the evolution of individuals.

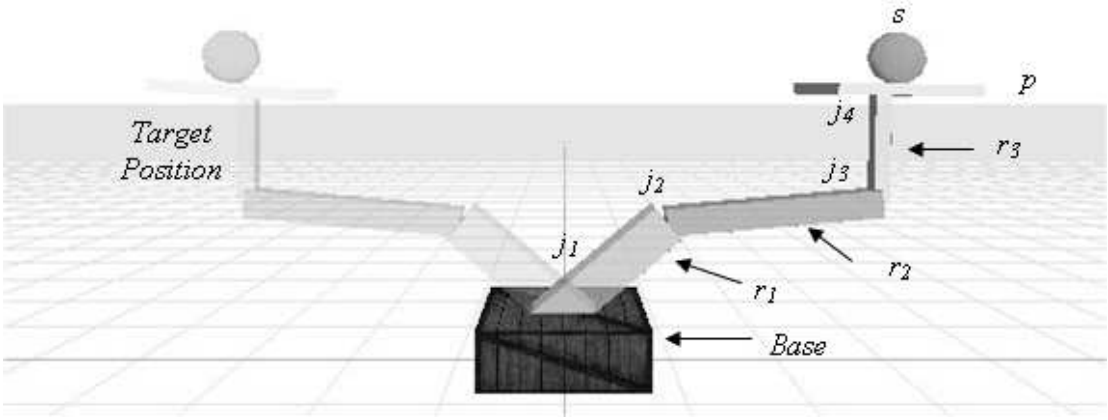


Figure 4.17: Sphere position control using an evolved PID controller.

A problem demonstrating PID controller evolution is presented in Figure 4.17. The presented articulated structure consists of a base and the three robotic arms: r_1, r_2, and r_3. The three arms are connected by three hinge joints: j_1, j_2, and j_3. The plate p is connected to the top most arm, r_3, utilizing the universal joint j_4 which has two degrees of freedom. The sphere s is placed in the center of p, and the goal of the articulated system is to keep the sphere situated on the plate as it moves from its initial position to its target position at the opposite side of the base. As the structure starts moving, the sphere immediately starts to roll out of place and eventually drops off the plate. The system aims to fulfill two main goals:

- To maintain the sphere s situated on the plate by changing the plate angle relative to its supporting arm. The rotation of the plate must counter the rolling of the sphere by reducing its velocity and eventually moving it back to a stable position on the plate.

- To minimize the average distance of s from the center of the plate over all time steps through out the execution of the entire behavior.

In order to achieve the behavioral goals, we utilize three different actions:

- Action a_1 represents an evolved control strategy action responsible for keeping arm r_3 always in an upright position as the robotic structure moves through its different positions. This task is accomplished by continuously monitoring the deviation of the vertical arm axis from the global vertical axis and changing the angle of j_3 accordingly. Since a direct mapping exist between the deviation angle and the desired

angle adjustment, the required change in the angle of j_3, δ, is equal to the negative of deviation angle θ.

- Action a_2 represents an evolved PID controller acting on joint j_4 and is responsible for keeping s in close proximity to the surface center of p. This is accomplished by utilizing the distance x, which represents the distance between the center of s and the surface center of p as the input to the PID controller. The three gain components k_p, k_i, and k_d are evolved in order to achieve the best results. The output of the PID controller represents adjustment signals that apply change to the angle of j_4.

- Action a_3 also represents an evolved PID controller acting on joint j_4, yet a_3 is responsible for maintaining the velocity of s as close to zero as possible. Although the rate of change in x is taken into account as a_2 calculates its output signals, a separate PID controller with an aim to reduce the velocity gives the system a faster response time helping maintain s situated on p at all times.

The three control actions utilized represent continuous actions whose duration spans the entire life span of the behavior. The three actions are triggered by the network root node, then they proceeds to execute indefinitely making constant adjustments until the behavior terminates. Figure 4.18 illustrates the independence of all three actions as no trigger vectors are present among them, so each action executes locally without triggering any other parts of the system.

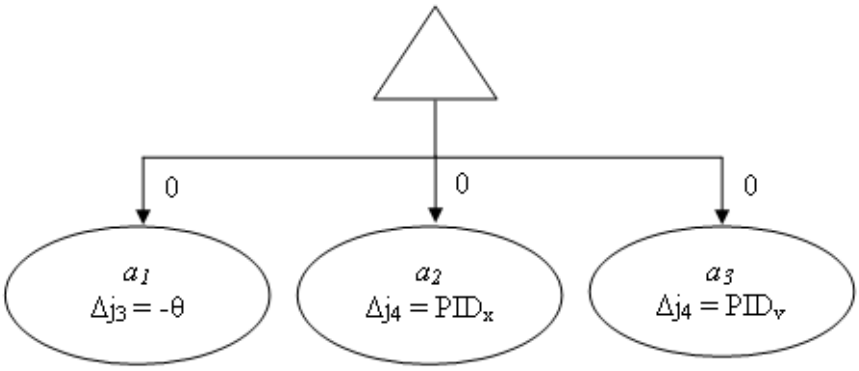


Figure 4.18: Trigger network for the sphere position control problem.

The fitness function utilized is based on the aim to maintain the placement of s on p through out the entire span of motion. The function is also based on the total accumulated distance of the sphere center at time t, $s_c(t)$, from the surface center of the plate at time t,

$p_c(t)$, over all time steps. The fitness value for each individual is inversely proportional to the total accumulated distance d_{total} defined as:

$$d_{total} = \sum_{t=1}^{t_{max}} |p_c(t) - s_c(t)|$$

where t_{max} represents the maximum time step count for the behavior execution. The evolutionary process utilized 24 individuals over 100 generations. A crossover $\alpha = 0.1$ and a mutation probability p_m=0.01 were utilized. The final values of the PID gain parameters for both PID actions are shown in Table 4.2.

	PID_x	PID_v
k_p	3.97	3.71
k_i	1.41	3.60
k_d	-4.11	-2.38

Table 4.2: Final evolved PID gain parameters.

The evolved PID controllers manage to maintain s situated on p for the complete movement cycle. In addition, the control strategy helps maintain the sphere as close to the center of the plate as possible to reduce the risk of falling off. The average distance of s from the surface center of p over all time steps is 0.26 units, compared to a maximum possible distance of 1.25 units which represent half the width of the plate. The evolution progression of the presented problem over 50 generations of evolution is shown in Figure 4.19.

4.6 Guiding the Genetic Process

The guided genetic evolution approach allows the designer to decrease the size of the problem search space by specifying boundaries that govern the evolution of actions and trigger vectors. The guiding mechanisms provides the evolutionary system with a general loose strategy for behavior execution. This is accomplished by supplying the system with bounding criteria for the evolution of the representative trigger network. Genetic guidance may be supplied by specifying any of the following criteria:

- General trigger network layout
- Action nodes within each subnetwork
- Ranges for joint angles

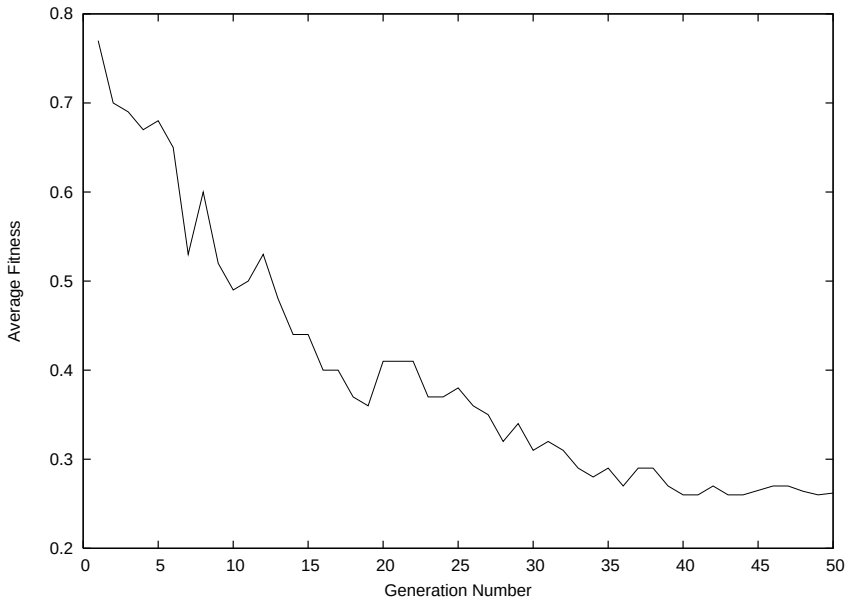


Figure 4.19: Evolution progression of the robotic arm sphere balancing problem.

- Ranges for execution urgency
- Ranges for PID gain parameters
- Ranges for execution priorities
- Trigger vector connectivity for actions
- Trigger vector connectivity for subnetworks
- Trigger vector sign (direction)
- Ranges for trigger vector magnitudes

The following sections describe the guidance methods in more detail. Some comparative studies are also provided to compare solution convergence time and accuracy between guided and unguided trigger networks.

4.6.1 Trigger Network Layout

An unguided trigger network must utilize a large enough layout to accommodate possible structural changes and required subnetwork count. If guidance is not utilized, the network layout must contain a large enough number of initial subnetworks to eventually

represent the different behaviors to be executed by the agent. All subnetworks are initially connected by trigger vectors representing possible relationships between behavior executions. Several significant issues are present in the initialization of unguided trigger networks:

- The designer has to make an estimate of the maximum possible number of behaviors required for the agent to achieve its goals. This may not be accomplished without a thorough analysis of the possible execution scenarios for agent behaviors. Once such analysis has been performed, the insight attained may be utilized in a more productive manner than the simple determination of maximum possible behavior count and network hierarchy levels.

- Several layers may be present in the network hierarchy. Utilizing a large number of behaviors at each level may cause the size of the unguided network to grow exponentially drastically reducing the possibility for accurate convergence of the network and significantly increasing the evolution duration.

- A complex articulated structure would contain a large number of joints with possibly multiple degrees of freedom for each joint. Including all possible actions [4] in each subnetwork would create an exponentially large variable count to be optimized for each behavior within the network.

Guiding the trigger network into faster and more accurate convergence requires the inclusions of constraints that limit the problem search space by minimizing the trigger network size. Such constraints would relate to subnetwork structuring and dependency, action inclusion, parameter ranges, and trigger vector directions and magnitudes.

4.6.2 Subnetwork Structuring

Subnetworks represent primitive behaviors that are used as building blocks within the trigger network. In order to limit the network size, guidance is provided by loosely defining the basic behavioral structure essential for the achievement of the goals. In order to achieve the most optimum control strategy, the following structuring steps are utilized:

- The execution path that connects the beginning of the agent's life cycle with its associated goal achievement point is divided into multiple segments where the end point of each segment represents a desired known state.

[4] An action is defined as a control strategy relating to a particular axis of a particular joint. A joint with multiple degrees of freedom would be represented by multiple actions

- Each of the path segments is represented by a subnetwork transporting the agent from one known state to the next.
- Each subnetwork is structured loosely by specifying the possible actions that would contribute to the successful achievement of the desired state.
- Subnetworks are connected via trigger vectors representing the relationships that govern the triggering of primitive behaviors.

Once the core subnetwork structure is in place, the evolutionary process optimizes the control strategy by altering the internal subnetwork components as well as the vector connectivity between subnetworks.

If we examine an example of a four-legged robotic agent with the goal of achieving forward mobility, we notice some essential segmentations that exist on the path to the successful achievement of the goal. The segments represent intermediate goals the agent must achieve in order to reach successful forward stepping actions. Figure 4.20 shows the four-legged robot in the simulation environment.

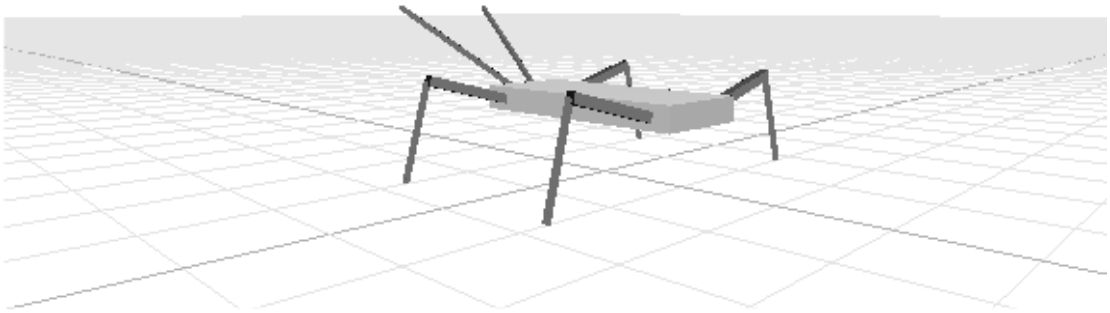


Figure 4.20: Four-legged robot placed in the simulation environment.

In order to simplify the control problem, we structure the four-legged robot problem utilizing only 4 active joints. The four joints are used to connect the 4 legs to the main torso, while the 4 feet are connected to the legs using passive joints that lock the feet at a specific angle. Each of the active joints utilize two degrees of freedom in order to be able to move the leg upward and downward as well as forward and backward. Each movement along an axis of a joint is considered a separate action that is independently evolvable. Hence, we utilize a total of eight actions for controlling the robot. Table 4.6 lists the action classifications for the four-legged articulated structure.

ID	Joint	Axis	Description
$a1$	$j1$	1	Up/down motion for front right joint
$a2$	$j2$	1	Up/down motion for front left joint
$a3$	$j3$	1	Up/down motion for rear right joint
$a4$	$j4$	1	Up/down motion for rear left joint
$a5$	$j1$	2	Front/back motion for front right joint
$a6$	$j2$	2	Front/back motion for front left joint
$a7$	$j3$	2	Front/back motion for rear right joint
$a8$	$j4$	2	Front/back motion for rear left joint

Table 4.3: Action classification for the four-legged robot problem.

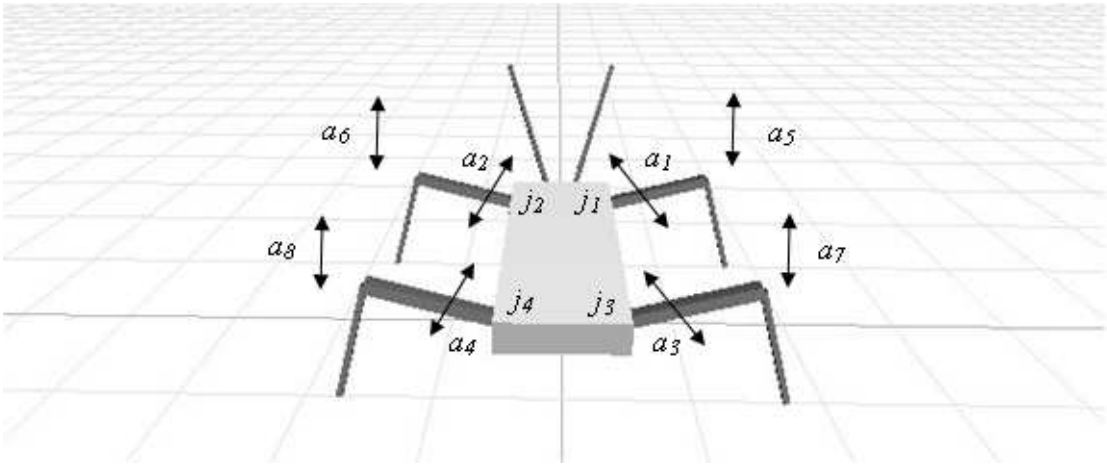


Figure 4.21: Action classifications for the four legged articulated structure.

Figure 4.21 shows the location for each of the joints being utilized as well as the associated action tied to each of the joint axis. In order to achieve successful forward mobility, the joints must be controlled using an appropriate strategy. If we do not utilize any guidance in the robot motion planning, then we must estimate the approximate number of intermediate behaviors the robot must step through in order to achieve its primary goal. For each of the behaviors decided upon, we must include all eight actions as possible contributors to the behavior. In addition, we must include trigger vectors between each action and the other seven actions within each behavior to represent possible execution dependencies. Similarly, each behavior, represented by a subnetwork, must be connected to all other behaviors to represent possible dependencies among the behaviors themselves. Considering the unguided behavior b_1 and its eight associated actions a_1,...,a_8, Figure 4.22 shows the structuring of the b_1 subnetwork.

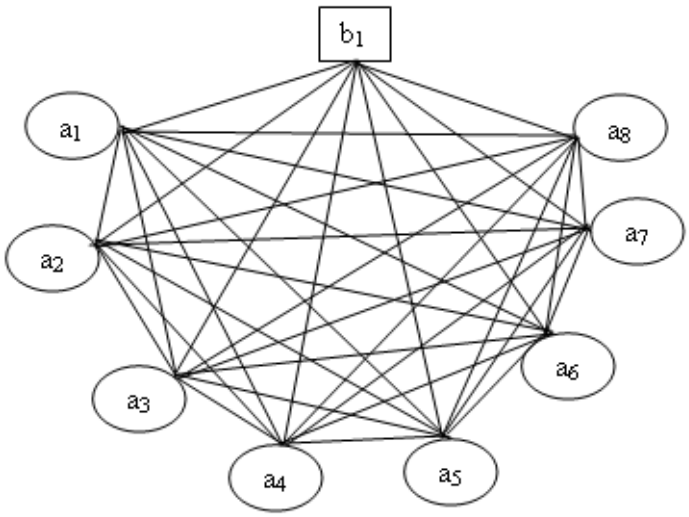


Figure 4.22: Single behavior in unguided trigger network including all associated actions and action dependencies.

For each of the behaviors included in the trigger network, evolvable parameters must be utilized for each of the utilized actions and trigger vectors. For a single behavior with eight direct control actions, we require a single parameter for the desired angle, and another parameter for the associated urgency. Hence, we require a total of 16 action variables for the subnetwork. In addition, since each action node is connected to each other node as well as the representative relay node via a trigger vector, we have a completely connected mesh network of 9 nodes. For a network of n nodes, the number of required connections is $n(n-1)/2$, hence, we require 36 trigger vectors within the network. Two variable are required to hold the state of for each trigger vector: one variable is utilized to represent the connection direction and magnitude of the vector, and another is used to hold the activation state. In addition to the internal sub network connections, additional connections are required to represent dependency relationships between subnetworks. Figure 4.23 shows the interconnections between subnetworks utilized in the four-legged robot control problem.

Table 4.4 details the different variable counts for the genetic optimization problem. We notice that without the use of genetic guidance, the number of variables to be optimized in order to achieve the desired goal is very large. In order to achieve successful evolution of a robotic controller for the presented problem, a significantly large population size must be utilized over a large number of generations. The complexity of the structure being evolved exponentially increases the chances for achieving convergence to

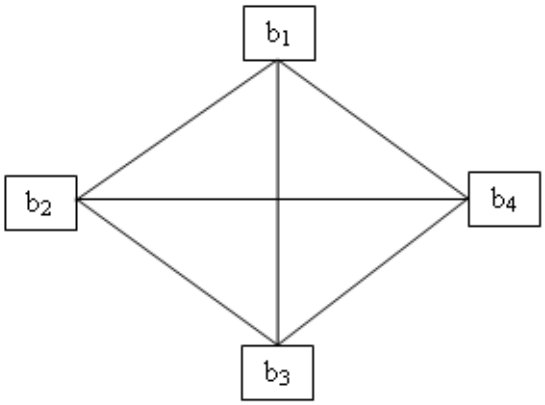


Figure 4.23: Subnetwork connectivity among four behaviors in the unguided configuration of the trigger network.

an suboptimum control strategy.

Actions/Subnet	8
Variables/Action	2
Vectors/Subnet	36
Variables/Vector	2
Total Variables/Subnet	88
Subnets	4
Subnet Connections	6
Total Network Variables	358

Table 4.4: Variable counts for the unguided four-legged robot problem.

Structural guidance may be utilized in the presented problem to minimize the number of variables being optimized. This is achieved by applying several strategies:

- Approximating the core behaviors necessary for achieving the goal.
- Choosing contributing actions for each behavior and using only those actions in the structuring of the subnetwork.
- Minimizing the number of vector connections within each subnetwork by utilizing only vector connections that have a high probability of being utilized.
- Minimizing the number of vector connections among behaviors depending on the possible dependencies that govern the achievement of intermediate goals.

In order to provide layout guidance for the trigger network, we formulate a possible configuration that would yield successful forward mobility. We base the trigger network on two subnetworks each representing a single behavior that utilizes all of the four joints. Since each joint moves on two axis, each subnetwork contains 8 action nodes all to be triggered simultaneously. The aim is to have the evolutionary process locate an appropriate configuration that propels the robot forward. The most intuitive mobility scenario is achieved if the genetic process reaches the general sequence of behaviors listed in the following table:

Behavior 1	Joints j_1 and j_4 manage to move their associated legs up and forward, while joints j_2 and j_3 manage to move their associated legs down and back.
Behavior 2	Joints j_2 and j_3 manage to move their associated legs up and forward, while joints j_1 and j_4 manage to move their associated legs down and back.

The reverse of the given strategy, where the action couples are swapped, should also yield the same results. Although the given structure is the most intuitive configuration for four-legged mobility, the genetic process probabilistically may find other configurations that also work as well. Table 4.5 shows the variable counts after the application of network modifications.

Actions/Subnet	8
Vectors/Subnet	8
Total Variables/Subnet	32
Subnets	2
Subnet Connections	2
Total Network Variables	36

Table 4.5: Variable counts for modified trigger network.

For the specific given problem, the following modifications have been implemented to reduce the complexity of the trigger network:

- Number of subnetworks was reduced to only two subnets representing two core behaviors,
- All trigger vector connections between actions were removed in both subnetworks,
- All trigger vector magnitudes in the network were fixed,

- Only a single trigger vector connects the first subnetwork to the second using a single prominent action of the first subnetwork,
- Only a single trigger vector connects the second subnetwork to the reset node using a single prominent action of the second subnetwork.

Figure 4.24 shows the structure of the guided trigger network. The network consists of the two subnetworks b_1 and b_2.

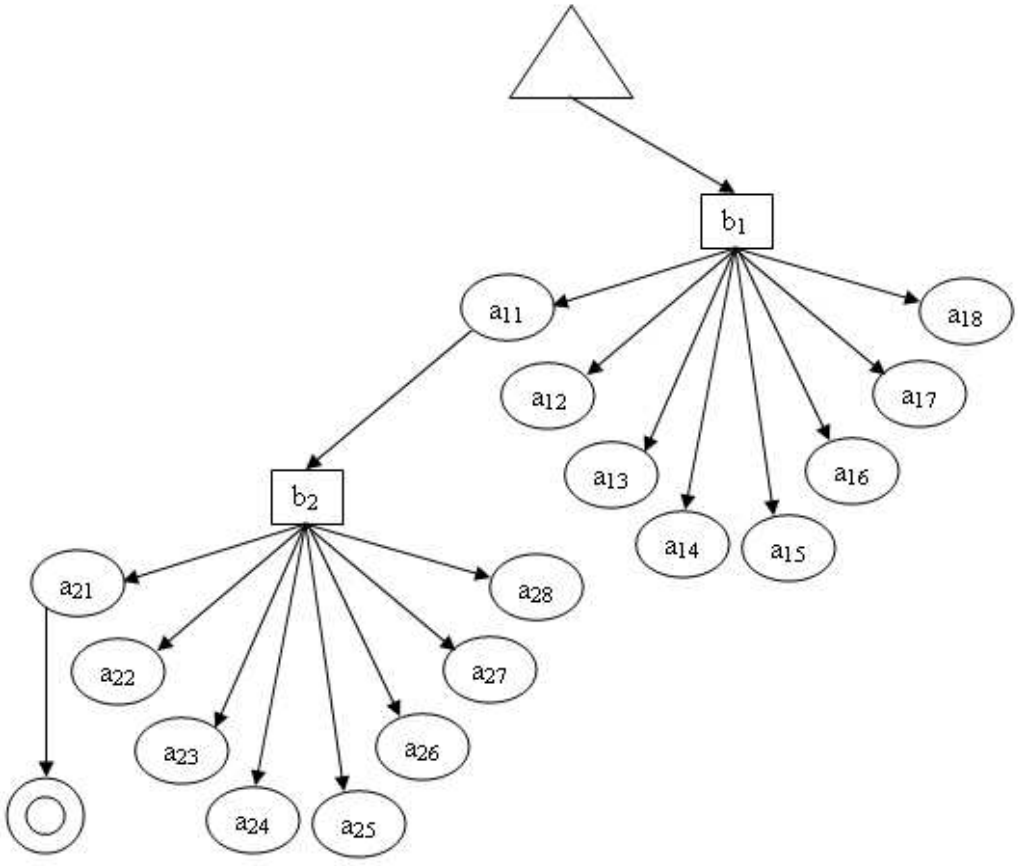


Figure 4.24: Guided trigger network for the evolution of the four-legged robot.

The relay node b_1 is triggered directly by the root of the network making it the first node to be executed. The node b_1 in turn triggers its associated eight actions represented by the unique IDs $a_{11}...a_{18}$. The actions directly correspond to the action classifications listed in Table 4.6.

Actions of the subnetwork b_1 are executed immediately following the execution of their associated relay node since all trigger vectors connecting the relay node to the associated actions are fixed to a value of zero.

Action a_{11} is selected as the prominent action of b_1. The subnetwork b_2 is triggered by a_1 with a trigger vector magnitude of 1, which means that b_2 is triggered immediately following the termination of a_1.

The execution of b_2 prompts the immediate execution of actions $a_{21}...a_{28}$, which also directly correspond to the action classifications listed in Table 4.6. The actions are triggered immediately as the internal trigger vectors of b_2 are also fixed to a value of zero.

Action a_{21} is selected as the prominent action of b_2. Once a_{21} terminates its execution, it triggers a reset node which resets the state of the entire network then triggers b_1 to restart the execution cycle.

The evolution of the four-legged robot utilizes 50 individuals being evolved over 100 generations. The genetic process utilizes a roulette wheel selection method, crossover probability $p_c = 0.01$, crossover alpha $\alpha = 0.1$, and mutation probability $p_m = 0.01$. The evolution progression is shown in Figure 4.25.

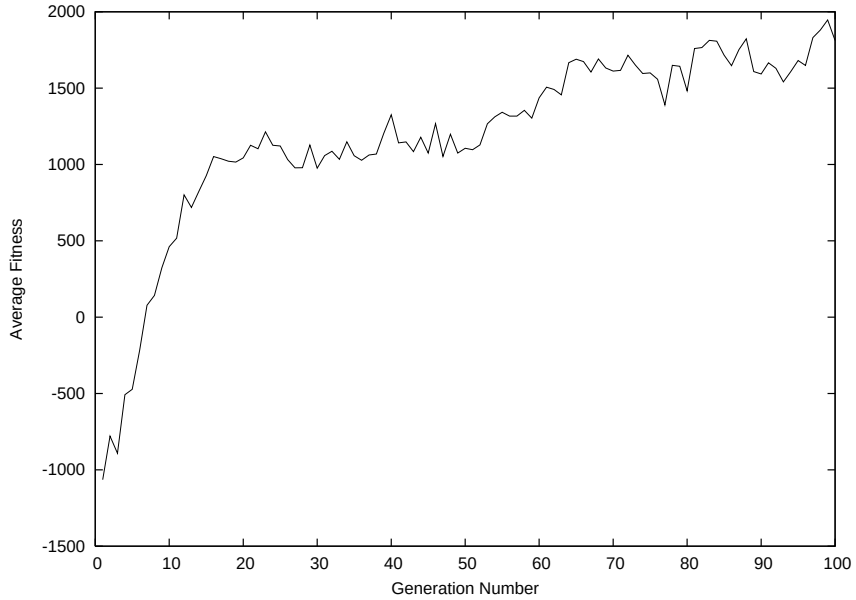


Figure 4.25: Evolution progression of the four-legged robot over 100 generations.

The associated fitness function utilized is structured to promote forward mobility in a straight line without rotation. In order to evaluate the performance of each individual, the following strategy was followed:

- A reward is assigned based on the distance traveled by the robot within a 10 second period. The reward assigned is proportionate to the distance traversed along a straight line originating at the origin.
- A penalty is applied based on the deviation of the robot from the desired straight path. The penalty assigned is proportionate to the perpendicular distance between the final robot position and the desired path.
- A penalty is applied based on the rotation of the robot. The penalty assigned is proportionate to the rotation angle about the z-axis after the 10-second run.

The specific fitness function f_i which represents the fitness of individual i is defined as:

$$f_i = (x_i \cdot 1000) - (y_i \cdot 1000) - (\theta_i \cdot 2000) \tag{4.8}$$

where x_i is the distance travelled along the target path, y_i is the final perpendicular distance between the robot position and the target path, and θ_i is the rotation along the z-axis that has occurred since the triggering of the network.

As predicted, the four-legged robot does not conform to the most intuitive method for forward mobility. The robot follows a systematic hopping motion that ultimately allows it to achieve forward mobility (Figure 4.26). Although very limited guidance has been applied to the network structuring, the genetic process was able to evolve a method for forward mobility based on the criteria specified in the fitness function.

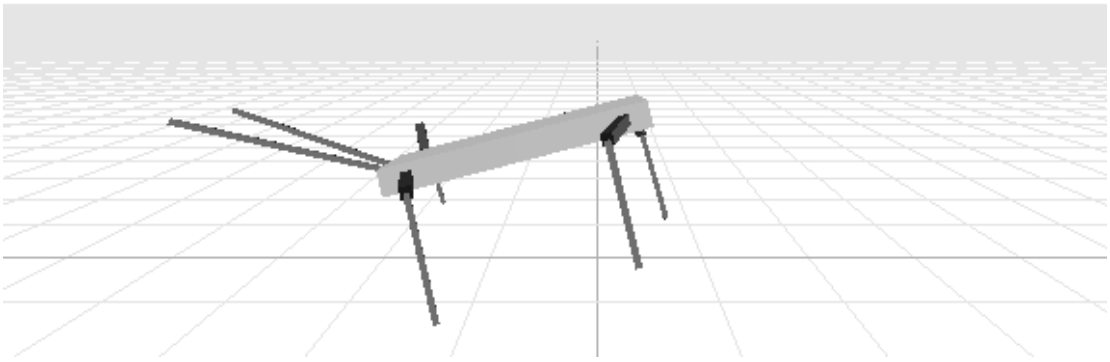


Figure 4.26: Four-legged robot forward mobility utilizing a hopping behavior.

4.6.3 Guidance of Actions

In addition to guiding the trigger network layout, the network actions could also be guided towards specific configurations that would allow faster and more accurate convergence.

Guidance of actions aim to reduce the problem search space by reducing the size of the search interval for each of the action parameters. Any of the action types utilized may be guided by providing a level of parameter range focusing. The three action types may be guided as follows:

- *Direct control* actions may be guided by reducing the search interval for both the angle and urgency parameters. Without guidance, the action would utilize the joint low and high stop points as the boundaries for its search. However, by specifying an alternate range that is of a smaller size than the full range of motion for the joint, better results may be achieved. The urgency parameter may also be focused by utilizing insight into the nature of the problem and using an urgency range that would be most appropriate for achieving the desired results.

- *Control strategy* action evolution aims to find a functional dependency between an input value and a desired output signal. By utilizing only a subset of the available functions, we may cut down the evolution time. For example, if knowledge of the problem leads to a determination that the output signal must exceed the input value, then only functions that result in an output that exceeds the input value may be used.

- *PID control* actions evolve their control parameters in order to achieve the most optimum control strategy based on an input value. The parameters k_p, k_i, and k_d could be represented by any value, and reducing the possible interval for each of the parameters may drastically increase the probability of optimum convergence.

For the four-legged robot problem, we utilize further guidance of direct control parameters in an effort to achieve quicker and better results. The following action guidance strategy was followed:

In the first behavior identified by b_1, the intervals for possible leg angles where reduced in size so that the legs connected to joints j_1 and j_4 were limited to moving upward and forward throughout the behavior. Similarly, the legs connected to joints j_2 and j_3 were limited to moving downward and backward during the behavior.

In the second behavior identified by b_2, the legs connected to joints j_1 and j_4 were limited to moving downward and backward, while the legs connected to joints j_2 and j_3 were limited to moving upward and forward.

The urgency of action execution was limited so that the up/down movements of the legs executes in half the time (on average) as the forward/backward movement. This strategy allows for the forward stepping to be achieve by having the legs touch the ground and then continue to pull backwards propelling the robot forward.

The same genetic parameters were utilized for the evolution of the agent after the guidance modifications have been made. Also, the same fitness function were used to measure the results. The evolution progression of the guided robot is shown in Figure 4.27. If we compare the evolution results to that of the unguided evolution, we notice the same progressive improvement in average fitness over the entire evolutionary process. However, we notice that the final average fitness achieved after 100 generations of evolution has almost doubled due to the evolutionary guidance introduced to the actions executed by the agent. The final average fitness of the entire population increased from 1812 to 3600 due to the network modifications implemented, which represents a 98% increase.

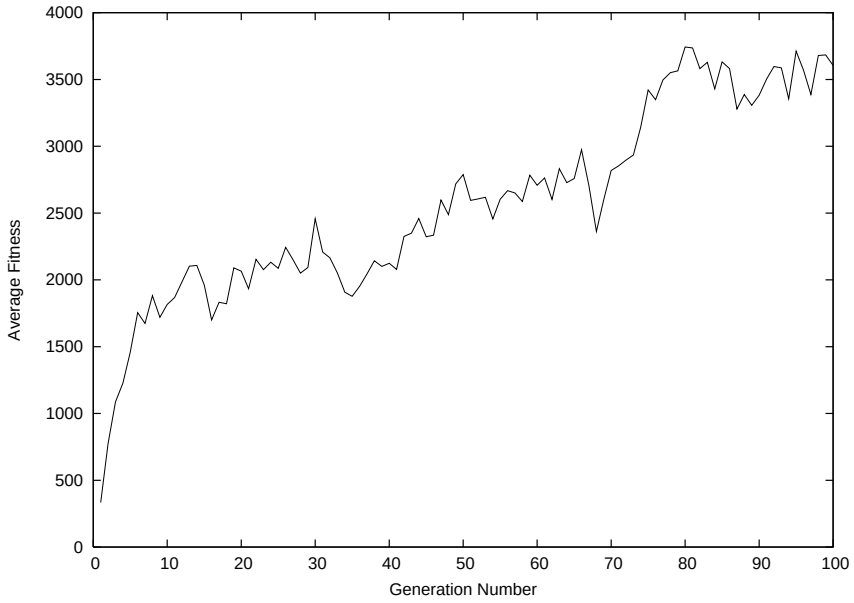


Figure 4.27: Guided evolution progression of the four-legged robot over 100 generations.

4.6.4 Guidance of Trigger Vectors

Trigger vectors are utilized as means for determining the sequencing of event execution within the trigger network. An unguided trigger network would include trigger vectors connecting all network nodes representing possible execution relationships. Guidance

of trigger vector allocation is achieved by utilizing knowledge of the problem as well as possible solution scenarios to limit the trigger vector allocation by applying specific vector configurations.

In the four-legged robot problem, knowledge of the problem allows us to determine that a possible mobility scenario would consist of a two-phase approach where the first phase triggers the second. Each phase consists of multiple actions triggered simultaneously by the subnetwork relay node without any trigger vectors connecting the actions to each other. By guiding the trigger vector allocation in that manner, we have significantly reduced the complexity of the problem by reducing the vector count within each subnetwork. However, further problem guidance could be achieved by the further partitioning of the problem into smaller core behaviors supported by trigger relationships.

We expand the problem guidance to encapsulate 16 different actions. Each leg is to have the following four independent actions associated with it:

- Upward movement
- Downward movement
- Forward movement
- Backward movement

The following expanded action classifications are to be used:
The desired sequencing to be achieved is as follows:

1. Legs 1 and 4 move upwards and forwards while legs 2 and 3 move backwards,
2. Legs 1 and 4 both lower their legs to make contact with the walking surface,
3. Legs 2 and 3 move upwards and forwards while legs 1 and 4 move backwards,
4. Legs 2 and 3 both lower their legs to make contact with the walking surface,
5. Repeat cycle.

Four behaviors are needed for achieving the action combinations described above. The first behavior identified by b_1 consists of the six described actions. Figure 4.28 shows the structure of the b_1 subnetwork. The six actions contained in b_1 are all executed simultaneously at the time the subnetwork is executed. Each of the actions in b_1 is given the

ID	Joint	Axis	Description
$a1$	$j1$	1	Up motion for front right joint
$a2$	$j1$	1	Down motion for front right joint
$a3$	$j1$	2	Front motion for front right joint
$a4$	$j1$	2	Back motion for front right joint
$a5$	$j2$	1	Up motion for front left joint
$a6$	$j2$	1	Down motion for front left joint
$a7$	$j2$	2	Front motion for front left joint
$a8$	$j2$	2	Back motion for front left joint
$a9$	$j3$	1	Up motion for rear right joint
$a10$	$j3$	1	Down motion for rear right joint
$a11$	$j3$	2	Front motion for rear right joint
$a12$	$j3$	2	Back motion for rear right joint
$a13$	$j4$	1	Up motion for rear left joint
$a14$	$j4$	1	Down motion for rear left joint
$a15$	$j4$	2	Front motion for rear left joint
$a16$	$j4$	2	Back motion for rear left joint

Table 4.6: Expanded action classification for the four-legged robot problem.

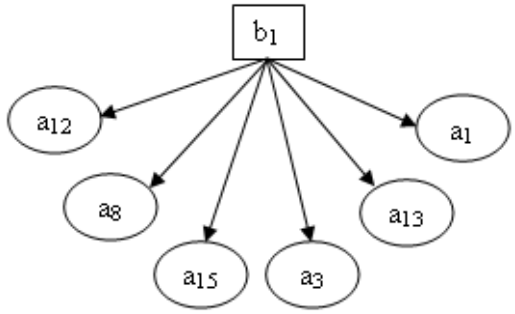


Figure 4.28: Subnetwork b_1 representing first agent behavior.

same minimum execution urgency of 0.5 and a maximum execution urgency of 2.0. The genetic process is to determine the best urgency value for each of the joint movements.

The second agent behavior identified by b_2 consists of two actions that lower legs 1 and 4 to make contact with the ground in preparation for propelling the robot forward. Both actions are also connected to their associated relay node via zero-magnitude trigger vectors, and they are given the same urgency bounds as the actions of the first behavior with a minimum of 0.5 and a maximum of 2.0. Behavior b_2 is represented by Figure 4.29

The third behavior is quite similar to the first except for the fact that it reversed the roles of the two leg groups. Legs 2 and 3 are to move upwards and forwards while legs 1

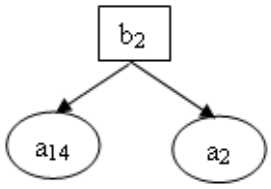


Figure 4.29: Subnetwork b_2 representing the second agent behavior.

and 4 are to move backwards. The backward action of legs 1 and 4 should be responsible for propelling the robot forward allowing it to achieve a stepping behavior. The same trigger vector configuration is utilized for this behavior as for the previous two. Behavior b_3 is represented by Figure 4.30

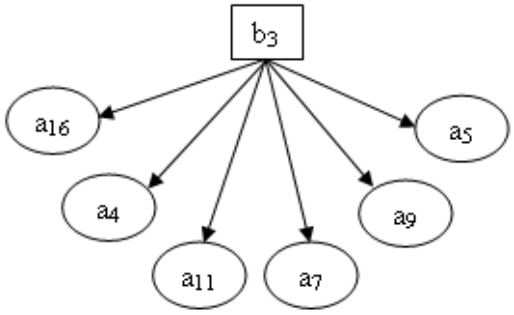


Figure 4.30: Subnetwork b_3 representing the third agent behavior.

The fourth and final behavior is similar to behavior b_2 except for the fact that it lowers legs 2 and 3 to the walking surface instead of legs 1 and 4. Zero-magnitude trigger vectors are also utilized to trigger the actions within the subnetwork.

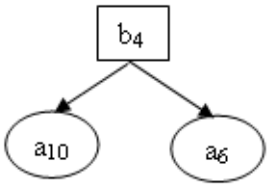


Figure 4.31: Subnetwork b_4 representing the fourth agent behavior.

The sequencing of behavior execution is structured based on immediate transitions from one behavior to the next. A dominant action is chosen in each subnetwork for the purposes of triggering the next behavior once its own execution has terminated. The following dominant actions were chosen for each of the behaviors:

- Action a_3 in b_1 triggers b_2.
- Action a_{14} in b_2 triggers b_3.
- Action a_7 in b_3 triggers b_4.
- Action a_6 in b_4 triggers the reset node.

The complete trigger network for the expanded problem is shown in Figure 4.32. The network root triggers subnetwork b_1 starting the execution of the first behavior. Each subnetwork triggers the next until subnetwork b_4 triggers a reset node which resets all nodes to their original state and starts the network execution once more. The evolutionary process utilized the same genetic parameters as in the previous iterations of the problem. A roulette wheel selection method was utilized. A crossover probability $p_c = 0.01$, crossover alpha $\alpha = 0.1$, and mutation probability $p_m = 0.01$ were utilized in the genetic process. The evolution progression over 100 generations is shown in Figure 4.33. We notice that due to the added network guidance, the average fitness exceeds that of the previous network layout within the first 10 generations of evolution. The progress continues throughout the evolutionary process reaching a final average fitness of approximately 4800, which represents a further 33% increase in average fitness over the previous network layout.

4.7 Detailed Algorithm

In this section, we detail the algorithm utilized for the evolution of trigger networks for articulated robot control. Once the network has been structured based on knowledge of the problem as well as the systematic guidance utilized to reduce the problem complexity, the genetic evolution algorithm is used to evolve the most optimum control strategy in relation to the desired goals. The algorithm presented covers the evolution of all aspects of the trigger network, including action nodes, trigger vector directions, magnitudes and activations, as well as the general subnetwork connectivity.

The evolution algorithm utilized is structured as follows:

Evolution Algorithm
begin(1)
t := 0
initialize G_t based on network guidance

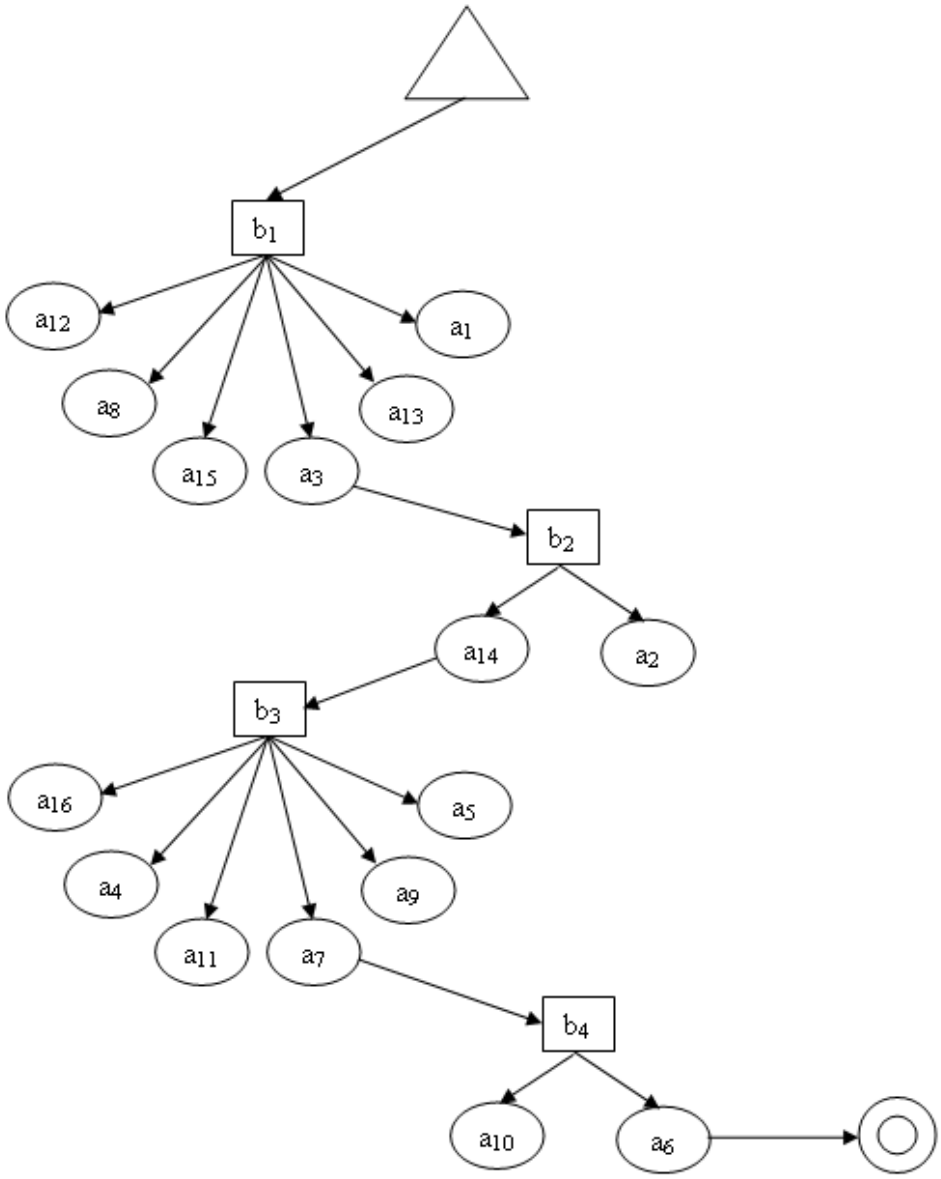


Figure 4.32: Trigger network for expanded four-legged robot problem showing the four subnetworks representing the four core agent behaviors.

evaluate fitness of individuals in G_t
While *Not termination-condition* **do**
begin(2)
$t = t + 1$
select G_t *from* G_{t-1}*:*

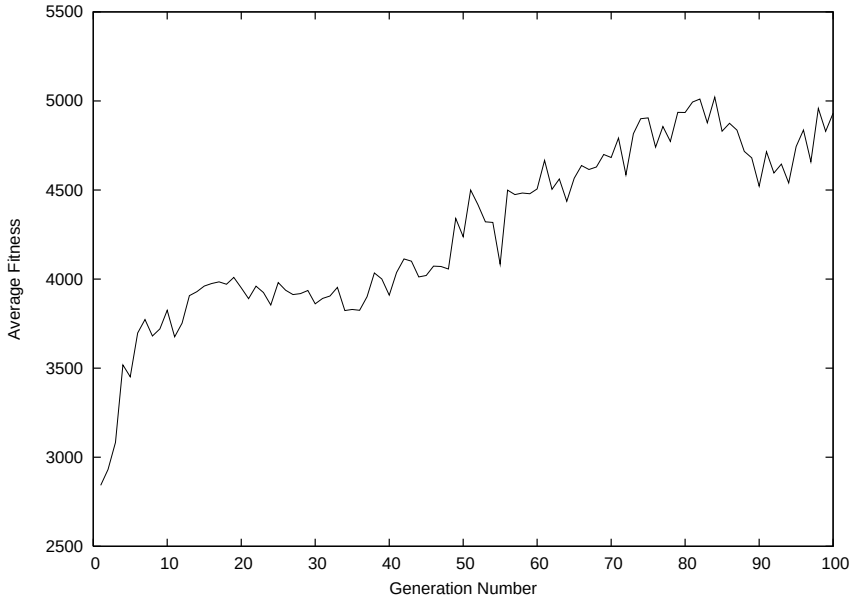


Figure 4.33: Evolution progression of the four-legged robot over 100 generations.

$$P(c_j) = \frac{\varphi(f(c_j))}{\sum_{k=1}^{m} \varphi(f(c_k))}$$

where $P(c_j)$ is the probability for selecting individual c_j, $f(c_j)$ is the fitness value for individual c_j, $f(c_j)$ is the fitness value for individual c_j, m is the number of individuals in the generation, and $\varphi : \mathbb{R} \rightarrow \mathbb{R}^+$ is a non-decreasing transformation used to shift the fitness values to $\mathbb{R}^+$.

crossover G_t with probability p_c:

$$\left[min(x_i^1, x_i^2) - I \cdot \left(\alpha_{\max} - \frac{g \cdot (\alpha_{\max} - \alpha_{\min})}{g_{\max}} \right), max(x_i^1, x_i^2) + I \cdot \left(\alpha_{\max} - \frac{g \cdot (\alpha_{\max} - \alpha_{\min})}{g_{\max}} \right) \right]$$

where p_c is the crossover probability, g_{max} is the maximum generation number, g is the current generation number, α_{min} and α_{max} are the minimum and maximum α values desired respectively, x_i^1 represents gene i for the first parent individual, x_i^2 represents gene i for the second parent individual, and

$$I = max(x_i^1, x_i^2) - min(x_i^1, x_i^2)$$

mutate G_t with probability p_m:

Randomly choose:

$$c_i = x_i + \Delta(t, b_i - x_i)$$
$$or$$
$$c_i = x_i - \Delta(t, x_i - a_i)$$

where

$$\Delta t(t,x) = x \cdot \left(1 - r^{\left(1 - \frac{g}{g\max}\right)^b}\right)$$

The random value r is chosen from the interval $[m_{min}, m_{max}]$, p_m is the mutation probability, and m_{min} and m_{max} define the bounds of the target domain for the specific gene.

evaluate fitness of individuals in G_t

end(2)

end(1)

The initialization, crossover and mutation algorithms presented above are used for all action nodes as well as trigger vectors within the network. In the case of action nodes, the algorithm is applied depending on the node subtype as each subtype holds a different type of structure to be evolved. Action nodes are evolved as follows:

- **Direct Control**
  - The joint angle for each active axis is initialized using a random value generated from the interval $[Angle_{min}, Angle_{max}]$.
  - The execution urgency for each active joint axis is initialized using a random value generated from the interval $[Urgency_{min}, Urgency_{max}]$.
  - Crossover is applied to both the angle and urgency values of two individuals using the algorithm presented.
  - Mutation is applied to both the angle and urgency values of a single individual using the algorithm presented.
- **Control Strategy**

  - The control function ID is randomly chosen from the interval $[f_{min}, f_{max}]$. Discretization is used to map the floating point value generated by the genetic process to an integer function ID.
  - The control scaling value s is initialized using a random value generated from the interval $[s_{min}, s_{max}]$.
  - Crossover is applied to both the function ID and scaling value of two individuals using the algorithm presented.
  - Mutation is applied to both the function ID and scaling value of a single individual using the algorithm presented.

- **PID Control**
  - The PID parameter kp is randomly chosen from the interval $[kp_{min}, kp_{max}]$.
  - The PID parameter ki is randomly chosen from the interval $[ki_{min}, ki_{max}]$.
  - The PID parameter kd is randomly chosen from the interval $[kd_{min}, kd_{max}]$.
  - Crossover is applied to the values kp, ki and kp of two individuals using the algorithm presented.
  - Mutation is applied to the values kp, ki and kp of a single individual using the algorithm presented.

The application of initialization, crossover and mutation operators to trigger vector parameters is quite similar to their application to action nodes. The following strategy is followed for trigger vectors:

- The trigger vector value is randomly chosen from the interval $[v_{min}, v_{max}]$, where v_{min} and v_{max} represent the minimum and maximum dependency values respectively. The sign of the vector value represents the direction of the vector, while the magnitude represents the execution dependency beteween the two nodes involved. Guidance is achieved by taking the desired direction and magnitude into account when choosing the interval bounds for the specific vector.

- The trigger vector activation value is chosen randomly to indicate if the vector will in fact trigger the execution of other vectors within the network or not. The activation value is chosen randomly from the interval $[0, 1]$. A value below 0.5 indicates that the vector is inactive, while a value greater than or equal to 0.5 indicates that it is active.

- Crossover is applied to the trigger vector value and activation of two individuals using the algorithm presented.
- Mutation is applied to the trigger vector value and activation of a single individual using the algorithm presented.

The guidance utilized with trigger networks represent a loose strategy for pointing the evolutionary process towards a certain desired configuration. However, the guidance is not meant to enforce rigid constraints that constrict the evolutionary process taking away the essence and benefits of the genetic methodologies. Several approaches may be used to allow the network to evolve to an area outside of the suggested guidance criteria:

- The crossover operation utilizes values from both parents to generate offspring. The crossover α value is used to expand the target range beyond the boundaries represented by the values extracted from the parents. The larger the α value, the more of an opportunity the process has to move outside of the guidance interval by gradually reaching one of the boundaries then crossing it reaching a different area of the search space.
- The mutation operation generates offspring by generating random values from a predetermined interval for each target gene. Guidance is provided through the selection of desired intervals from which to select the genes for the first generation of individuals. Future generations, however, may utilize a larger interval encompassing all feasible values for the gene as the basis for selecting the random values. This would allow the process to select gene values that contradict the initial guidance given to the system. If any of the selected values prove to produce higher fitness values, then those values will be chosen for reproduction, and they may eventually have prominent presence in the population.

4.8 Conclusion

The guided genetic evolution platform utilizes trigger networks as a representational vehicle for the robotic control problem. The framework achieves optimization of the control strategy allowing for the successful goal-based control of autonomous robotic agents in real-time. Trigger networks represent a connectionist model structured for the encapsulation of the articulated robotic control problem. The network structure allows for the modeling of robotic actions and behaviors as well as the relationships the govern the

achievement of control goals. The evolution of trigger networks utilizing customized genetic methodologies allows the system to achieve optimized control strategies by following a systematic learning policy geared towards the continuous and gradual enhancement of system performance.

The guided genetic evolution platform presents constructs for the guidance of the control problem genetic representation and evolution drastically reducing the complexity of the problem. The genetic guidance reduces the variable complexity associated with the optimization problem which allows for faster system convergence and reduces the chances for convergence to suboptimal results. The guidance mechanism also allows for the inclusion of *learning by example* methodologies without taking away from the essence of the genetic process.

Genetic guidance is achieved through the manipulation of genetic initialization to favor specific configurations relating to agent actions as well as governing relationships represented by trigger vectors. Trigger network evolution applies genetic selection gradually increasing the population average fitness over multiple generations. The genetic crossover α choice as well as the mutation interval maintains the network ability to deviate from the guidance strategy when proven to be beneficial for the evolutionary process.

The following chapters discuss the utilization of guided genetic evolution in several specific scenarios for the control of articulated robotic structures. For each of the problems presented, the guidance strategy as well as the results are discussed.

Chapter 5

TESTING THE EVOLUTION PLATFORM

5.1 Introduction

This chapter offers an implementation strategy to be followed in order to structure a control problem as an evolvable guided trigger network. The chapter also presents several control scenarios for articulated robotic control using guided genetic evolution. For each of the scenarios presented, the structuring of the control problem is discussed in detail as well as the trigger network problem representation and guidance. The implementations given aim to convey different perspectives of the control problem along with design methodologies for achieving successful control.

5.2 Implementation

This section describes a systematic implementation approach for structuring the control problem in order to utilize guided genetic evolution methods. The following steps list the implementation process:

1. The robotic agent is encoded as a trigger network which represents all the actions to be controlled by the evolved controller. Each of the a robot's joint axis is associated with a direct control action. The minimum and maximum angles for each of the actions are determined by the low and high stop points of the axis. Each possible force application within the system is also associated with a control strategy action or PID control action depending on the nature of the problem.

2. The subnetwork count representing the number of possible agent behaviors is chosen based on the total number of joint axis to be controlled. Each behavior maybe executed multiple times utilizing different parameters, so the subnetwork count may require expansion depending on the estimated repeat count of each behavior.

3. The trigger network is structured based on the number of actions and sub networks. Each subnetwork is structured with all possible actions as internal nodes, and trigger vectors are utilized to connect each action to each other action in both directions. The same trigger vector connectivity is utilized for subnetwork connectivity.

4. Guidance is applied to the network utilizing the following strategy:

    - The number of sub networks to be utilized is reduced using insight into the specific problem to be solved as well the possible sequencing scenarios involved.
    - The number of actions within each subnetwork is reduced depending on the positioning of the subnetwork in the behavior execution sequence and the actions that could possibly allow for achieving the desired behavior goals.

The following sections discuss the implementation details of several control problems and their associated control strategies developed utilizing guided genetic evolution. For each of the problems presented, the implementation process discussed is used to encode the robotic agent as a trigger network, then guided evolution is applied to reach a trained controller capable of achieving the desired goals.

5.3 Inverted Pendulum

The *Inverted Pendulum* problem is used extensively as a benchmark for testing various types of control strategies. The problem has been presented previously, and we present the general setup of the inverted pendulum problem in Figure 5.1 for convenience. In this section, we encode the associated control problem as an evolvable trigger network in order to achieve the most optimum control strategy. The final results are evaluated according to the level at which the desired output is achieved.

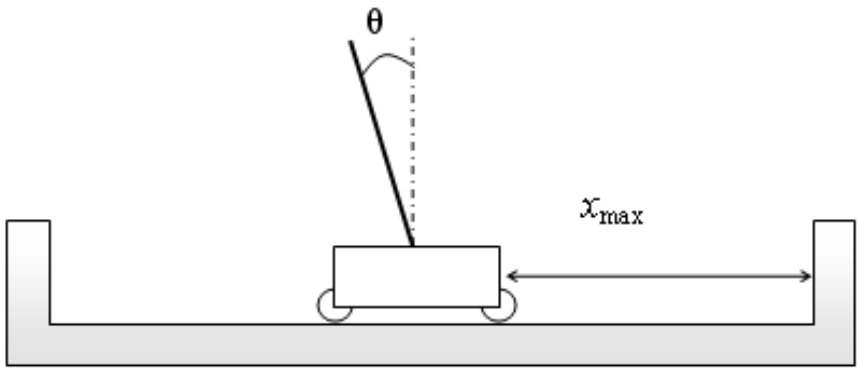


Figure 5.1: The inverted pendulum environment.

Four different input parameters are used to describe the state of the pendulum system at any given point in time t. Instead of partitioning each parameter space into several

partitions, we use the direct value of the parameter as input to the system and build our control strategy based on the entire parameter space in order to achieve the most optimum results.

The following four parameters are available to the controller at each time step t:

- x_t: The horizontal distance of the cart along the x-axis measured from the cart initial starting position. The value is given in meters and is constrained to a maximum value of x_{max}. The control strategy has to detect when the cart is located in close proximity to the distance boundary and apply corrective measures to move the cart closer to the point of origin.

- v_t: The horizontal velocity of the cart along the x-axis. The value is given in meters per second. The magnitude of v_t should stay close to zero as long as the pole is balanced. As forces are applied to the pole in an effort to balance the pole, $|v_t|$ will temporarily increase for a duration of time then decrease back to being close to zero minimizing the rotational motion of the pole when it is in a balanced state.

- θ_t: The pole's clockwise angle measure to the z-axis. The maximum angle allowed is defined as θ_{max}. The pole balancing is considered successful as long as the absolute value of the angle θ is less than θ_{max}.

- ω_t: The angular velocity of the pole measured in degrees per second. Pole balancing is achieved by trying to minimize the effect of the forces on the pole at any given time. Minimizing and maintaining ω_t close to zero guarantees keeping the pole in a balanced position.

5.3.1 Action Specifications

The first step in encoding the control problem as a trigger network requires the determination of all relevant actions to be performed by the agent and applied by the evolved controller. In this particular scenario, the specifications of the problem dictate a single action to be performed as a means for controlling the robotic structure. The problem states that the pole is to be balanced only via the application of forces to the cart in a manner that counters the instability of the pole. Hence, a single control mechanism is utilized within the trigger network, yet several partitions of the control strategy are utilized contributing to the calculation to the most optimum force application.

The net force vector v applied to the cart is defined as

$$v = (x, 0, 0)$$

where x is the magnitude of the force to be applied to correct the pole position.

Although a single force application is used for controlling the robotic system, the calculation of the force magnitude and direction stems from the execution of the following actions:

- *Action* a_1 represents the θ constraint component of the control strategy. As the angle θ approaches θ_{max}, forces are applied to the cart in the direction of pole movement to increase the pole's angular velocity in the opposite direction. The closer θ is to zero, the lower the level of force application in order to reduce the possibility of pushing the pole into a violent movement.

  Action a_1 is configured as a *control strategy* action which is to be evolved in an aim to find the most optimal functional relationship between the pole angle θ and the force to be applied to the cart. The action aims to maintain the constraint

  $$|\theta| < \theta_{max} \tag{5.1}$$

  through the utilization of the most optimal functional relationship genetically chosen from Table 4.1.

- *Action* a_2 represents the *x* constraint component of the control strategy. The action aims to prevent the cart from moving beyond x_{max} distance from the point of origin, so as the distance *x* approaches *xmax*, forces are applied to the cart in order to move it back towards the point of origin. Action a_2 is also configured as a *control strategy* action which is to be evolved in an aim to find the most optimal functional relationship between the distance of the cart relative to x_{max} and the force to be applied to the cart. The action aims to maintain the constraint

  $$|x| < x_{max} \tag{5.2}$$

  through the utilization of the most optimal functional relationship genetically chosen from Table 4.1.

5.3.2 Initial Network Layout

The initial trigger network layout consists only of the two actions listed, and both actions are connected to each other via dual-directional trigger vectors. The layout is shown in Figure 5.2.

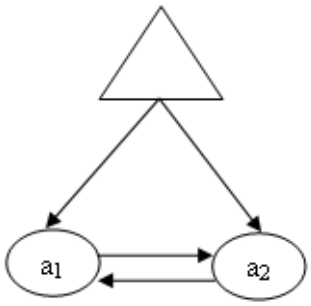


Figure 5.2: Layout for the inverted pendulum trigger network.

Both actions a_1 and a_2 are control strategy actions, which means that they execute for the entire life-cycle of the agent attempting to continuously achieve a particular goal. Hence, the trigger vectors connecting the two actions are not quite relevant as both actions should start their execution cycle as soon as the controller is triggered.

5.3.3 Network Guidance

As long as the mentioned system goals are being maintained, the system is considered to be in a successful balancing state. Network guidance is achieved by using insight into the nature of the problem as well as the actions to be executed by the agent to enhance the network layout and reduce its complexity. The guidance procedure aims to make adjustments to action parameters and inclusion, trigger vector parameters and activation, as well as subnetwork connectivity when sub networks are utilized within the network.

Control strategy actions require the specification of a global force application parameter to be utilized in the evolutionary process. The value of 20 is specified as the global force application parameter for both actions within the network. In addition to the evolution of the functional relationships related to actions, a scaling parameter is also evolved for each action in order to scale the constant force value to the most optimum force magnitude to be utilized by the action.

Since the network actions are continuous actions that perform continuous adjustments to the system, we guide the actions to commence execution at the moment the entire network is triggered. Hence, we set the magnitudes of the trigger vectors connecting the

network root to the action nodes to zero. In addition, all trigger vectors connecting action nodes to each other are removed. All actions are to be triggered by the root, and none of the actions trigger other actions; hence, no dependency exists among action nodes, and the presence of the trigger vectors is not needed. Figure 5.3 shows the layout of the guided trigger network for the inverted pendulum problem.

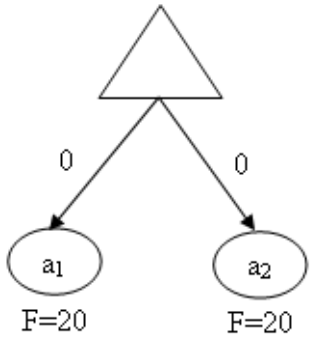


Figure 5.3: Guided trigger network for the inverted pendulum problem.

5.3.4 Fitness Function

The ultimate goal to be achieved by the system is to maintain balance of the pole on a continuous basis. The fitness function aims to measure the performance of each individual within the population by evaluating their results in relation to the goal. For the inverted pendulum problem, the performance of each individual is measured according to the number of time steps through which the pole remains balanced. Once the pole balancing fails, the fitness function returns the time step count to be used as the fitness value for the individual. The pole balancing fails if any of the following conditions become true:

$$|\theta| \geq \theta_{max}$$

or

$$|x| \geq x_{max}$$

The time step value returned by the fitness function is used to determine the selection probability for each individual.

5.3.5 Evolution Results

The following parameters were used to drive the evolutionary process:

Individuals:	10
Generations:	80
Crossover α	0.1
p_c	0.1
p_m	0.01
θ_{max}	12
x_{max}	2.5

Table 5.1: Evolution parameters for $x_{max} = 2.5$.

The final evolutionary results of the corrective force application components were as follows:

$$\begin{array}{ll} \theta \text{ corrective force} & f_\theta = 19.32 \cdot \theta^3 \\ x \text{ corrective force} & f_x = 3.74 \cdot x^3 \end{array}$$

The sum of the two force components was used as the final force magnitude to be applied to the cart. We notice that the scaling parameter for f_θ has evolved to a value close to 1.0 showing that the θ correction requires the most amount of force in order to achieve the best results. The scaling parameter for f_x, however, has evolved to a smaller value.

Figure 5.4 shows the results of the evolutionary process. We notice a gradual increase in the overall fitness of individuals throughout the 80 generations of evolution.

5.3.6 Problem Expansion

We now present a slight expansion of the problem in an aim to promote system stability as part of the evolutionary process. We repeat the previous genetic experiment with a single modification of the fitness function limiting the x_{max} parameter to 1.0 instead of 2.5. Following this newly introduced limitation, the cart must remain within 1.0 unit of the point of origin in order for the balancing to be considered successful. Table 5.2 shows the parameters used in the evolutionary process.

Although we do not view a significant change in f_θ as x_{max} is reduced, we do notice a significant increase in f_x in order to maintain the cart closer to the origination point. The final evolved force application functions for $x_{max} = 1.0$ are defined as:

$$\begin{array}{ll} \theta \text{ corrective force} & f_\theta = 18.68 \cdot \theta^3 \\ x \text{ corrective force} & f_x = 11.46 \cdot x^3 \end{array}$$

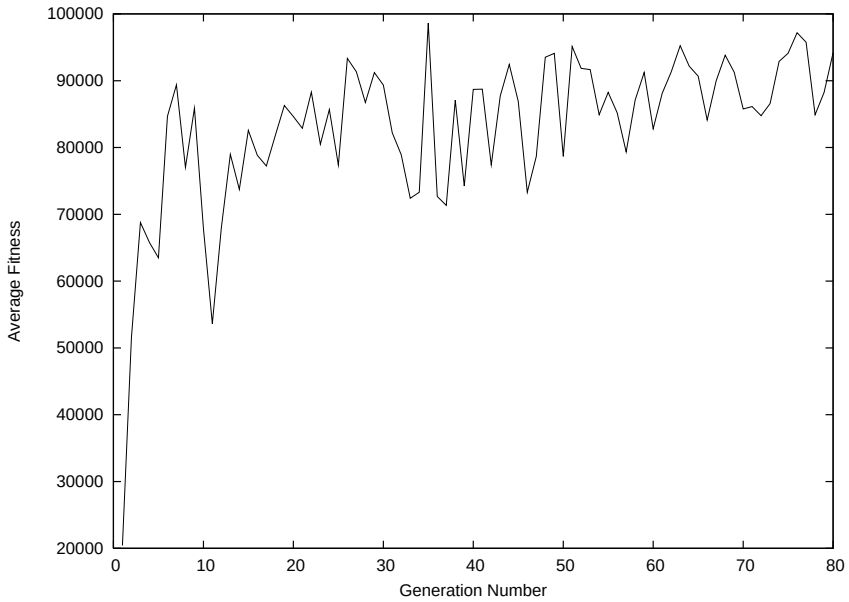


Figure 5.4: Evolutionary results for $x_{max} = 2.5$.

Individuals:	10
Generations:	80
Crossover α	0.1
p_c	0.1
p_m	0.01
θ_{max}	12
x_{max}	1.0

Table 5.2: Evolution parameters for $x_{max} = 1.0$.

Figure 5.5 shows the results of the $x_{max} = 1.0$ problem variation. We notice a gradual increase in performance very similar to the previous results. The average fitness results demonstrate that by applying the correct behavior application, we may maintain successful evolution, even as more constraints are placed on the system.

5.3.7 PID Control

We further expand the problem methodology to utilize evolved PID controller actions instead of control strategy actions. This change would allow us to gauge the efficiency of utilizing evolved PID compared to other methods. For this problem, we use the same network layout as in the previous experiment. However, we limit the action nodes to only two described as follows:

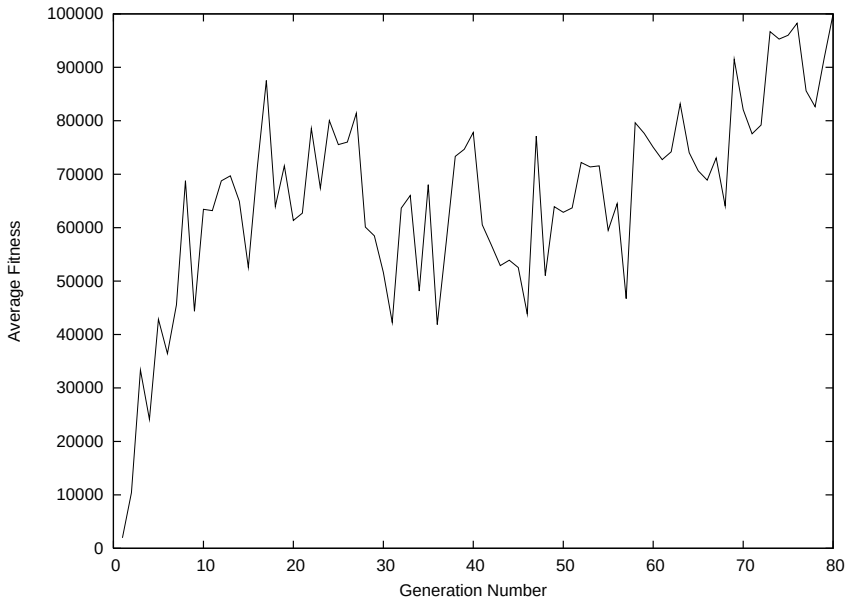


Figure 5.5: Evolutionary results for $x_{max} = 1.0$.

- Action a_1 is configured as a PID action which aims to find the most optimal PID control strategy using the pole angle θ as input and outputting the force component value f_θ.

- Action a_2 is configured as a PID action using the distance from the origin x as input and outputting the force component value f_x.

In the previous experiment, the velocity of the cart v and the angular velocity of the pole ω were not used as part of the balancing strategy. The evolved PID strategy, however, utilizes the distance x and the angle θ to internally calculate the associated rates of change to allow for more efficient and stable control.

The main goal of the PID control evolution is to find the most optimal PID parameters K_p, K_i and K_d to satisfy the equation

$$f = K_p \cdot e + K_i \int e \cdot dt + K_d \frac{de}{dt} \tag{5.3}$$

where f is the resultant force calculated by the controller, and e represents the difference between the current state of the system and a desired goal state.

For the purposes of controlling the pole, we utilize the two force values f_θ and f_x to be combined into a single force value f to be applied to the cart. Given the two input values θ and x, the two forces f_θ and f_x are defined as:

$$f_\theta = K_{p\theta} \cdot \theta + K_{i\theta} \int \theta \cdot dt + K_{d\theta} \frac{d\theta}{dt} \tag{5.4}$$

$$f_x = K_{px} \cdot x + K_{ix} \int x \cdot dt + K_{dx} \frac{dx}{dt} \tag{5.5}$$

where θ represents the angle of the pole and x represents the distance from the origin. The force f to be applied to the cart is defined as

$$f = f_\theta + f_x \tag{5.6}$$

Table 5.3 shows the parameters used in the evolutionary process:

Individuals:	50
Generations:	50
Crossover α	0.2
p_c	0.1
p_m	0.05
θ_{max}	12
x_{max}	1.0

Table 5.3: PID control evolution parameters for inverted pendulum problem.

Figure 5.6 shows the evolution results of the inverted pendulum controller utilizing evolved PID control. We notice a gradual and stable increase in performance through out the evolutionary process. The slope of the performance growth is significantly more stable than that of control strategy based implementation, which is attributed to the PID control tolerance for slight changes in the PID parameters. Figure 5.7 shows the final evolved guided trigger network for the inverted pendulum problem utilizing PID control. The figure shows the final PID parameter values attained through the evolutionary process.

5.4 Robotic Arm

This section presents another robotic control problem which relies heavily on the evolution of trigger vector parameters as well as action parameters. The problem demonstrates the evolution of the trigger network layout in an effort to achieve higher population fitness. The problem is based on a robotic arm consisting of a base and two partitions connected using universal joints. The robotic arm setup is shown in Figure 5.8. As seen in the figure, the arm is stationed in front of a wall which contains a tunnel large enough

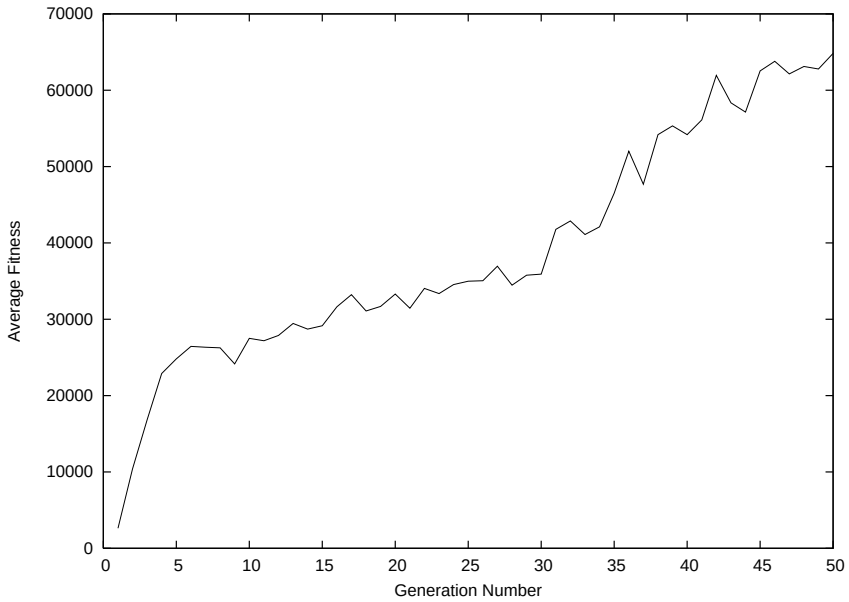


Figure 5.6: Evolution progression of the inverted pendulum problem utilizing evolved PID control.

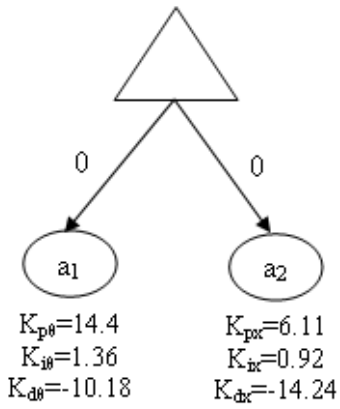


Figure 5.7: Evolved guided trigger network for the inverted pendulum problem utilizing PID control.

to accommodate the robotic arm being extended within. Towards the end of the tunnel, a target sphere is placed marking the desired destination for the tip of the robotic arm.

Since the arm is given the freedom to rotate in any direction, collision with the wall or the ground is prohibited. If collision occurs, the individual is considered to have failed the task and given a fitness value of zero. Otherwise, both arm partitions can move freely on two different axis for each joint. The arm has four degrees of freedom and the only limitations placed on the motion of the arm are dictated by the low and high stop points

of each of the joint axis.

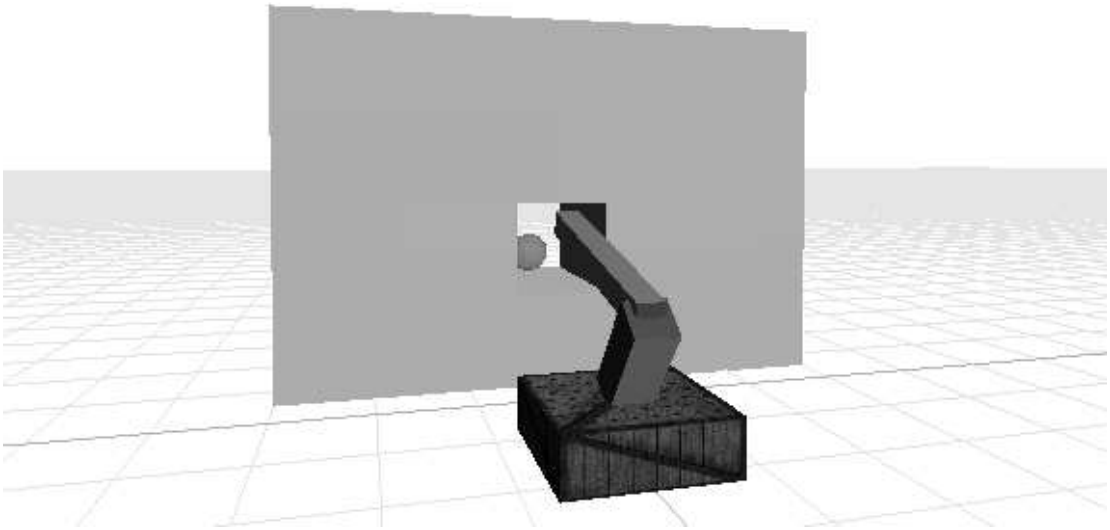


Figure 5.8: Robotic arm problem setup showing the target sphere representing the desired destination for the tip of the arm.

5.4.1 Action Specifications

The robotic arm is controlled via two universal joints. The first joint connects the lower arm partition to the base, and the second joint connects the upper arm partition to the lower one. Since each of the universal joints operates on two degrees of freedom, the entire arm has four degrees of freedom for which control commands may be issued. The robotic arm control scenario translates to four different actions to be encoded in the trigger network. The four actions are listed as follows (Figure 5.9):

- Action a_1 is configured as a direct control action to control the lower universal joint issuing commands to control the joint axis movement perpendicular to the wall. Hence, the action may issue commands to move the entire arm closer or further away from the wall.

- Action a_2 is configured as a direct control action to control the lower universal joint issuing commands to control the joint axis movement parallel to the wall. Hence, the action may issue commands to move the entire arm to the right or to the left relative to the wall tunnel.

- Action a_3 is configured as a direct control action to control the upper universal joint issuing commands to control the joint axis movement perpendicular to the wall.

Hence, the action may issue commands to move the upper arm partition closer or further away from the wall.

- Action a_4 is configured as a direct control action to control the upper universal joint issuing commands to control the joint axis movement parallel to the wall. Hence, the action may issue commands to move the upper arm partition to the right or to the left relative to the wall tunnel.

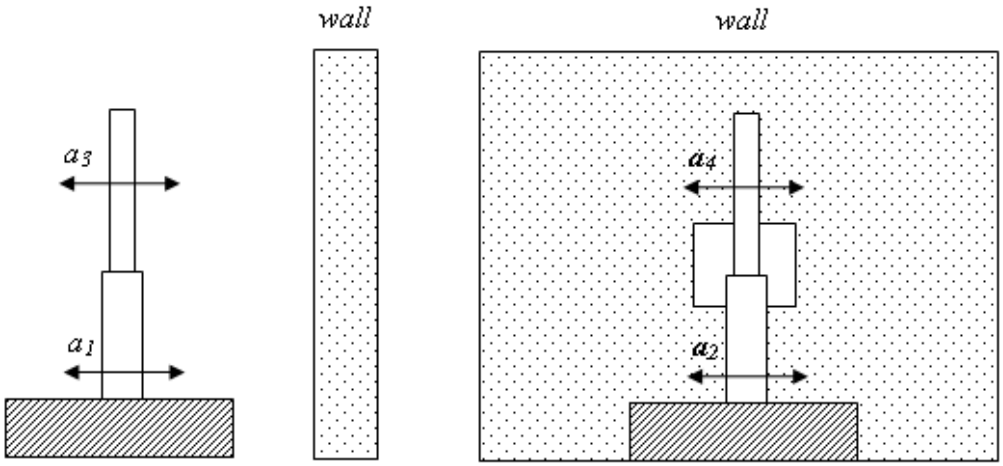


Figure 5.9: Action specifications for the robotic arm problem.

The four actions listed above are duplicated as actions a_5, a_6, a_7, and a_8 respectively for the purposes of using an instance of each action in two different sub networks utilized within the network.

5.4.2 Initial Network Layout

The initial layout for the robotic arm problem utilizes two sub networks representing two distinct behaviors to be performed in succession by the agent. Each of the two sub networks will contain instances of all the actions the agent is capable of performing. The network is configured where one of the sub networks is triggered by the network root, and once all the actions of that first network conclude their execution cycle, the second subnetwork is then triggered to perform its own actions.

The first subnetwork utilized in the network is labeled b_1, which is triggered directly by the root, and in turn it triggers its four internal actions a_1, a_2, a_3, and a_4. The trigger vectors connecting the relay node b_1 to the four actions are assigned a trigger vector magnitude between 0.0 and 1.0. The exact value is to be genetically evolved.

The second subnetwork utilized is labeled b_2. Once all the actions of b_1 conclude their execution, b_2 is triggered. This is indicated by the presence of trigger vectors connecting each of the actions of b_1 to the relay node b_2. Each of the trigger vectors entering b_2 are assigned a magnitude of 1.0. This indicates that b_2 is triggered immediately after the associated action has concluded. Once b_2 is triggered, it triggers its own associated actions, a_5, a_6, a_7, and a_8 depending on the trigger vector magnitude utilized. The exact magnitudes are also evolved to a value between 0.0 and 1.0.

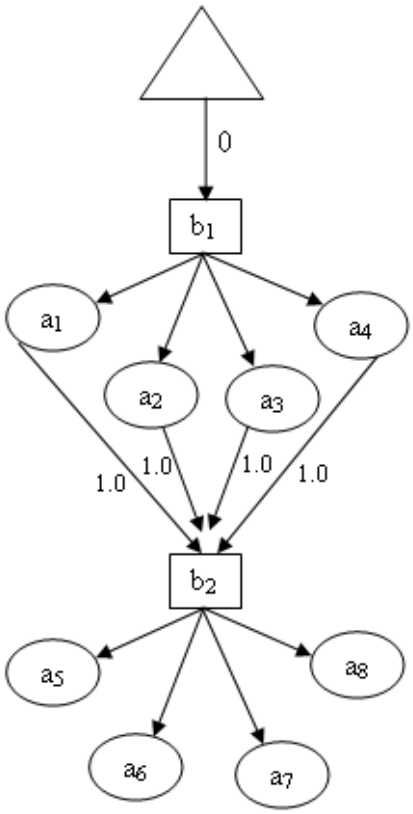


Figure 5.10: Initial trigger network layout for the robotic arm problem.

5.4.3 Network Guidance

For the presented problem, we limit the network guidance to the minimum in order to fully demonstrate the network evolution phases. Although insight into the problem suggests that since the tunnel is directly in front of the robotic arm, the actions designated for moving both arm partitions in a direction that is parallel to the wall are not needed. However, we aim to give the evolutionary process a chance to deactivate the actions in

question by carefully selecting a fitness function that takes into the account any harmful effects that those actions might cause.

Insight into the problem suggests the following desired configuration that would yield successful results, even though guidance is not applied to achieve those configurations (Figure 5.11):

- Actions a_2, a_4, a_6 and a_8 should be deactivated or evolved to a direct angle control that is close to zero. This would allow the arm to maintain a position in front of the tunnel opening.
- Behavior b_1 should bring the tip of the upper arm partition in front of the tunnel opening in preparation for pushing the arm through the tunnel. This is achieved by utilizing actions a_1 and a_3 to position the upper arm partition parallel to the ground facing the tunnel opening.
- Behavior b_2 should then move the upper arm partition through the tunnel by using a_5 to push the arm towards the wall while a_7 is utilized to keep the arm from colliding with any of the tunnel walls.

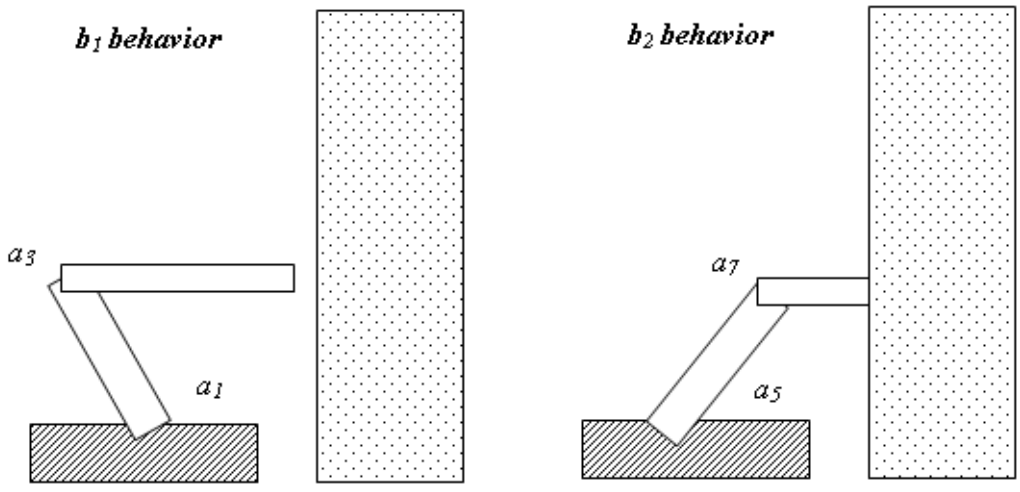


Figure 5.11: Desired robotic arm configuration for behaviors b_1 and b_2.

5.4.4 Fitness Function

The fitness function for the robotic arm problem is chosen carefully to maintain the overall execution goals. The following rules are followed:

The arm is not allowed to collide with the wall or the ground throughout its execution cycle. Any collision with the said entities causes the agent to fail its control attempt and a fitness value of zero is returned.

Any movement along the y-axis that moves the arm away from the target causes the agent to incur a penalty proportional to the deviation distance.

Any movement along the z-axis that moves the arm away from the target causes the agent to incur a penalty proportional to the deviation distance.

Once the entire network has concluded its execution a penalty is applied proportional to the distance separating the tip of the robotic arm and the target sphere.

Given the final distance from the y-axis, d_y, the final distance from the target along the z-axis, d_z, and the final distance from the target along the x-axis, d_x, the fitness function f is defined as:

$$f = d_x^5 + d_y^5 + d_z^5 \tag{5.7}$$

5.4.5 Evolution Results

Due to the higher complexity of the presented problem, a dynamic individual count is utilized for the evolutionary process. The dynamic allocation of individuals allows the system to start with a high individual count and then gradually decrease the count to a minimum value as the system approaches the end of the evolutionary cycle. The following parameters are used to drive the evolutionary process:

Individuals (Minimum):	50
Individuals (Maximum):	200
Generations:	100
Crossover α	0.2
p_c	0.1
p_m	0.01

Table 5.4: Evolution parameters for the robotic arm problem.

The robotic arm evolution utilizes a two-phase approach in an effort to reach the desired goals in a short duration of time. The first phase evolves the behavior presented by b_1, while the second phase expands the network training to include the behavior presented

by b_2. The fitness function is structured in a manner that gradually forces the tip of the robotic arm closer to the tunnel opening by the end of the execution of the first behavior.

Figure 5.15 shows the best performing individual representing behavior b_1 after 25 generations of evolution.

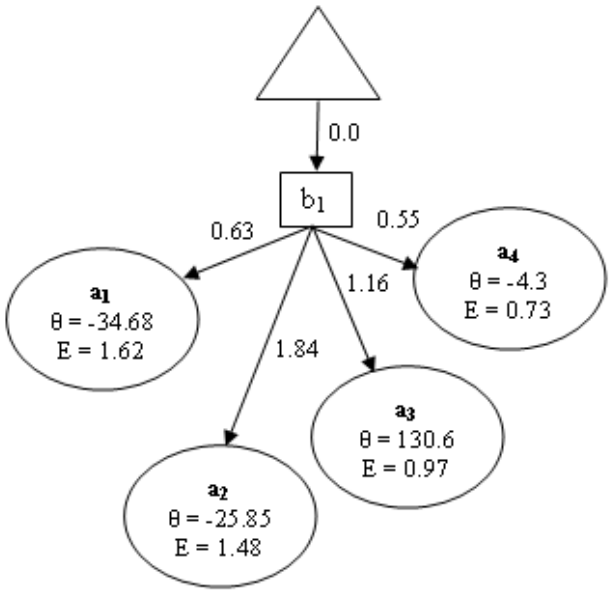


Figure 5.12: Best performing individual representing behavior b_1 after 25 generations of evolution.

We notice the assortment of urgency values that have been reached for the different actions performed. The urgency of action execution becomes significantly more relevant in the execution of b_2 as the possibility of colliding with the wall or the ground is much higher. The urgency values evolved, along with the angle commands, guarantee that the arm does not collide with any of the walls or the ground. In addition, the fitness function guarantees the selection of individuals achieving more proximity to the target.

Actions a_2 and a_4 contribute to the lateral movement of the arm, and we notice that the combination of angle commands given allow the top of the upper arm partition to still be close to the tunnel opening. Figure 5.15 shows the arm position of the best performing individual after 25 generations of evolution.

As the evolution process continuous, we find that the genetic process eventually eliminates action a_3 by deactivating it completely. This elimination allows the system to achieve higher consistency by reducing the lateral movement keeping the arm closer to the desired target position. The best performing network representing behavior b_1 after 50 generations of evolution is shown in Figure 5.14.

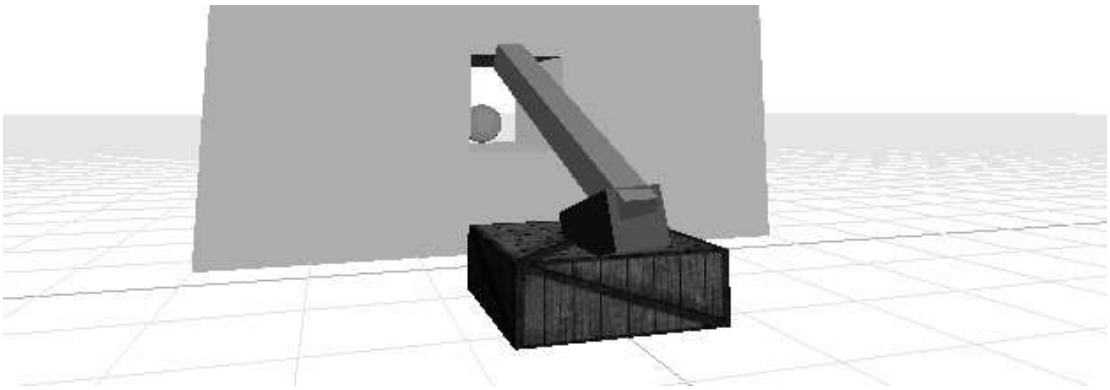


Figure 5.13: Arm position of best performing individual after 25 generations of evolution.

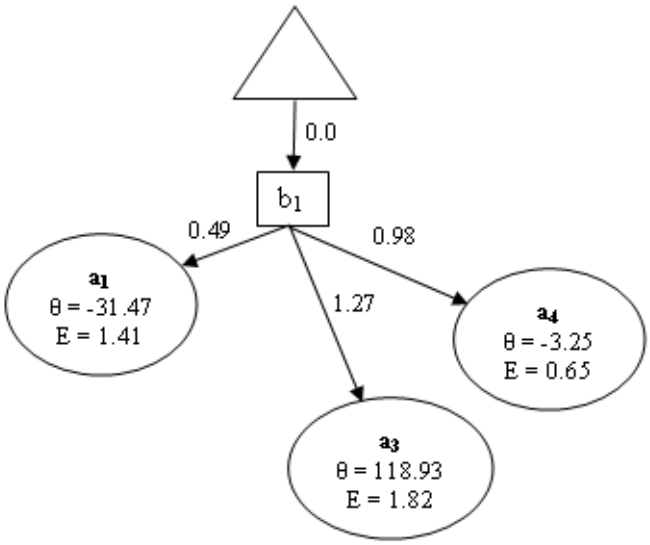


Figure 5.14: Best performing individual representing behavior b_1 after 50 generations of evolution.

We notice that the angle and urgency commands of actions a_1 and a_3 are still in close proximity to their associated values in Figure 5.15. The angle command associated with action a_4 has converged to value small enough to maintain a relatively high fitness value.

After the first 50 generations of evolution, the second phase of the genetic process is initiated through the inclusion of the second subnetwork, b_2. This allows the genetic process to evaluate the performance of each individual as a whole including all associated behaviors. Starting from a point already close to the tunnel opening increases the chances for finding a proper configuration that would allow the arm to achieve the goal. A desired configuration would involve increasing the angle associated with action a_1 moving the arm closer to the wall while decreasing the angle associated with action a_2 in order to

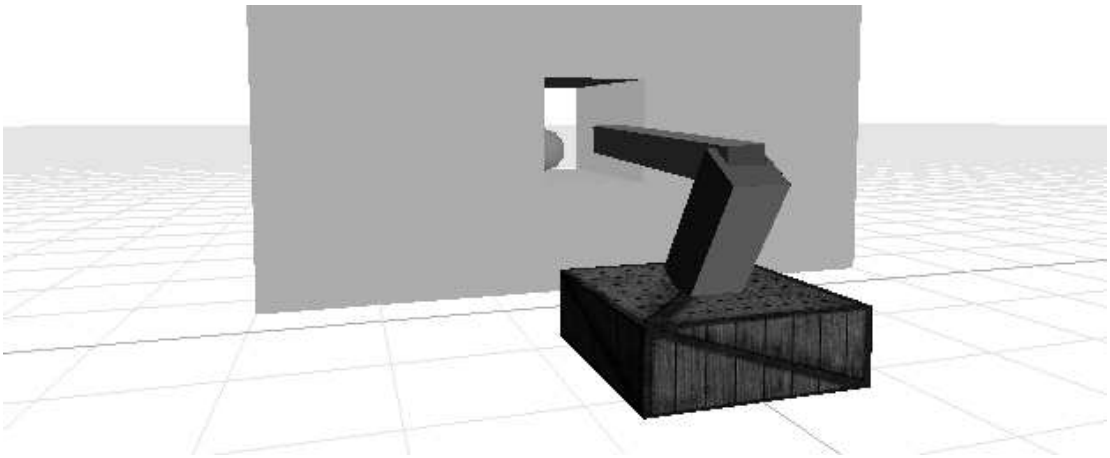


Figure 5.15: Arm position of best performing individual after 50 generations of evolution.

keep the second arm partition level. This would allow the arm to move through the tunnel without colliding with any of the walls.

The final configuration of the trigger network after 100 generations of evolution is shown in Figure 5.16. In the second phase of evolution, actions a_6 and a_8, which are responsible for the lateral movement of the arm, were eliminated by the evolutionary process. The utilization of only actions a_5 and a_7 allows the system to converge to a solution faster as both actions force the arm to move in a direction that is normal to the wall surface.

The final position of the complete arm behavior after 100 generations of evolution is shown in Figure 5.17. The genetic process was successful in evolving an individual capable of reaching the desired evolution goals. The execution sequencing and urgency were critical in allowing the arm to navigate the tunnel without colliding with any of the wall surfaces ultimately reaching a position close to the target.

5.5 Conclusion

This chapter demonstrated several testing scenarios utilizing trigger network evolution for achieving successful control of articulated robotic structures. It has been shown that the network evolution methods presented are capable of achieving the desired goals for both continuous PID-based control as well as complex execution sequencing of joint actions.

In addition to the base network evolution, the careful application of network guidance has also been proven to increase the overall fitness of individuals relative to unguided evolution. This is accomplished through the simplification of the network structure by

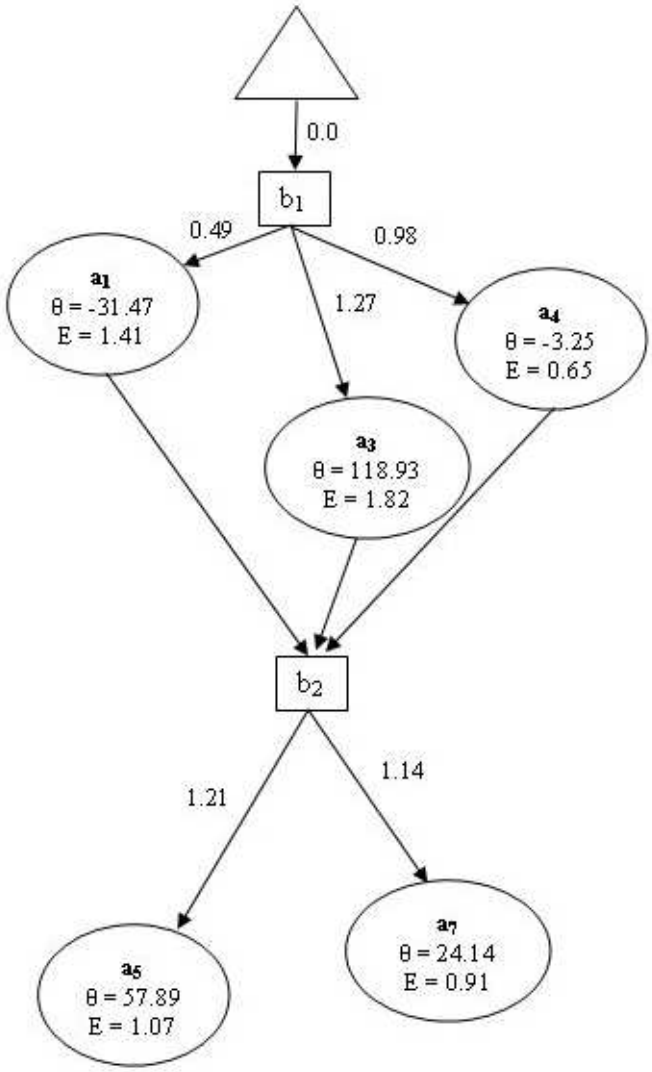


Figure 5.16: Final configuration of trigger network for the robotic arm problem after 100 generations of evolution.

reducing the number of variables involved in the genetic process. A reduced search space increases the system likeliness for converging to a solution that meets the preset goals. The careful selection of evolution parameters, like generation size as well as crossover and mutation probabilities for example, also play a crucial role in the achievement of desirable evolution results.

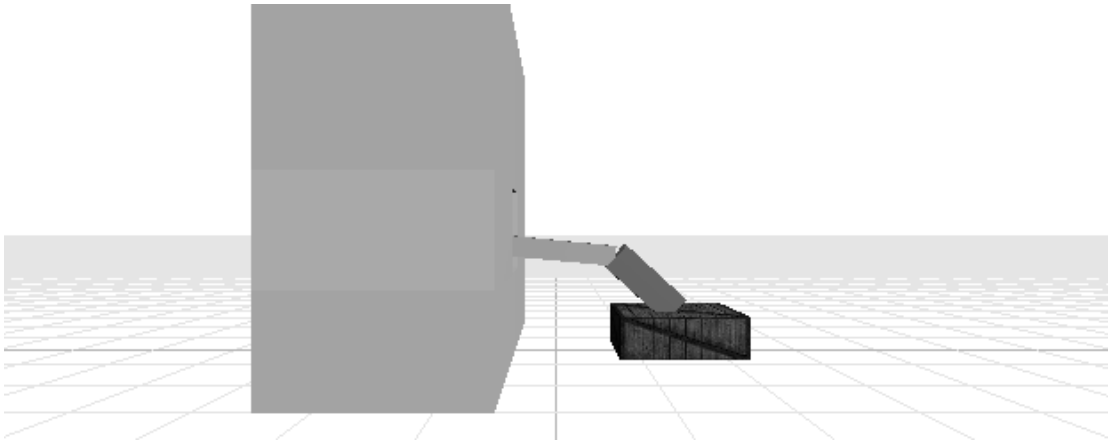


Figure 5.17: Final robotic arm position after 100 generations of evolution.

Chapter 6

EVOLUTION OF ROBOTIC MOBILITY

6.1 Introduction

The guided genetic evolution methodologies have been presented as an efficient tool for the evolution of autonomous robotic control. The previous chapter presented several examples relating to the application of the genetic framework towards the evolution of different types of robotic structures. The guidance of the genetic process has also been shown to increase the evolution efficiency by reducing the search space size allowing for the achievement of higher fitness.

In this chapter, the more complex problem of controlling a biped robot is presented. The number of degrees of freedom as well as the dynamics of the robotic structure significantly complicates the control problem. Hence, we present a framework for the structuring of the control problem to achieve biped mobility within a known simulated environment. The robotic articulated structure will be presented along with the configuration of each of the joints utilized. The trigger network layout will also be presented to show the execution sequencing of the different agent behaviors. The details of the network guidance applied to the network will be discussed to convey the effect of the guidance process on the convergence rate and the accuracy of the control strategy.

6.2 Robotic Structure

The structure of the articulated biped robot consists of multiple body parts connected through the use of hinge, universal and ball-and-socket joints. The joint selection is made based on the degrees of freedom required at each of the robotic joints. For each of the joint axis, low and high stop points are specified depending on the desired range of motion for the joint. Since the structure of the biped robot resembles that of a the human body, the attributes of the human joints were used as the basis for specifying the parameters for the biped. The trigger network layout is dependant on the joint count as well as the number of axis used at each joint, hence, the articulated structure was configured to support the minimum requirements for biped locomotion in an effort to minimize the complexity of the control problem.

In order to achieve maximum efficiency in the structuring of the biped, a direct mapping was utilized between the average human body dynamics and that of the biped robot. This method allowed for the approximation of proper joint locations to connect the different body parts. Figure 6.1 shows the general articulated structure for the biped robot. As seen in the figure, boxes were used as the main primitive objects representing the different body parts. The utilization of primitive object types drastically improved the real-time performance of the system without a significant impact on the simulation accuracy. A uniform density was used for all parts utilized in the robotic structure; hence, the weight of each body part was determined based on volume.

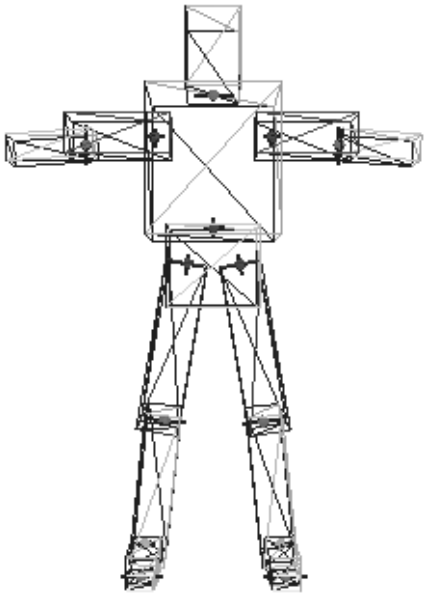


Figure 6.1: General articulation structure for biped robot.

6.2.1 Lower Section Articulation

The lower section of the robotic agent consists of the thighs, legs, and feet. Three body parts were utilized on each side of the body while maintaining symmetry between both sides. Table 6.1 shows the dimensions and orientation of the body parts utilized in the lower section of the articulated structure.

The joint coordinates of the lower section of the biped are shown in Figure 6.2. For each joint, The x, y and z coordinates are given indicating the joint anchor. The knee and toe joints for both legs are structured as hinge joints, which means that they operate along

Part	Position(x, y, z)	Dimensions (x, y, z)	Orientation(x, y, z)
Left Toe	(9.5, 0.0, 0.0)	(1.27, 0.76, 0.63)	(0.0, 0.0, 0.0)
Left Foot	(3.32, 1.92, 0.0)	(1.42, 3.07, 1.19)	(0.0, 0.0, 0.0)
Left Leg	(3.32, 1.92, 4.85)	(1.67, 2.06, 7.31)	(-0.10, 0.11, 0.0)
Left Thigh	(2.80, 2.05, 11.90)	(1.691, 2.08, 8.10)	(0, 0.17, 0.0)
Right Toe	(-9.5, 0.0, 0.0)	(1.27, 0.76, 0.63)	(0.0, 0.0, 0.0)
Right Foot	(-3.32, 1.92, 0.0)	(1.42, 3.07, 1.19)	(0.0, 0.0, 0.0)
Right Leg	(-3.32, 1.92, 4.85)	(1.67, 2.06, 7.31)	(-0.10, -0.11, 0.0)
Right Thigh	(-2.80, 2.05, 11.90)	(1.691, 2.08, 8.10)	(0, -0.17, 0.0)

Table 6.1: Parameters utilized for the lower section of the robotic structure.

a single axis. The ankle joint, however, is structured as a universal joint which operates along two axis allowing the controller to rotate the foot upwards and downwards as well as to the sides.

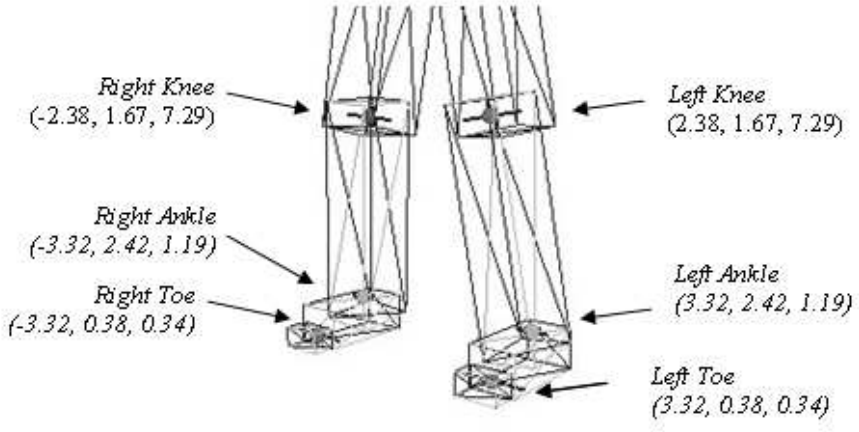


Figure 6.2: Biped lower section joint coordinates.

Joint	Axis 1	Axis 2
Left Toe	(-0.52, 0.87)	
Left Ankle	(-0.52, 1.31)	(-0.43, 0.43)
Left Knee	(-0.001, 2.97))	
Right Toe	(-0.52, 0.87)	
Right Ankle	(-0.52, 1.31)	(-0.43, 0.43)
Right Knee	(-0.001, 2.97))	

Table 6.2: Low and high stop values for lower section joints.

Table 6.2 shows the low and high stop values for each of the joint axis for the lower section of the biped. A single set of values is specified for each of the toe and knee joints. Two sets of stop values are provided for the knee joint; the first set given (axis 1) relates to the up/down rotation, while the second set (axis 2) relates to the side rotation of the ankle. Although the controller aims to maintain specific articulation angles at all instances, the low and high stop points guard against the occurrence of irregular configurations if excessive forces are applied to one or more of the body parts.

6.2.2 Middle Section Articulation

The middle section of the biped consists of the hip and torso parts. The hip connects to the legs utilizing ball-and-socket joints, which operate on three axis. The torso also connects to the hip utilizing a ball-and-socket. In addition, the torso connects to the upper body parts, consisting of the arms and head. Table 6.3 shows the dimensions and orientation of the body parts utilized in the middle section of the articulated structure, while Figure 6.3 shows the coordinates utilized for the placement of each of the middle section joints.

Part	Position(x, y, z)	Dimensions (x, y, z)	Orientation(x, y, z)
Hip	(0.0, 2.05, 16.8)	(4.37, 3.27, 3.81)	(0.0, 0.0, 0.0)
Torso	(0.0, 2.05, 20.09)	(6.27, 4.35, 6.91)	(0.0, 0.0, 0.0)

Table 6.3: Parameters utilized for the middle section of the robotic structure.

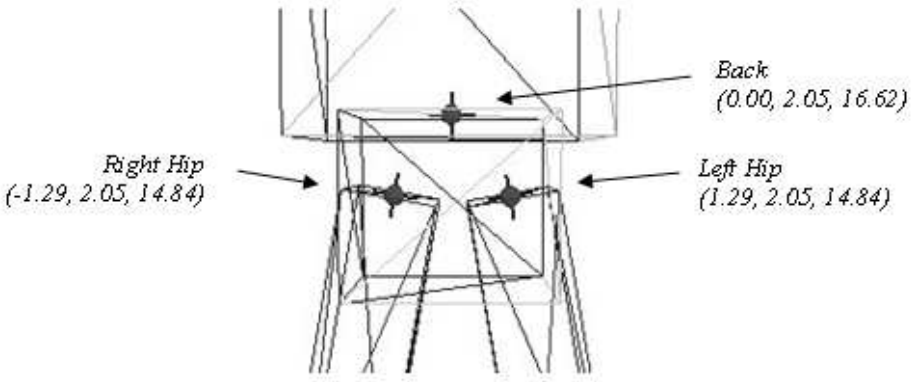


Figure 6.3: Biped middle section joint coordinates.

As mentioned above, a ball-and-socket joint, representing the waist, is used to connect the torso to the hip. The joint operates on three axis allowing for movement of the torso in

any direction as well as for rotation. The agent's ability to rotate the upper body section forwards, backwards or to the side significantly improves the agent's balancing abilities while walking. The hip joints for both legs consist of ball-and-socket joints operating on three axis. This allows for the legs to rotate in any direction resembling known hip movement. Table 6.4 shows the low and high stop values for each of the joint axis for the middle section of the biped.

Joint	**Axis 1**	**Axis 2**	**Axis 3**
Waist	(-1.31, 0.52)	(-0.52, 0.52)	(-0.52, 0.52)
Left Hip	(-1.39, 0.34)	(-0.61, 0.61)	(-0.69, 0.69)
Right Hip	(-0.34, 1.39)	(-0.61, 0.61)	(-0.69, 0.69)

Table 6.4: Low and high stop values for middle section joints.

The first axis of the waist joint controls the forwards and backwards movement, while the second axis controls the lateral movement. Both hip joints utilize the same configuration where the first axis controls the lateral movement, the second axis controls the rotation of each leg, while the third axis controls the forwards and backwards movement.

6.2.3 Upper Section Articulation

The upper section of the robotic structure consists of the head, shoulders and arms. Although the upper body parts do not contribute directly to the mobility of the robot, they play a crucial role in the task of keeping the robot balanced. The shoulders and arms also paly a role in minimizing the swaying side effect that occurs during the execution of the walking behavior. Table 6.5 shows the dimensions and orientation of the body parts utilized in the upper section of the articulated structure.

Part	**Position(x, y, z)**	**Dimensions (x, y, z)**	**Orientation(x, y, z)**
Head	(0.0, 2.05, 24.85)	(2.54, 3.22, 3.64)	(0.0, 0.0, 0.0)
Left Shoulder	(5.60, 2.05, 19.56)	(4.92, 2.12, 1.94)	(0.0, 0.0, 0.0)
Left Arm	(8.94, 2.75, 21.14)	(3.85, 1.55, 1.29)	(0.0, -0.07, 0.0)
Right Shoulder	(-5.60, 2.05, 19.5)	(4.92, 2.12, 1.94)	(0, 0.0, 0.0)
Right Arm	(-8.94, 2.75, 21.14)	(3.85, 1.55, 1.29)	(0.0, 0.07, 0.0)

Table 6.5: Parameters utilized for the upper section of the robotic structure.

The neck is connected directly to the torso utilizing a ball-and-socket joint that would allow rotation in any direction. A ball-and-socket joint is also used for the shoulders

to allow freedom of motion along any axis. The elbows, however, utilize a hinge joint allowing rotation along a single axis. The exact joint anchor coordinates are shown in Figure 6.4. Table 6.6 shows the low and high stop values for each of the joint axis for the upper section of the biped.

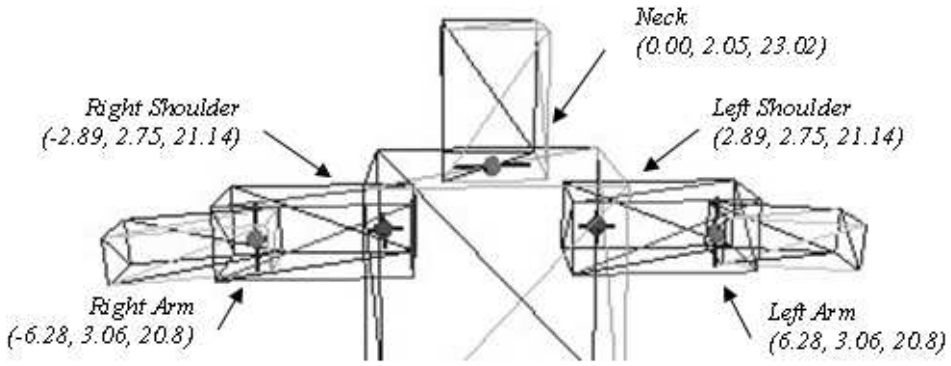


Figure 6.4: Biped upper section joint coordinates.

Joint	Axis 1	Axis 2	Axis 3
Neck	(-1.13, 1.13)	(-1.13, 1.13)	
Left Shoulder	(-1.22, 1.57)	(-0.43, 0.43)	(-1.39, 1.71)
Left Arm	(-0.01, 1.31)		
Right Shoulder	(-1.57, 1.22)	(-0.43, 0.43)	(-1.71, 1.39)
Right Arm	(-1.31, 0.01)		

Table 6.6: Low and high stop values for Upper section joints.

6.2.4 Polygon-based Representation

In order to enhance the realism of the agent rendering, a polygon-based representation of each of the body parts is utilized. A modeled object is used for the visual representation of each part of the articulated structure to better represent its associated purpose, then the modeled object is positioned to be encapsulated by the primitive object being utilized for articulation. The primitive objects are still the main parts used for volume calculations as well as for collision detection and avoidance, yet the use of the modelled objects significantly enhanced the realistic rendering of the robotic structure.

Figure 6.5 shows a polygonal wire frame rendering of the robotic agent. The figure also shows the location of all joint anchors which correspond to the exact positions

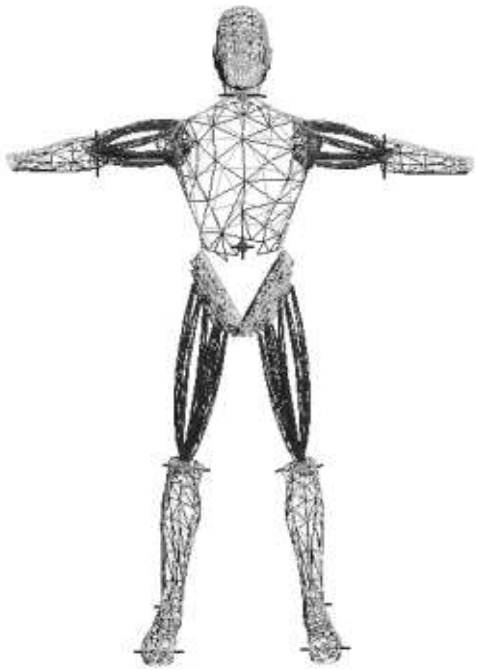


Figure 6.5: Polygonal wire frame rendering of the robotic agent showing the position of joint anchors.

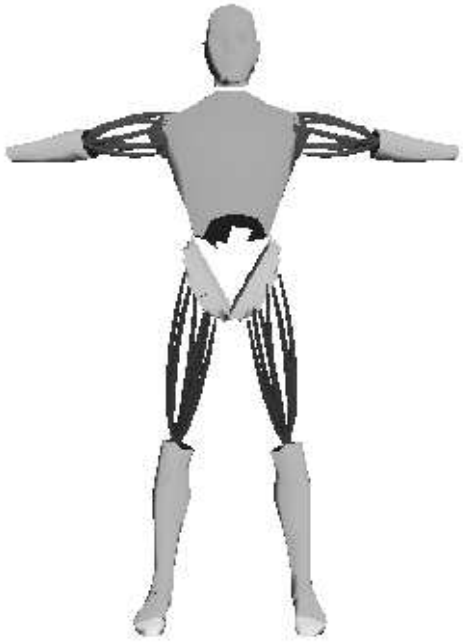


Figure 6.6: Shaded rendering of the robotic agent.

utilized with the primitive object types. Figure 6.6 shows a shaded rendering of the agent.

6.3 Action Specifications

The actions to be utilized in the evolutionary process relate directly to the degrees of freedom associated with each of the articulation joints. Each axis of each joint must have an associated direct control action configured to move the body parts about the joint. Additional actions may be used to incorporate PID or strategy-based control in the overall strategy. The minimum and maximum angles associated with each direct control action are set to correspond to the low and high axis stop values respectively.

Table 6.7 shows the action specifications of the lower section of the biped. The overall network complexity was reduced through the representation the entire foot as a single object. Hence, the toe joints were fixed at a rotational angle of zero. Preliminary experimentations have demonstrated good behavioral results utilizing a single object to represent the foot.

Action	Joint	Axis	Description
a_1	Left Ankle	1	Front/Back movement
a_2	Left Ankle	2	Lateral movement
a_3	Left Knee	1	Lateral movement
a_4	Right Ankle	1	Front/Back movement
a_5	Right Ankle	2	Lateral movement
a_6	Right Knee	1	Front/Back movement

Table 6.7: Biped Action specifications for lower section.

The action specifications for the middle section are shown in Figure 6.8. The hip joints are the two most essential components of the agent's mobility. They control both forward-stepping behavior as well as the side-stepping behavior of the robot.

Table 6.9 shows the action specifications of the upper section of the biped. The arms and neck do not contribute directly to the stepping action, but they contribute to the balancing and stability of the biped.

In addition to the direct joint control actions listed, several PID control actions are utilized to maintain essential desired configurations. The following list details the PID control actions used as part of the biped stepping motion:

- a_{25} - Waist Center: In order to assist in the continuous balancing of the biped, this action is used to keep the upper body of the agent centered while the weight is

Action	Joint	Axis	Description
a_7	Left Hip	1	Front/Back movement
a_8	Left Hip	2	Leg rotation
a_9	Left Hip	3	Lateral movement
a_{10}	Right Hip	1	Front/Back movement
a_{11}	Right Hip	2	Leg rotation
a_{12}	Right Hip	3	Lateral movement
a_{13}	Waist	1	Front/Back movement
a_{14}	Waist	2	Lateral movement

Table 6.8: Biped Action specifications for middle section.

Action	Joint	Axis	Description
a_{15}	Left Shoulder	1	Front/Back movement
a_{16}	Left Shoulder	2	Arm rotation
a_{17}	Left Shoulder	3	Lateral movement
a_{18}	Left Arm	1	Front/Back movement
a_{19}	Right Shoulder	1	Front/Back movement
a_{20}	Right Shoulder	2	Arm rotation
a_{21}	Right Shoulder	3	Lateral movement
a_{22}	Right Arm	3	Front/Back movement
a_{23}	Neck	1	Front/Back movement
a_{24}	Neck	2	Lateral movement

Table 6.9: Biped Action specifications for upper section.

shifting from side to side.

- a_{26} - Left Ankle Level: In preparation for meeting the ground plane after stepping using the left foot, the foot is kept horizontal to the ground to maximize the agent's stability through the stepping motion.

- a_{27} - Right Ankle Level: The same foot leveling strategy is applied to the right foot as the right leg meets the ground after stepping with the right foot.

6.4 Network Layout

The biped mobility is focused around several sub-behaviors that must be executed in succession to allow the agent to attain a walking behavior. The main network layout is designed to reflect the different parts of the overall walking motion. In addition, the net-

work shows the relationships that exist between the different parts as well as the execution sequencing that must be followed in order to achieve successful mobility. The different phases involved in the walking motion are listed as follows:

- Agent shifts weight to leg A
- Agent prepares leg B for stepping
- Agent steps forward using leg B while shifting weight to it
- Agent steps forward using leg A while shifting weight to it

Maintaining continuous balance while executing all phases listed is clearly a high priority of the execution strategy. Balancing is mostly achieved through the careful execution of direct control sequences. Figure 6.7 shows the layout of the main phases involved in the biped walking motion.

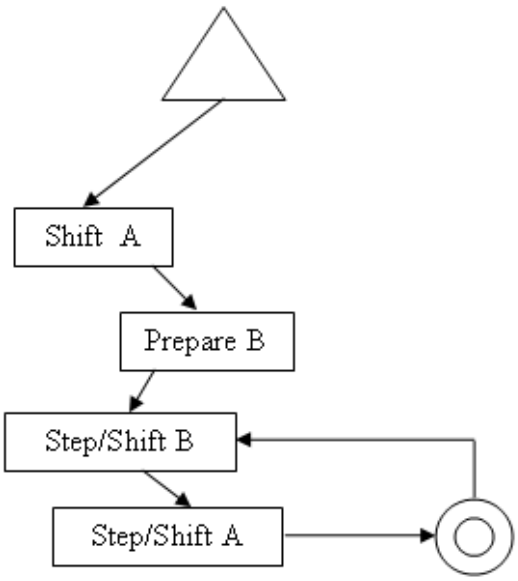


Figure 6.7: Main phases of the biped walking motion.

The first phase of the walking motion involves shifting the entire weight of the robot to one foot in preparation for stepping with the other. In the given scenario, the agent starts its motion balanced on both feet, then shifts its weight to the right foot while maintaining its balance. A direct joint control strategy must be created to allow the agent to steadily shift its weight without making any actions that would cause it to lose its balance. Each joint in the articulated body could potentially be involved in the first phase of the walking

motion, however, in order to reduce the complexity of the trigger network, guidance is applied in order to minimize the number of joint axis. The joint axis selected for the first phase of the motion, as well as the reasoning behind the selections, are classified in the following list.

- *Left Ankle*: In order to initiate a motion towards the right foot, the left ankle is used to push the body laterally towards the right.
- *Right Ankle*: Shifting the agent's weight to the left foot causes a shift in the center of mass of the body. Adjusting the angle of the right ankle joint enables the agent to balance its weight on just one foot.
- *Left Hip*: Moving the right hip laterally also allows the agent to push its body towards the other foot assisting in the weight transfer.
- *Right Hip*: The proper positioning of the right hip assists in balancing the entire body in the new desired position as the center of mass is moved to the right.

Once the agent is balanced on the right foot, the left leg is pulled closer to the body in preparation for the stepping action. This involves commanding the hip joint to pull the leg closer while applying changes to the knee and ankle positions. The joint axis to be manipulated in this phase of the motion are classified as follows:

- *Left Hip*: At this stage, the hip is pulled closer to the body prior to launching the body forward.
- *Left Knee*: The entire leg is placed in a suitable position for stepping by pulling the leg up utilizing the hip joint as well as the bending of the knee joint.
- *Left Ankle*: The ankle must be positioned properly in order to meet the ground after the stepping action has been carried out.

The actual stepping action involves utilizing all lower body joints to achieve the desired goal. The left foot is launched forward by direct joint control being applied to the hip and knee. At the same time, the right hip and ankle are used to propel the body forward. The following direct joint control actions are applied to the lower body section of the piped in order to carry out a stepping action with the left foot:

- *Left Hip*: The left leg is launched forward and outward in order to meet the ground after propelling the body forward.

- *Left Knee*: In order for the left leg to clear the ground, the lower part of the leg is raised by bending the left knee. During the stepping action, the leg is straightened once more in order to meet the ground.

- *Right Hip*: The backward rotation of the right hip helps propel the entire body forward due to the friction with the ground plane.

- *Right Ankle*: The right ankle also contributes to the forward propulsion by assisting in leaning the entire body forward.

Once the left foot stepping motion has been completed and the left foot meets the ground, the body is given a brief moment to balance itself then the entire process is repeated once more for the right foot. The same parameters used for the left side of the body are utilized for the right side in order to reach a cyclic behavior that would allow the biped to achieve continuous walking. The main trigger network layout for the biped walking problem is shown in Figure 6.8.

6.5 Network Evolution

The evolution of the biped trigger network is performed over multiple phases allowing the network to converge to a successful solution at each level before proceeding to the next. The most essential part in the evolution of each phase revolves around the careful structuring of the associated fitness function. For each phase, the fitness function is formulated in direct correspondence to the specific goals to be achieved by the subnetwork. This section describes the main goals behind each of the network behaviors as well as the formulation of the fitness function for achieving those goals.

Subnetwork b_1 is responsible for shifting the weight of the biped to the right foot in order to free the left foot for stepping. The motion must be accomplished in a smooth manner in order to maintain balance and prevent the biped from overshooting its desired destination which would result in a loss of balance. The following guidelines are utilized in the formulation of the fitness function governing the evolution of b_1:

- The location of the right foot must remain unchanged. A change in the location of the foot results in the application of a penalty proportionate to the distance associated with the foot displacement.

- The behavior aims to move the center of gravity of the body to the right without loss of balance. Hence, a reward is applied based on the rightward displacement of

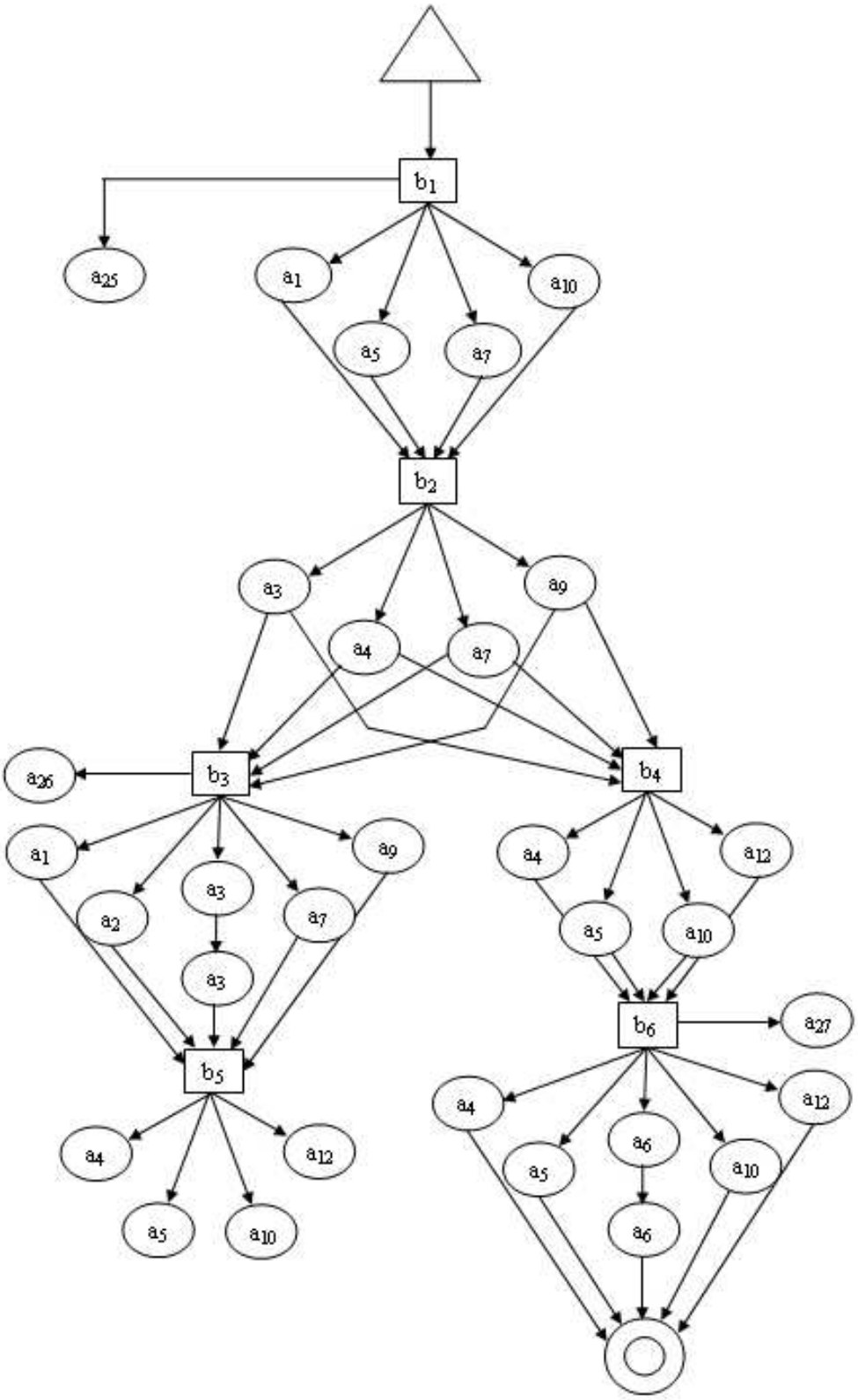


Figure 6.8: Main trigger network for biped walking motion.

the center of mass.

- Loss of balance results in the failure of the agent in achieving its goals. In order to allow for the gradual achievement of successful balancing, a reward is applied based on the number of time steps in which the agent remains balanced. A maximum time step count is selected to signify successful balancing in order to proceed with the next individual in the generation.

Given the right foot displacement, d_f, the rightward center of gravity displacement, d_c, and the number of time steps through which the agent has remained balanced, t, the fitness function is formulated as follows:

$$f = t + d_c^5 - d_f^5$$

Table 6.10 shows the parameters utilized in the evolution of the b_1 subnetwork. The same set of parameters were also used in the evolution of all other network behaviors.

Individuals:	50
Generations:	50
Crossover α	0.2
p_c	0.1
p_m	0.01

Table 6.10: Evolution parameters for biped evolution.

The genetic process was successful in converging to a configuration that fulfills the goals set forth. The agent was able to shift its entire weight to the right foot without losing balance. The center of gravity was successfully moved to the right a sufficient distance for the weight transfer to be successful without jeopardizing the agent's stability. The final values of the fitness function parameters are as follows:

t	200 (successful balancing)
d_c	2.31
d_f	0.03

The final direct joint control parameters are shown in Table 6.11. The final joint angles are given in degrees. In order to simplify the trigger network and to minimize the number of evolution variables, the execution urgency of all four actions involved was preset to a value of 2. The posture of the biped after the execution of the b_1 behavior is shown in

Action	Angle
a_1	19.73
a_5	15.35
a_7	-30.24
a_{10}	-32.07

Table 6.11: Final direct joint control parameters for behavior b_1.

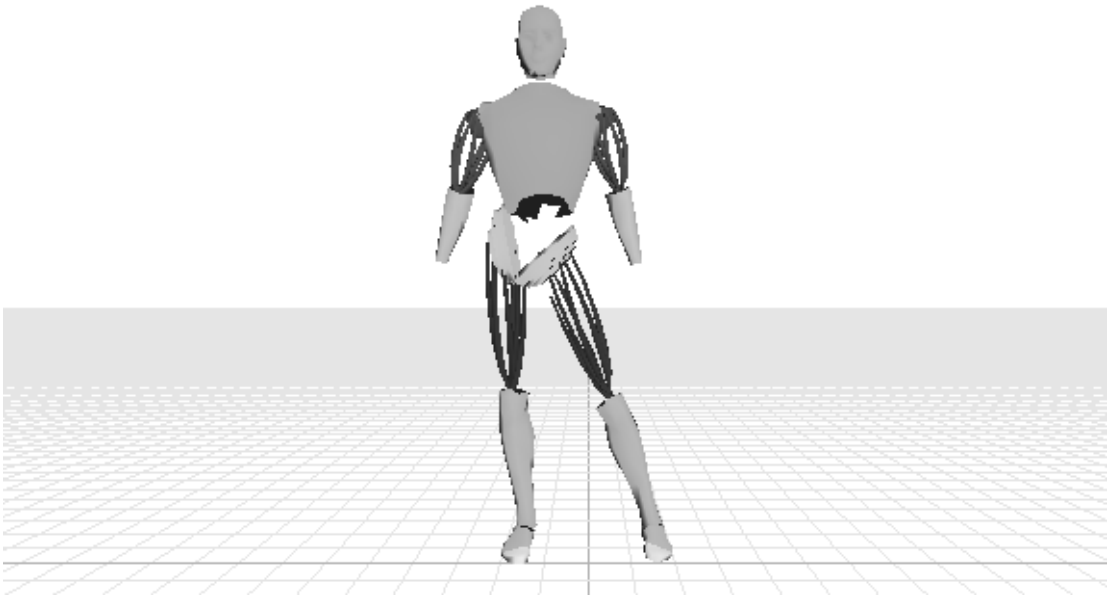


Figure 6.9: Biped posture after the execution of the evolved b_1 behavior.

Figure 6.9. We notice that the biped is now leaning to the right with all its weight on the right foot in preparation for stepping with the left foot.

Once the articulated structure is stable and balanced on the right foot, the left foot must be positioned properly in order for the stepping motion to commence. This is accomplished by pulling the leg closer to the body and bending the knee in order to clear the ground while stepping. This process must be completed while maintaining the overall balance of the structure.

The structuring of the fitness function which governs the evolution of behavior b_2 is based upon the requirement of maintaining the balance of the biped as well as minimizing the change in the position of the center of gravity. The right foot position should also remain static through out the behavior. Given the right foot displacement, d_f, and the center of gravity displacement, d_c, the fitness function is formulated as follows:

$$f = -|d_c|^5 - |d_f|^5$$

The final direct joint control parameters for behavior b_2 are shown in Table 6.12. The execution urgency of all four actions involved was preset to a value of 2. The posture of the biped after the execution of the b_1 behavior is shown in Figure 6.10.

Action	**Angle**
a_3	47.36
a_4	15.65
a_7	-24.94
a_9	-22.81

Table 6.12: Final direct joint control parameters for behavior b_2.

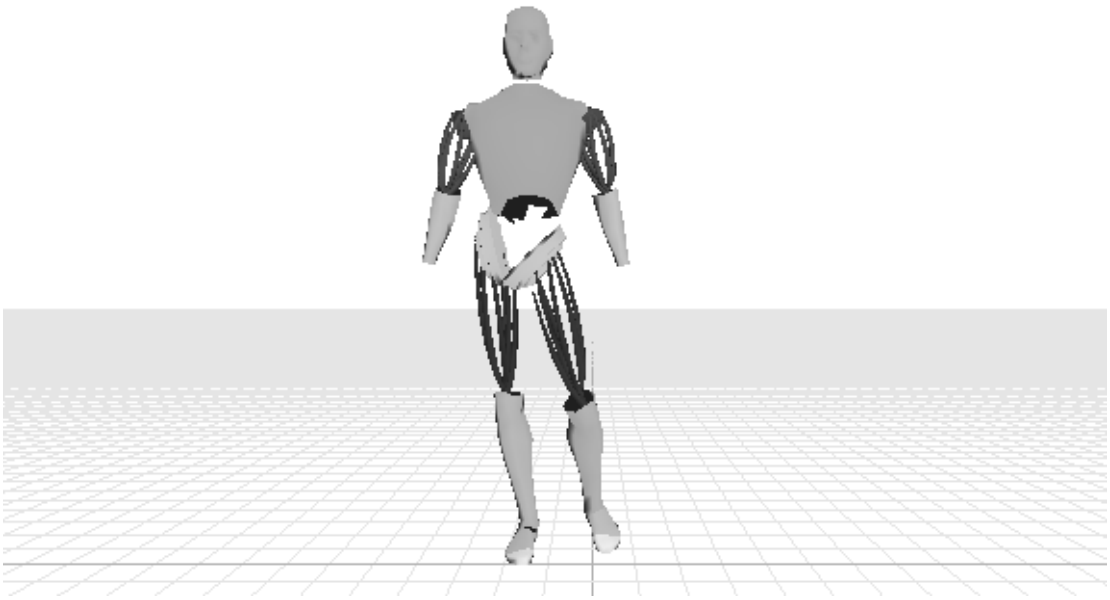


Figure 6.10: Biped posture after the execution of the evolved b_2 behavior.

Once the articulated structure is set for the walking motion to take place, a cyclic strategy must be established to utilize the left and right legs in order to achieve continuous forward motion. Behaviors b_3 and b_4 are responsible for stepping with the left foot, while behaviors b_5 and b_6 are responsible for stepping with the right. The same control parameters utilized for the stepping with the left side are to be applied to controlling the right side in a symmetric manner. This strategy reduces the complexity of the stepping problem and promotes the cyclic stability of the entire motion.

The fitness function associated with the stepping motion is based on the agent's ability to propel itself forward within a specific duration of time. In order for the motion to be successful, the balance of the biped must be maintained for the entire duration. The fitness value is assigned based on the distance by the biped with a penalty being applied in the event that the biped loses its balance. Given the biped forward displacement, d_y, and the deviation from the axis line, d_x, the fitness function, given that the biped does not lose its balance, is formulated as follows:

$$f = d_y - |d_x|^5$$

If the biped does lose its balance before the end of the stepping duration, a penalty is applied reducing the agent's fitness, and the fitness function is formulated as follows:

$$f = d_y - |d_x|^5 - 15$$

Figure 6.11 shows the evolution progression of the biped stepping motion over 50 generations of evolution using the genetic parameters listed in Table 6.10.

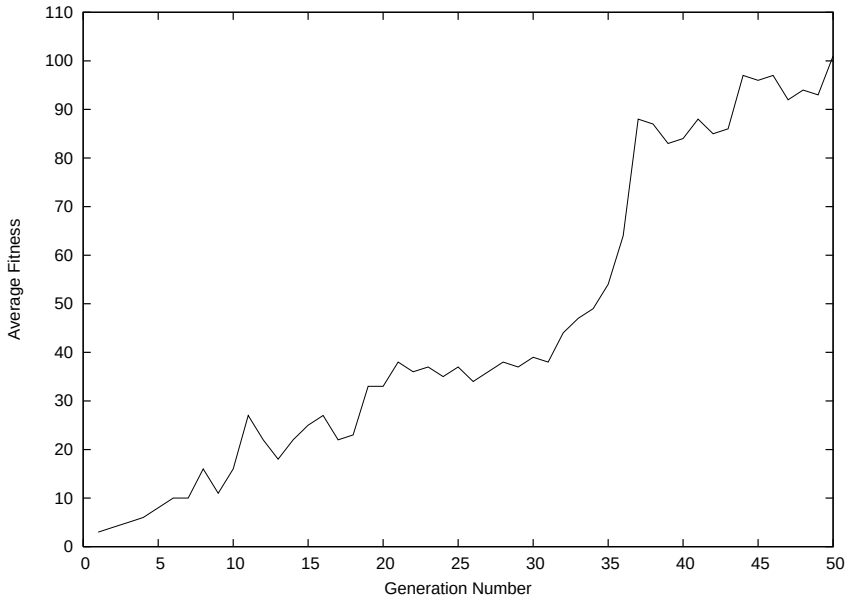


Figure 6.11: Evolution progression of the biped stepping motion over 50 generations.

The evolution progression shows a steady and gradual increase in average population fitness over the 50 generations of evolution. The biped initially would fall within the first few steps as shown in Figure 6.12, however, a strategy for successful stepping is slowly

developed through the genetic selection and evolution of trigger network configurations that exhibit the most success in the forward mobility of the biped. Figures 6.13 to 6.17 show the entire successful stepping motion of the biped.

The execution urgency of all four actions involved was preset to a value of 2. Actions a_3 and a_6 represent the knee bending in the stepping leg during the stepping motion. The bending of the knee involves a direct action to bend the knee and another to straighten it before making contact with the ground. The execution urgency of both the first and the second knee actions was set to a value of 1. The second knee angle was set to zero to guarantee the straightening of the entire leg before touching the ground. The final direct joint control parameters for the first phase of cyclic stepping behavior are shown in Table 6.13.

Action	**Angle**
a_1	.8.05
a_5	7.51
a_6	42.17
a_7	12.92
a_9	-10.13
a_{10}	-13.26
a_{12}	-9.94
a_{13}	14.79

Table 6.13: Final direct joint control parameters for the first phase of the cyclic stepping behavior.

The same set of evolved parameters are is used in the second phase of the cyclic stepping motion represented by behaviors b_5 and b_6. This involves executing the same direct control strategy on the opposite side of the biped. This strategy produces favorable results, as shown in Figure 6.11.

6.6 Conclusion

Guided genetic evolution methodologies has been shown to successfully represent and solve complex robotic control problems, including the problem of biped robot balancing and walking. The first phase of the solution involves the representation of the robot control abilities through the utilization of the trigger network structure. The utilization of trigger network encoding reduces the demands on the designer in regards to determining the exact control components required for achieving a specific result. Instead, only the main

candidate components that may contribute to achieving the preset goals are required.

The trigger network evolution algorithm allows the genetic process to choose the best components suitable for achieving the desired goals based on the performance of each agent. The careful formulation of an appropriate fitness function guarantees the elimination of control strategies that reduce the system performance while maximizing the presence of strategies that move the population closer to converging to an appropriate solution.

Although the representation and guidance methodologies allow for significant flexibility in the formulation of the control problem, the level of guidance utilized greatly affects the outcome of the genetic process as well as the precision of the results. A higher guidance level allows the genetic process to converge faster. In addition, reliable and precise guidance has been shown to significantly improve the final control results produced by the system.

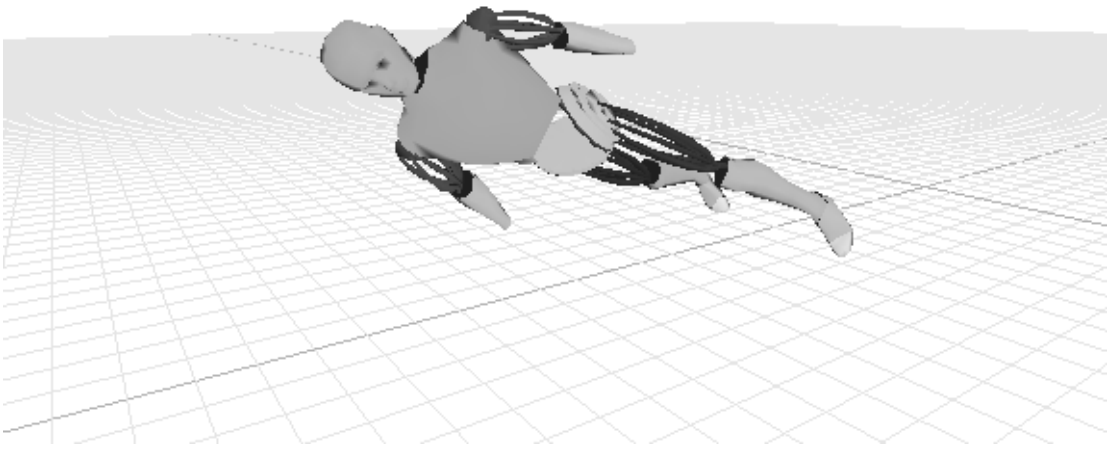


Figure 6.12: Stepping motion: biped falling during the initial phases of training.

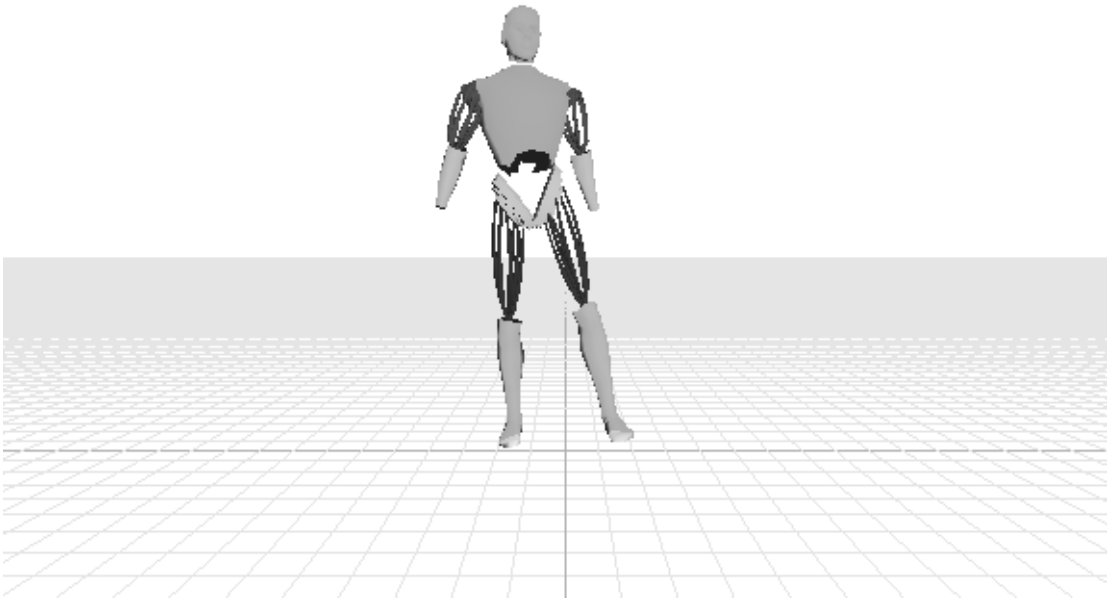


Figure 6.13: Stepping motion: beginning of stepping motion utilizing left foot.

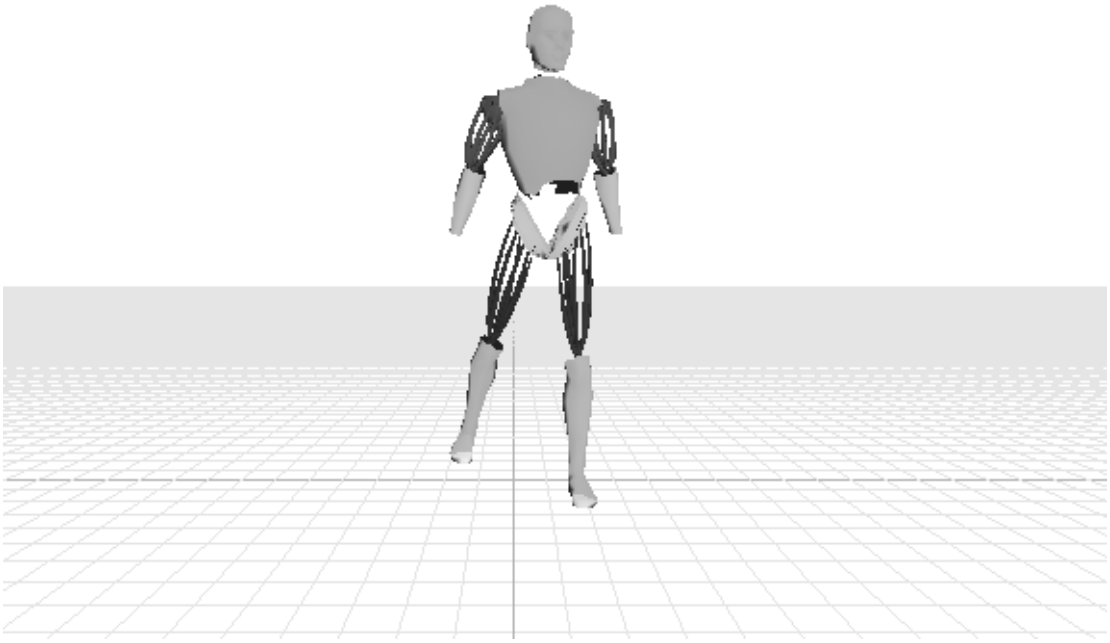


Figure 6.14: Stepping motion: left foot makes contact with the ground.

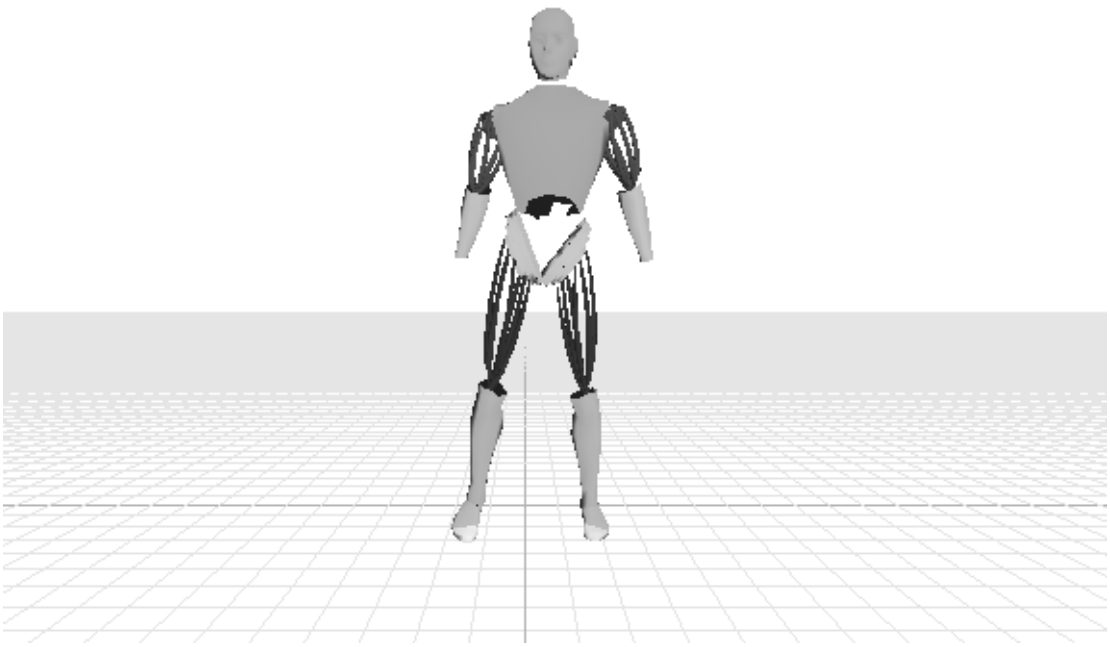


Figure 6.15: Stepping motion: continued stepping utilizing right foot.

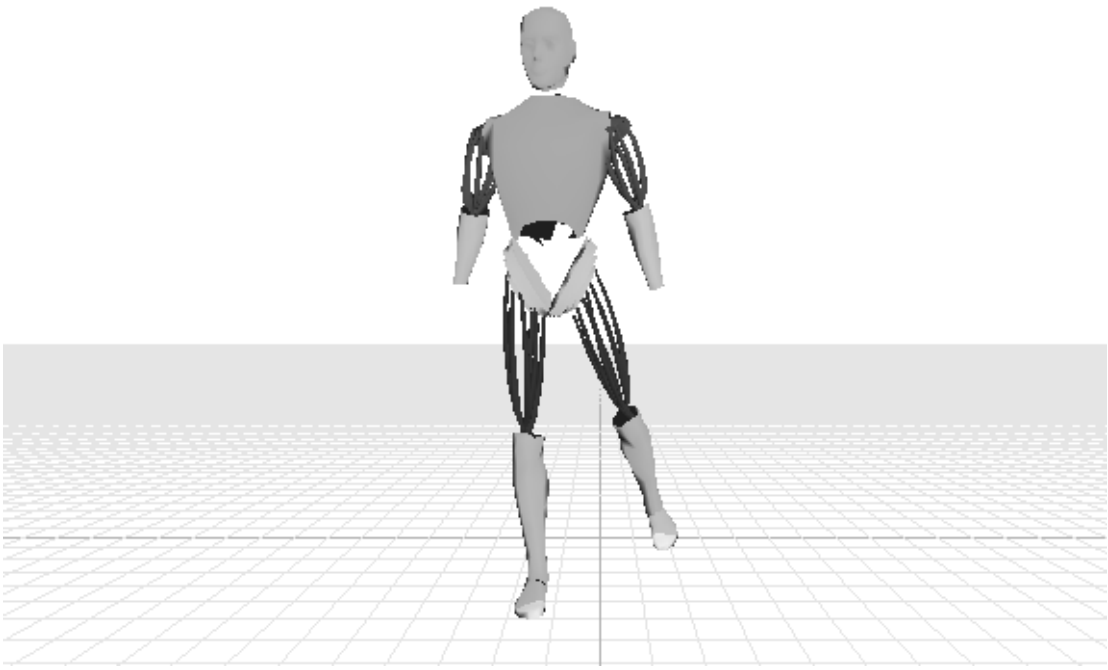


Figure 6.16: Stepping motion: right foot makes contact with the ground.

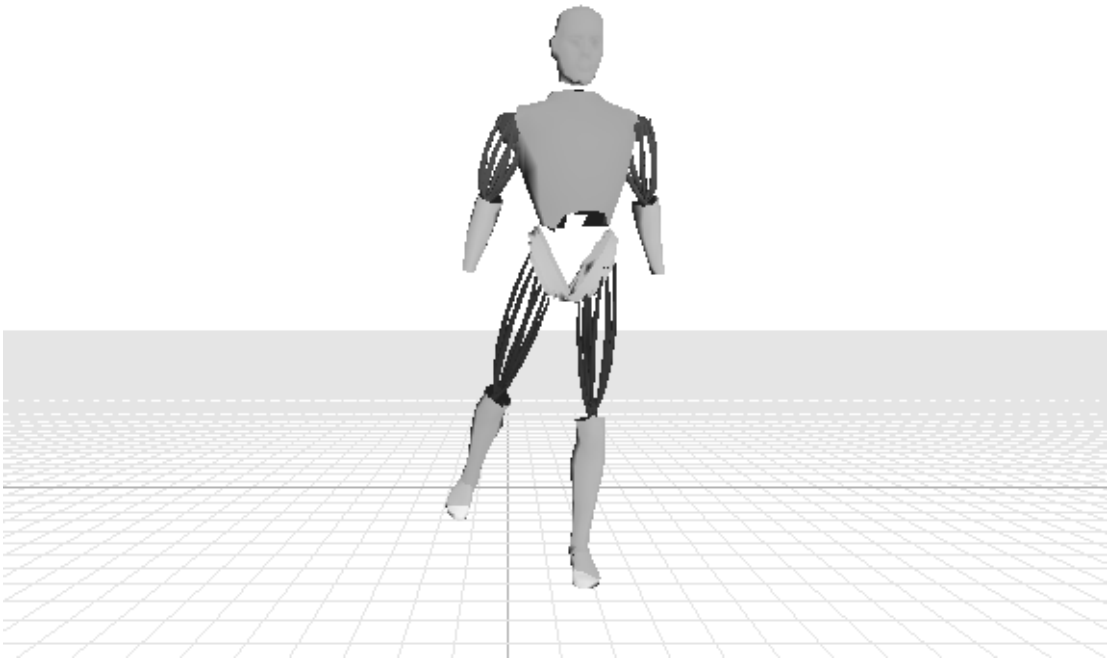


Figure 6.17: Stepping motion: continued stepping utilizing left foot.

Chapter 7

CONCLUSIONS AND FUTURE WORK

This chapter summarizes the methodologies presented in this dissertation in relation to the representation and evolution of robotic controllers for the real-time control of articulated robotic structures. The chapter will also cover the challenged faced by the proposed framework as well as suggestions for future work.

7.1 Summary of Work

Autonomous robotic control is an extremely intriguing area of study that could pave the way for many beneficial applications. However, the control of articulated structures has proven to be a problem of extreme difficulty due to the complex nature of the internal dynamics of the robot as well as the complexity of the interactions between the robot and the environment. In order to conquer such complexity, an organized algorithmic framework is needed for the efficient encoding and training of robotic controllers in a manner that circumvents some of the existing hurdles.

This dissertation has explored and addressed the problem of articulated robotic control through three main contributions:

1. A new connectionist model, labeled *Trigger Networks* was created for the encoding of agent attributes and control capabilities. The model offers a high level descriptive structure for the representation of control strategies at any level of sophistication for the control of articulated robots. Trigger networks offer a time-based model for the description of execution sequencing as well as control urgency associated with each of the robotic joints. The network structure has proven successful in the representation of robotic behaviors associated with robotic arms, four-legged robots, as well as biped robots.

2. A genetic evolution algorithm was formulated for the evolution of trigger networks based on one or more fitness functions associated with the desired behaviors. The algorithms presented as part of the evolution framework allows for the processing of trigger networks through genetic selection, crossover, and mutation operators over multiple generations in an effort to achieve successful fulfillment of the preset

behavioral goals. The genetic algorithms proposed and implemented have proven to be highly successful in training different types of controllers for performing complex tasks through the utilization of joint control.

3. Mechanisms for guiding the genetic process have been formulated in order to reduce the network convergence time and increase the quality of the produced results. The guidance methodologies have been proven to systematically increase the overall fitness of individuals as well as reduce the evolution time required for achieving the desired evolution goals.

7.2 Limitations of the Proposed Framework

Through the utilization of guided genetic evolution, different classes of robots have been trained to successfully perform desired tasks with a considerable level of accuracy. In order to transfer the training methodologies to real robots outside of the simulation environment, a higher level of uncertainty has to be introduced into the training algorithms in order to create agents that are more reliably fit for real environments. Real world parameters introduce a significant level of unpredictability to the control equation. In order for the simulated environment to better represent real-world dynamics, a level of noise may be included as part of the input signal acquisition and joint control modules. The introduction of noise and the evaluation of its effect on the agent behavior would be essential for the creation of robotic agents that behave reliably in real-world environments.

The implementation of genetic training methodologies requires a significant amount of computational power in order to achieve results within a reasonable duration of time. In order to evolve controllers utilizing low computational power, a significant amount of guidance must be applied to the trigger network to reduce the complexity of the problem being evolved. However, the application of excessive network guidance restricts the evolutionary path of the system and limits the search space to a degree that might produce harmful results. A balance must be achieved between the network guidance and computing power utilization in order to maintain the essence of the genetic process and achieve the desired training goals.

7.3 Future Directions

Several areas of the guided genetic evolution framework require further investigation:

1. The introduction of noise in the evolution framework should be investigated in order to achieve higher levels of reliable fitness of the evolved robotic agents.

2. Alternative evolutionary means should be examined as possible candidates for the enhancement of trigger network evolution.

3. The combination of trigger networks, genetic evolution, and neural methodologies should be investigated as a possible grouping that could enhance the training process.

4. Additional testing should be performed on trigger network evolution. Such testing may introduce areas of possible enhancements and modifications.

5. The utilization of agent state determination and transitioning, on a larger scale, should be investigated as a tool that could enable a higher level of robotic control. This is achieved through awareness of robotic states, the ability to apply intelligent transitions between known states, as well as the ability to acquire knowledge of new states that were previously unknown to the agent.

Appendix A

RIGID BODY ARTICULATION

A.1 Introduction

In order to allow for the evaluation and optimization of the evolutionary techniques presented in this thesis, a need existed for the development of a simulation environment based on computational models of the entities being studied. The simulation environment offers a crucial visual dimension through which the various aspects of the system can be monitored and enhanced. Since this study revolves around the intelligent control of complex articulated bodies, a comprehensive physical simulation system is needed to model the different system components as well as their interactions. The simulation physics engine is responsible for the numeric calculations and transformations determining the position and orientation of bodies as they interact with each other and the environment. The engine also works on maintaining the physical constraints that govern the different applications of the forces and torques being exherted on the articulated structures. The following list demonstrates the essential parts of the physics engine needed for achieving the desired simulation goals:

- Particle State Management: The environment has to offer persistence of particle states. The physical properties of the bodies need to be accurately represented and utilized in order to produce accurate simulation models.

- Numeric Integration: In order to evolve the system from one state to another, numerical methods are utilized to perform the needed calculations and transformations. The numerical errors produced by the methods being utilized has to be considered in order to achieve the accuracy requirements set forth by the simulation.

- Geometric Body Representation: Geometric structures are tied to simulation bodies in order to achieve accurate representation of the entities being evolved. Primitive geometric shapes as well as more complex polyhedra-based models maybe utilized as representation of articulated bodies.

- Collision Detection and Response: The ability to enforce non-penetrating constraints within the system is crucial. Hence, the engine has to provide means for

detecting collisions between objects as well as the numerical methods required for resolving such collisions with physical accuracy.

- Joint Constraints: In addition to non-penetrating constraints, joint-based constraints that allows for the modeling of articulated structures is needed. Different types of joints may be used on different parts of the structure. At a minimum, the system is to offer support for hinge, universal, and ball and socket type joints.

- Dynamic Force and Torque Control: Finally, the ability to dynamically apply forces and torques to different parts of the articulated structures is needed for the simulation of muscle or actuator controls. The system is to offer tools for applying such elements at different levels depending on the desired movement being simulated.

A.2 Rigid Body Kinematics

A system based on rigid body dynamics is suitable for the core design of the physical simulation engine needed. A rigid body can be viewed as being composed of a system of particles. Since the general shape of the body is rigid, the particles do not migrate within the body, so the mass within the body remains consistently distributed. This important characteristic allows for the analysis of motion of a rigid body using only the linear motion of its center of mass as well as the angular motion about its center of mass. In essence, the body can be treated as a single particle within the environment instead of accounting for the various particles that make up the body. Consequently, the system gains performance by ignoring deformation calculations and low level particle management without harming the accuracy of the produced results. Soft contact constraints could be applied to simulate the elasticity of collisions, which in reality would occur due to object deformations.

In addition to the motion characteristics, the physical characteristics of a rigid body is also essential for the implementation of collision detection and resolution. The geometric structure of an object as well as its inertial properties[1] are used to describe the shape of a body as well as its internal structure which affects its mass distribution.

This discussion on rigid body dynamics refers to and condenses some of the content presented by Baraff [5],[6],[7],[8],[9],[10],[11], Eberly [**?**], Lengyl [78], Bergen [15] and Bourg[18], as well as other content with references added where appropriate. Baraff gives a very thorough and detailed presentation of physically based modeling and rigid body dynamics. Eberly expands the discussion to cover the simultaneous numeric resolution of

[1] The body's inertial properties are based on the associated geometric structure and mass distribution

multiple contact points using Linear Complementarity. Lengyl presents essential mathematical topics relating to computer simulations, while Bergen offers a thorough coverage of collision detection techniques and their applications in physical simulation.

A.2.1 Position and Orientation

A rigid body can undergo rotation and translation, so given the geometric shape of the body, we use the vector $x(t)$ and the 3 x 3 rotation matrix $R(t)$ to describe the position and orientation of the body respectively. The vector $x(t)$ is used to translate the body to the proper world-space position. The vector describes the position of the body's center of mass in space at time t. Since each body is represented as a single particle, this single vector is sufficient for describing the body's position. The geometric shape of the body is attached to its center of mass, so the polygonal structure of the body is rendered relative to its position given by $x(t)$.

Similarly, $R(t)$ is used to orient the body transforming the associated geometric shape from body space into world space as seen in Figure A.1. This step is necessary for detecting collision and determining collision points within the simulation. Several methods can be used to describe the orientation of the body. The rotation matrix and quaternion representations are the most common and most efficient methods for representing rotation.

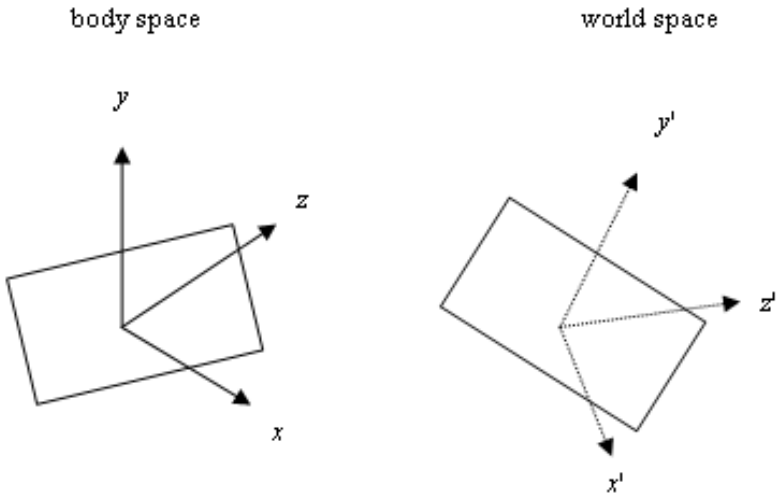


Figure A.1: The axes of the body represented by *x*,*y* and *z* are transformed to x',y' and z' where $x' = R(t)x$, $y' = R(t)y$ and $z' = R(t)z$.

Given the orientation heading ϕ, attitude θ, and bank ψ, the orientation matrix $R(t)$ is defined as:

$$R(t) = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{A.1}$$

Where:

$$\begin{aligned} r_{11} &= \cos(\Phi)\cos(\Theta) \\ r_{12} &= \sin(\Phi)\sin(\psi) - \cos(\Phi)\sin(\Theta)\sin(\psi) \\ r_{13} &= \cos(\Phi)\sin(\Theta)\sin(\psi) + \sin(\Phi)\cos(\psi) \\ r_{21} &= \sin(\Theta) \\ r_{22} &= \cos(\Theta)\cos(\psi) \\ r_{23} &= -\cos(\Theta)\sin(\psi) \\ r_{31} &= -\sin(\Phi)\cos(\Theta) \\ r_{32} &= \sin(\Phi)\sin(\Theta)\cos(\psi) + \cos(\Phi)\sin(\psi) \\ r_{33} &= -\sin(\Phi)\sin(\Theta)\sin(\psi) + \cos(\Phi)\cos(\psi) \end{aligned} \tag{A.2}$$

Unit quaternions may also be utilized for the representation of rotation. The use of a quaternion is advantageous over the use of a rotation matrix as it stores the rotation data using four components, whereas the rotation matrix representation uses nine parameters. Reducing the number of parameters avoids the need for costly alignment to compensate for matrix drift. This drift accumulates within the simulation as a result of performing operations on matrices using finite point precision. In order for a quaternion to define rotation, it must be unit length. However, error buildup in the quaternion calculations can result in variations in its length. This can be corrected by normalizing the quaternion which is significantly less costly that combating matrix drift.

Given the orientation heading ϕ, attitude θ, and bank ψ, the quaternion q is defined as:

$$q = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} \cos\left(\frac{\phi}{2}\right)\cos\left(\frac{\Theta}{2}\right)\cos\left(\frac{\Psi}{2}\right) + \sin\left(\frac{\phi}{2}\right)\sin\left(\frac{\Theta}{2}\right)\sin\left(\frac{\Psi}{2}\right) \\ \sin\left(\frac{\phi}{2}\right)\cos\left(\frac{\Theta}{2}\right)\cos\left(\frac{\Psi}{2}\right) - \cos\left(\frac{\phi}{2}\right)\sin\left(\frac{\Theta}{2}\right)\sin\left(\frac{\Psi}{2}\right) \\ \cos\left(\frac{\phi}{2}\right)\sin\left(\frac{\Theta}{2}\right)\cos\left(\frac{\Psi}{2}\right) + \sin\left(\frac{\phi}{2}\right)\cos\left(\frac{\Theta}{2}\right)\sin\left(\frac{\Psi}{2}\right) \\ \cos\left(\frac{\phi}{2}\right)\cos\left(\frac{\Theta}{2}\right)\sin\left(\frac{\Psi}{2}\right) - \sin\left(\frac{\phi}{2}\right)\sin\left(\frac{\Theta}{2}\right)\cos\left(\frac{\Psi}{2}\right) \end{bmatrix} \tag{A.3}$$

A.2.2 Linear and Angular Velocity

Given $x(t)$ and $R(t)$ defining the position and orientation of the body at time t, we define $\dot{x}$ and $\dot{R}(t)$ to describe how the position and orientation of the body change over time. We also define the vector $v(t) = \dot{x}$ to describe the velocity of the translation of $x(t)$ over time. The vectors $x(t)$ and $v(t)$ are defined by the following formula:

$$v(t) = \frac{d}{dt}x(t)$$

.

We also define the vector $\omega(t)$ to describe the axis about which the body rotates. The rate of rotation is described by the magnitude of $\omega(t)$. The relationship between $R(t)$ and $\omega(t)$ is not as straight forward as that of $x(t)$ and $v(t)$. The rate of change of the body's orientation is defined as the product of the vector $\omega(t)$ and the matrix $R(t)$.

$$\dot{R}(t) = \omega(t) * R(t) \tag{A.4}$$

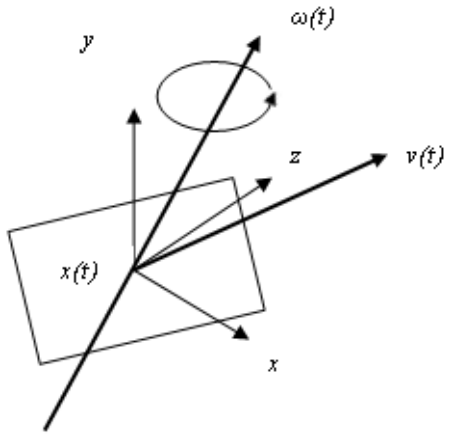


Figure A.2: Linear velocity, represented by $v(t)$, and angular velocity, represented by $\omega(t)$, of a rigid body.

A.2.3 Linear and Angular Momentum

Linear momentum is the tendency of a body moving in a certain direction to maintain its speed and direction of motion. The linear momentum $P(t)$ of a particle is defined as the product of its mass M and velocity $v(t)$.

$$P(t) = Mv(t) \tag{A.5}$$

and since M is a constant, $\dot{v}(t) = \frac{\dot{P}(t)}{M}$.

The concept of linear momentum also allows for the expression of the effect of the total force $F(t)$ applied on a rigid body. The change in linear momentum $P(t)$ is equivalent to the value of the force applied.

$$\dot{P}(t) = F(t) \tag{A.6}$$

The angular momentum of a rigid body is defined as the product of the moment of inertia and the angular velocity. Simply put, as a body spins about a particular axis, angular momentum is the tendency of a rotating body to maintain its rotational speed and axis. If no external torque is applied, the angular momentum of the body is conserved. Given the angular velocity $\omega(t)$ and the body's *inertia tensor* $I(t)$, which is a 3x3 matrix that describes the body's mass distribution relative to its center of mass, the angular momentum $L(t)$ is defined as

$$L(t) = I(t)\omega(t) \tag{A.7}$$

Analogous to the relation between linear momentum and force, we obtain the same result for angular momentum $L(t)$ and torque $\tau(t)$.

$$\dot{L}(t) = \tau(t) \tag{A.8}$$

A.2.4 The Inertia Tensor

The inertia tensor $I(t)$ is a 3 x 3 matrix that describes the body's mass distribution and how it is affected by angular velocity. The inertia tensor can be considered a scaling factor between the body's angular momentum $L(t)$ and angular velocity $\omega(t)$. $I(t)$ is usually computed in body space and then transformed to world space to carry out calculations. It is defined as the matrix:

$$I = \begin{bmatrix} \int (y^2+z^2)dV & -\int (xy)dV & -\int (xz)dV \\ -\int (xy)dV & \int (x^2+z^2)dV & -\int (yz)dV \\ -\int (xz)dV & -\int (yz)dV & \int (x^2+y^2)dV \end{bmatrix} \tag{A.9}$$

The integrals are over the volume of the body. If the body is a simple primitive shape with evenly distributed mass, then the elements of $I(t)$ have a simple closed form solution. However, if the shape or mass distribution of the body is more complex, numerical methods are required to accurately calculate the inertia tensor.

A.2.5 Body State Vector

Now that we have concluded the discussion of the different components that constitute the state of any body within our simulation, we define the state of a body[2] at time t as a vector X(t) which is defined by the body's center of mass position x(t), orientation matrix R(t), linear momentum P(t) and angular momentum L(t).

[2] as defined by Baraff[11]

$$X(t) = \begin{bmatrix} x(t) \\ R(t) \\ P(t) \\ L(t) \end{bmatrix} \tag{A.10}$$

At time t, the auxiliary quantities, inertia tensor $I(t)$, angular velocity $\omega(t)$ and linear velocity $v(t)$ can be computed as [32]

$$\begin{aligned} I(t) &= R(t) I_{body} R^T(t) \\ \omega(t) &= I^{-1}(t) L(t) \\ v(t) &= P(t)/M \end{aligned} \tag{A.11}$$

where I_{body} is the inertia tensor in body space and M is the mass of the body.

A.2.6 Integrating the Equations of Motion

The Euler integration formula utilizes a Taylor series expansion allowing for the approximation of the particle state vector at time t+dt given the state vector at time t as well as its derivative. For example, given the particle displacement at time t and the time step dt , the new displacement can be reformulated as:

$$x_{n+1} = x_n + dt \cdot v(t_n, x_n) \tag{A.12}$$

Formula A.12 yields a first-order approximation since only the first derivative is included in the calculation. The Euler method has an error of order dt^2. The error introduced by Euler's basic methods can be reduced by using more terms in the Taylor series. In order to overcome the difficulty associated with being able to determine the second, third, fourth and higher derivatives of the function being integrated, additional Taylor series expansions can be utilized to approximate the needed derivatives, then the approximated values can be substituted back into the original expansion.

The second-order Runge-Kutta method utilizes an Euler-like trial step to the midpoint of the interval and then uses the results at the mid-point to complete the step. The Runge-Kutta method is calculated as follows:

$$\begin{aligned} k_1 &= dt \cdot v(t_n, x_n) \\ k_2 &= dt \cdot v(t_n + \tfrac{1}{2} dt, x_n + \tfrac{1}{2} k_1) \\ x_{n+1} &= x_n + \tfrac{1}{2}(k_1 + k_2) \end{aligned} \tag{A.13}$$

In order to use the expansion methods mentioned, the derivative of the state vector is computed as [32]

$$\frac{d}{dt}X(t) = \begin{pmatrix} v(t) \\ \omega(t) * R(t) \\ F(t) \\ \tau(t) \end{pmatrix} \tag{A.14}$$

where $F(t)$ is the total force and $\tau(t)$ is the total torque acting on the body at time t.

A.3 Contact Forces

In a typical physical simulation environment, it is necessary to implement constraints that prevent bodies from inter-penetrating as they move around the simulation space. This is accomplished by implementing collision detection between objects, and then responding to the collision in a manner that maintains the constraints. Collision response involves the calculations of new body states as a consequence to the collisions that take place. Two commonly used approaches for implementing collision response are: analytical methods and non-analytical methods (also known as penalty methods).

Collision response utilizing analytical methods formulates and solves Newtonian equations producing exact results. Typically, the equations require fewer time steps than other methods for the solution to be produced. As stated by Baraff [5], the disadvantage of analytical methods stems from the difficulty involved in formulating and implementing them.

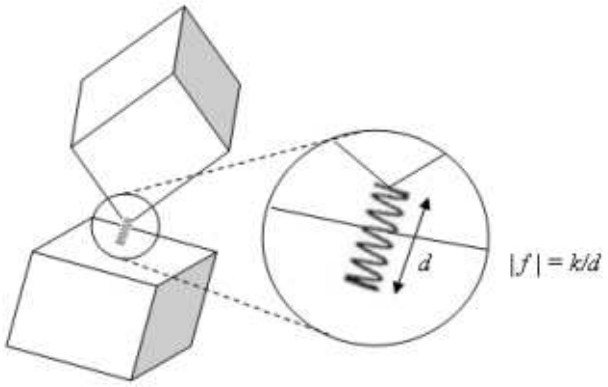


Figure A.3: The penalty method utilizes the insertion of springs at the contact points to force colliding bodies apart.

Instead of producing solutions to equations of motion, as shown in figure A.3, the penalty methods relies on the insertion of temporary springs to separate objects at the points of contact. The springs apply forces of equal magnitude but opposite directions

on both objects involved in the collision. Given the spring constant K and the distance between the colliding objects d, the magnitude of force applied is calculated as follows: $|f| = k/d$. Hence the distance between the objects and the force applied are inversely proportional. As the separation between the two objects decrease, the resulting separating force applied on each object increases pushing the objects apart. The penalty methods for collision response are easy to understand and implement, however, the results produced by the methods are not exact and are computationally expensive, specially for stiffer springs.

A.3.1 Analytical Contact Response

Suppose a collision occurs between the two bodies A and B as they come in contact at time t_0, let p define the contact point in world space [3]. The contact point p corresponds to the contact points on the two bodies p_a and p_b respectively, where $p_a(t_0) = p_b(t_0) = p$. Let $\dot{p}_a(t_0)$ and $\dot{p}_b(t_0)$ be the velocity of p_a and p_b at time t_0. The velocity values are defined as

$$\dot{p}_a(t_0) = v_a(t_0) + \omega_a(t_0) \times (p_a(t_0) - x_a(t_0)) \tag{A.15}$$

and

$$\dot{p}_b(t_0) = v_b(t_0) + \omega_b(t_0) \times (p_b(t_0) - x_b(t_0)) \tag{A.16}$$

In order to determine the type of contact taking place, the relative velocity of the contact points must be calculated. Let v_{rel} represent the relative velocity of p_a and p_b defined as

$$v_{rel} = \hat{n}(t_0) \bullet (\dot{p}_a(t_0) - \dot{p}_b(t_0)) \tag{A.17}$$

Three different types of contacts may exist between two bodies depending on the relative velocity of the contact points:

- Colliding contact: points are moving towards each other signified by

$$v_{rel} < -tolerance$$

.

[3] A typical collision might produce multiple contact points between the two bodies. Linear methods for resolving multiple contact point simultaneously will be discussed

- Separating contact: points are moving away from each other signified by

$$v_{rel} > tolerance,$$

  hence no further action is required.

- Resting contact: points are at rest relative to each other signified by

$$|v_{rel}| <= tolerance.$$

A.3.2 Colliding Contacts

We start by considering a frictionless collision along the line connecting the centers of mass. Let J be the linear impulse involved in the collision, v_+ is the post-collision velocity, while v_- is the pre-collision velocity. J is represented as [18]

$$J = m(v_+ - v_-) \tag{A.18}$$

The coefficient of restitution e if represented as

$$e = -(v_{a+} - v_{b+})/(v_{a-} - v_{b-}) \tag{A.19}$$

Since the impulse force acts on both bodies with the same magnitude yet opposite directions, the following three equations are available

$$\begin{aligned} J &= m_a(v_{a+} - v_{a-}) \\ -J &= m_b(v_{b+} - v_{b-}) \\ e &= -(v_{a+} - v_{b+})/(v_{a-} - v_{b-}) \end{aligned} \tag{A.20}$$

Let $v_{rel} = (v_{a-} - v_{b-})$. Given the three unknowns v_{a+}, v_{b+} and J, we can rearrange the three equations to achieve the following formula for J.

$$J = -v_{rel}(e+1)/(1/m_a + 1/m_b) \tag{A.21}$$

Since most collisions are not frictionless and are not along the line connecting the centers of mass, we expand our discussion to cover the angular impulse force as well. Given v_{rel} as the relative velocity along the line of contact, and $\mathbf{n}$ as the normal vector at the contact point pointing out from the first body, $\mathbf{r_a}$ is the vector from the center of mass of the first body to the point p_a and $\mathbf{r_b}$ is the vector from the center of mass of the second body to the point p_b, the formula for J taking into accounts both linear and angular forces is formulated as follows:

$$J = \frac{-v_{rel}(e+1)}{1/m_a + 1/m_b + \mathbf{n}\cdot[(\mathbf{r_a}\times\mathbf{n})/\mathbf{I_a}]\times\mathbf{r_a} + \mathbf{n}\cdot[(\mathbf{r_b}\times\mathbf{n})/\mathbf{I_b}]\times\mathbf{r_a}} \tag{A.22}$$

Given the formula for J, we can calculate the post-collision linear and angular velocities of the bodies involved. The linear velocities are calculated as

$$\begin{aligned} v_{a+} &= v_{a-} + (J\mathbf{n})/\mathbf{m_a} \\ v_{b+} &= v_{b-} + (-J\mathbf{n})/\mathbf{m_b} \end{aligned} \tag{A.23}$$

The angular velocities are calculated as

$$\begin{aligned} \omega_{a+} &= \omega_{a-} + (\mathbf{r_a}\times J\mathbf{n})/\mathbf{I_a} \\ \omega_{b+} &= \omega_{b-} + (\mathbf{r_b}\times -J\mathbf{n})/\mathbf{I_b} \end{aligned} \tag{A.24}$$

A.3.3 Friction

Most collisions do not demonstrate direct impact between the colliding bodies, so as the bodies come into contact, frictional forces are exerted on the bodies in a direction that is tangential to the contacting surfaces. The frictional forces will contribute to a change in both the linear and angular velocities of both bodies. The tangential forces produced by friction is proportional to the normal force exerted on the contact surface. Consider F_f as the friction force and F_n as the force normal to the surface, the coefficient of friction μ is defined as

$$\mu = F_f/F_n$$

.

In addition to changing the linear velocity of the body, frictional forces also change the angular velocity by creating a torque on the bodies about the center of gravity. The impulse force due to friction is given by

$$Impulse = I/(\mu r)(\omega_+ - \omega_-) \tag{A.25}$$

Finally, taking friction into consideration, the change in linear velocity is given by

$$\begin{aligned} v_{a+} &= v_{a-} + (J\mathbf{n} + (\mu\mathbf{J})\mathbf{t})/\mathbf{m_a} \\ v_{b+} &= v_{b-} + (J\mathbf{n} + \mu\mathbf{J})\mathbf{t})/\mathbf{m_b} \end{aligned} \tag{A.26}$$

and the change in angular velocities of the two colliding bodies is given by

$$\begin{aligned} \omega_{a+} &= \omega_{a-} + (\mathbf{r_a}\times(J\mathbf{n} + (\mu\mathbf{J})\mathbf{t}))/\mathbf{I_a} \\ \omega_{b+} &= \omega_{b-} + (\mathbf{r_b}\times(-J\mathbf{n} + (\mu\mathbf{J})\mathbf{t}))/\mathbf{I_b} \end{aligned} \tag{A.27}$$

where $\mathbf{t}$ is a unit vector tangent to the collision surfaces.

A.3.4 Resting Contacts

Calculating the response forces for resting contacts is considerably more complex than the calculations for colliding contacts. The main reason for the complexity lies in the fact that all contact forces have to be calculated simultaniously as one contact force might influence the resolution at another point. The same case also applies when multiple bodies are involved in the collision yielding multiple contact points. Figure A.4 shows four rigid bodies producing six contact points. The collision normals, and the centers of mass are also shown in the figure. If all points are not processed simultaniously, a deadlock situation can be reached where the response engine halts as one of the bodies oscilates back and forth being pushed by one contact point resolution and then the other.

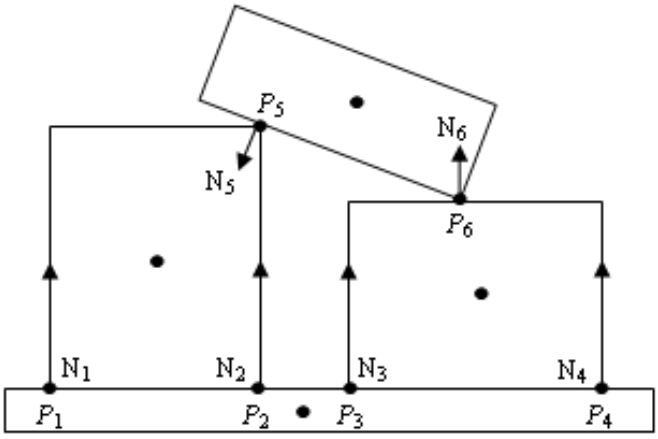


Figure A.4: Four rigid bodies producing six contact points.

The simultanious processing of multiple contact points can be formulated as a Linear Complementarity Problem (LCP), which revolves around finding a solution to a linear system satisfying specific inequality constraints. Given the $n \times n$ matrix M and the n-vector $\mathbf{q}$, an LCP presents the problems of calculating values for the variables $z_1, z_2, .., z_n$ and $w_1, w_2, .., w_n$ such that [26]

$$\begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} = M \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{pmatrix} + q \tag{A.28}$$

The following list presents the steps to be followed for handling collisions producing multiple contact points[33]:

1. The collision detection engine produces data for all contact points present at the current state of the system.
2. The contact points are pre-processed for formulation as a Linear Complementarity Problem.
3. The LCP solver produces a system solution guaranteeing the no-penetration constraint.
4. The post-collision velocities are calculated based on the results of the LCP solver.
5. The differential equation solver calculates the new state vector for each body within the simulation.
6. Repeat for the next timestep.

A.4 Joint Constraints

In order to create a complex articulated structure out of rigid-body components, a connection mechanism is needed that allows for the creation of Joints that hold the bodies together through the enforcing of different types of constraints. Similar to the impulses generated to enforce non-penetration constraints, impulses can be introduced to limit the relative motion of bodies. Schmitt [115] introduces a method for conserving joint constraints through the application of continous position and orientation correcting pulses. Schmitt's algorithm offers a simpler approach to conserving constraints, as it does not require solutions to complex differencial equations (Figure A.5).

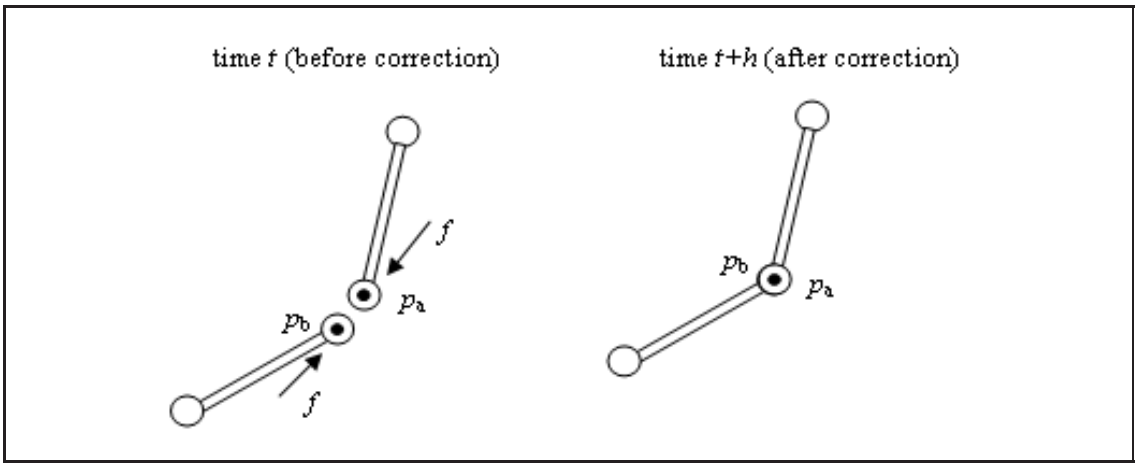


Figure A.5: Joint correction forces to conserve joint constraints.

Given the two bodies *A* and *B* connected via a ball-and-socket joint. The points p_a and p_b represents the anchor points on the two bodies, and the joint constraint is to keep the two anchor points within tolerance distance of each other. As the two points slide apart due to forces being exerted on the bodies, a corrective impulse force has to be applied to pull the bodies back together.

The impulse forces are calculated to reduce the distance between p_a and p_b to zero at time $t+h$. The velocity changes of A and B must equal the separating distance d divided by the time step size h. The corrective impulses can be computed accordingly and applied to the system at time t in order to achieve a consistent system at time $t+h$. For multiple joints, the process must repeat iteratively until all constraints within the system are satisfied.

In addition to the method presented, Lagrange multipliers could also be used to solve for the motion of bodies connected by joints. A discussion of such a solution is presented by Baker [119].

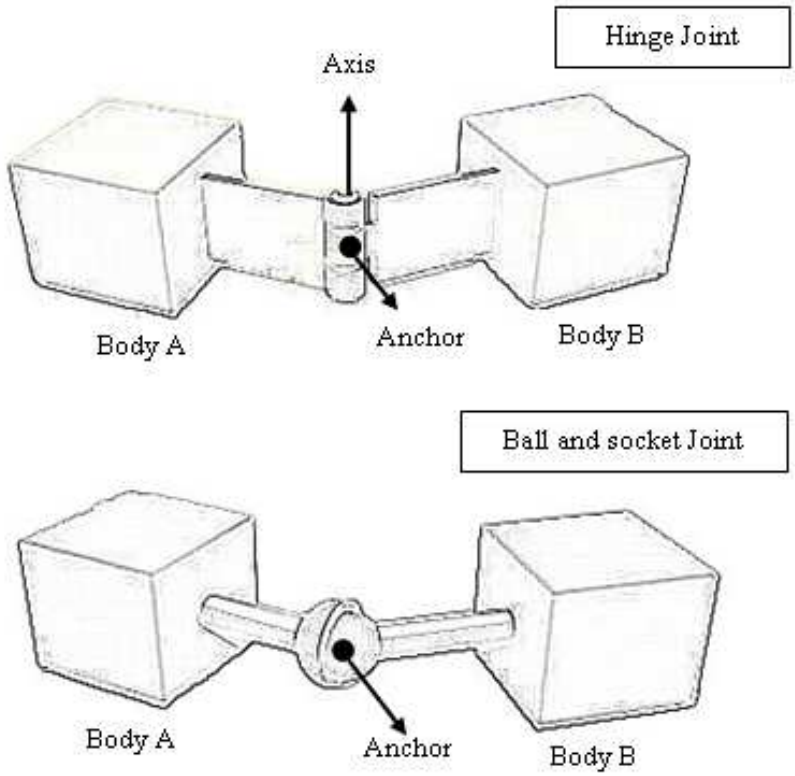


Figure A.6: Hinge as well as ball-and-socket joint constraints connecting the two bodies A and B.

In order to create the desired articulated structures, the simulation engine, at a mini-

mum, must have support for two different types of Joints. The first is a hinge joint offering a single degree of freedom about the hinge axis, and the second is a ball-and-socket joint offering three degrees of freedom. Both types of joints are shown in figure A.6[124]. The different joints may be used to simulate different connection types within the structure. For example, in the simulation of a biped robot, a hinge joint may be used to simulate the elbow and knee, while a ball-and-socket joint may be used to simulate the shoulder or hip.

The efficient utilization of joint constraints would allow for the simulation of any type of articulated structure. Ropes and fabric may be simulated through the placement of proper joints and relaxing the corrective impulses to simulate spring-like behavior at the contact points producing stretching. The simulation of fluids is also feasible using the same system by structuring a mesh of bodies connected by ball-and-socket joints and formulating a model governing the different fluid forces and particle motion. Cohesion forces may also be simulated with the inclusion of breaking threshholds at which the particles lose their connectivity.

Appendix B

AGENT AWARENESS AND PLANNING

B.1 Situation Awareness

Situation awareness revolves around the accurate perception of both the internal and external environments of an agent and understanding the value of the perceived information presently as well as for future planning. The notion of *forward thinking* allows a robotic agent to prepare for future events based on proactive and anticipatory actions [125]. Being fully aware of the current state is closely coupled with an agent's ability to create goals and execute goal-based strategies. An agent operating in a real-world environment will be compelled to autonomously create and follow its own goals as the environment could neither be modelled correctly nor completely in advance [98]. Thus, the dynamic creation, re-prioritization and effective management of goals become an essential element in robotic control. The more aware an agent is of its environment, the more it can effectively manage its goal vector. Two main types of goals motivate robot behavior:

- *Reactive goals* are created in response to changes in the belief system relating to the environment. In essence, such goals are created or destroyed based on environmental changes that take place, and they are the basic mechanisms behind belief, desire, and intention (BDI) principles [43].

- *Proactive goals* are created in relation to future prediction of the environment and the effect of such changes on the agent. For example, an agent sensing a movement towards a state of being off-balance will proactively attempt to correct the situation by planning and applying corrective measures.

As the level of complexity of the agent and the environment increases, the associated interactivity also increases making the accurate perception of the current situation more difficult. Consequently, the creation of reactive or proactive goals in real-time become a significant challenge. Several first and second order logical frameworks have been created to formalize the expression of states in a general sense. In the following sections, we will discuss existing situation and state management frameworks identifying the main components, advantages, and disadvantages of each.

B.2 Situation Calculus

Situation calculus (SC) was introduced by McCarthy and Hayes [86] as a logical formalism for the structured representation of information relating to dynamic environments. Several versions of the situational calculus have been released over the years, but the core principles remain the same. A situation may represent a snapshot of the world at some instant in time [87] or a history of all states and events leading up to an instant in time [111]. Either of the two approaches acknowledges the impossibility of representing a situation in its entirety. Hence, a situation is an approximation that aims to offer valuable information about the state of the world. The following are some of the key features of the situation calculus as described by Reiter [111]:

- SC is a second order logic formal language intended to model the logical structures as well as the changes that take place in dynamic worlds.
- SC currently views a situation as a history of actions leading to a particular situation starting at the initial situation S_0.
- Fluents are used to describe the state of the world in each situation.
- Actions are objects of the domain of discourse that prompt changes to the state of world objects.

B.2.1 SC Semantics

Situation calculus, as defined by Levesque, Pirri and Reiter [81] uses the usual definitions of the standard alphabet of logical symbols, including $\forall$, $\neg$ and $\exists$. The following alphabet is also utilized:

- The predicate symbol $do(action, situation)$ denotes the resulting situation after performing a specific action. The predicate $do(a, s)$ denotes the state produced after performing action a in situation s.
- The predicate symbol $\sqsubset$ $(situation, situation)$ defines the sequencing of situation occurrence. For example, $a \sqsubset b$ means that a occurred before b; i.e. a belongs to the history of b.
- The predicate symbol $Poss(action, situation)$ denotes the executability of an action in a particular situation. $Poss(a, s)$ signifies the possibility of performing action a

in situation s. The statement

$$Poss(step(a), s) \equiv Upright(a, s) \wedge Balanced(a, s)$$

denotes that the ability to step is dependant on being upright and balanced.

In addition to the alphabet mentioned, action functions are also utilized to define actions on objects in a particular situation. Relational and functional fluents are also included and will be explained in the next section.

B.2.2 Fluents

A fluent represents a system entity whose value may be altered over time. Fluents may represent numerical parameters whose value is subject to variation, or a proposition whose truth value might be changed as the system transitions from one situation to another. Fluents are often treated as functions to represent dynamic facts about certain objects within the environment. For example, given the two fluents f and g relating to the environment object x

$$f(x, s) \wedge g(x, s)$$

asserts the status of x in relation to both f and g at situation s.

Fluents are always expressed in relation to a particular situation. Several methods may be used to describe the association between a fluent and a situation, so given the operation op

$$(f\ op\ g)(s) = f(s)\ op\ g(s)$$

In certain instances, the situation specification may be suppressed in fluent expression producing a fluent that could be applied to any relevant situation. For example, given the fluent eq denoting the following equality,

$$\forall x, y, z(eq(x, y) \wedge eq(y, z) \rightarrow eq(x, z)),$$

we dan deduce that the inequelity applies to any implicit situation s, and the given transitive law holds.

Action fluents are utilized to define situation specific relationships among objects. For example, $occupied(x, s)$ signifies that the object x is occupied in situation s. On the other hand, *functional fluents* are used to denote situation specific functions, like $size(x, s)$ or $value(y, s)$ for example.

B.2.3 Effect and Successor Axioms

Given the agent a, the situation s, and the action σ, the fluent $result(a,\sigma,s)$ represents the resultant situation after a executes action σ in s. Actions may have effects on system fluents and effect axioms are used to model such effects. The following example demonstrates the structuring of effect axioms.

Given the two boxes a and b and the item i, we use the following fluent to describe the location of i as well as the status of box b:

Empty(b,s) box b is vacant in situation s

InBox(i,b,s) item i is in box b in situation s.

The following fluent is used to describe the transition of i from one box to another:

Move(i,b) move item i to box b.

The resultant state after applying the $Move(i,b)$ action in situation s is given by

$$Result(Move(i,b),s)$$

and the effect axiom for $Move(i,b)$ is given by

$$\forall s,i,b,InBox(i,b,Result(Move(i,b),s)))$$

As we add more effect axioms for different actions within the environment, we start to realize that effect axioms allows for describing the effects of actions but not for describing the parts of the system that remain unchanged. For example, using the effect axiom listed above, moving i from box A to box B would cause InBox(i,B,Result(Move(i,B),s)) to hold, however, given the existence of another box C, the effect axiom does not convey any information about the value of fluents in regards to C. This inability to handle non-effects is called the *frame problem*.

The most efficient solution to the frame problem revolves around replacing the effect axioms with a single successor state axiom for each fluent. For example, the successor axiom for $InBox(i,b,s)$ is as follows:

$$\forall s,a,i,b(InBox(i,b,Result(a,s)) \leftrightarrow$$
$$((InBox(i,b,s) \wedge \neg\exists a = (Move(i,b_0,s) \wedge b \neq b_0)) \vee (\neg InBox(i,b,s) \wedge \exists a = Move(i,b,s)))$$

The previous axiom signifies that item i would be classified as being in box b in situation $Result(a,s)$ if and only if either of the following takes place:

- Item *i* was already in box *b* in situation *s* and was not moved elsewhere, or
- Item *i* was not in box *b* in situation *s* and was then moved to *b*.

Using successor axioms allows for specifying the values of fluents in relation to all previous actions that might affect them. In addition, it removes the requirement for having to specify the values for all fluents, including those who have not changed. The integrity of the system is increased in that manner as the possibility of omitting the inclusion of one or more axioms does exist.

Once the axiomization of the system is complete, an agent should be able to deduce the resultant state reached after executing a series of actions. Planning is also feasible, as the agent is able to strategize a series of actions that would result in a particular situation. However, there are some limitations to the situation calculus that makes using it in dynamic robotic controller environments difficult. For example, situation calculus, as presented by McCarthy, does not include constructs for specifying concurrent actions or actions of a continuous nature. In addition, actions representing complex behaviors as well as their resultant states would be quite difficult to implement using the given methodologies.

B.3 Fluent Calculus

Fluent calculus was created with the purpose of providing better mechanisms for specifying non-effects of actions as well as the ability to infer these non-effects [129]. In Fluent calculus, situations are considered as representations of states. The inferential frame problem is addressed by utilizing universal state-update axioms that specify how an action modifies a state. This approach is based on the reification of primitive fluents into successor state axioms. A fully mechanical method for deriving state update axioms from an initial arbitrary grouping of effect axioms is presented [130]. Fluent calculus utilizes the key idea of combining multiple effect axioms into a single one. The result would be a more complex axiom that still only specifies effects, yet it would contain sufficient information regarding objects within the system not affected by an action.

Given the fluent $F(\vec{x})$ and a finite set of actions a relevant to $F(\vec{x})$, $\gamma_F^+(\vec{x},a,s)$ would specify all circumstances that would cause $F(\vec{x})$ to become true in s. Similarly, $\gamma_F^-(\vec{x},a,s)$ would describe all circumstances causing $F(\vec{x})$ to become false. A general form successor state axiom is given to describe the dependence of the value of F on its value in the previous situation as well as the effect of the action performed.

The following successor state axiom describes the effect of γ^+ and γ^- on F:

$$F(\vec{x}, Do(a,s)) \equiv \gamma_F^+(\vec{x}, a, s) \vee [F(\vec{x}, s) \wedge \neg\gamma_F^-(\vec{x}, a, s)]$$

Although this approach effectively addresses the representational frame problem, addressing the inferential aspect requires an alternative form of successor state axiom where actions are the main components of the system and not fluents. By reversing the representational model, we may specify the affect of each action on the values of fluents. Let $A(\vec{x})$ represent an action of interest. We may use the formula $\gamma_A^+(\vec{x}, f, s)$ to specify conditions such that f is a positive effect of performing $A(\vec{x})$. Similarly, $\gamma_A^-(\vec{x}, f, s)$ represents conditions such that f is a negative effect of performing $A(\vec{x})$.

The complete account of fluents that hold in situation s is given by

$$holds(f, Do(a(\vec{x}), s)) \equiv \gamma_A^+(\vec{x}, f, s) \vee [holds(f, s) \wedge \neg\gamma_A^-(\vec{x}, f, s)]$$

B.3.1 Mechanical Axiomization

Automating the transition from effect axioms to successor state axioms would be quite beneficial. This would allow for the system logic to deduce the non-effects of actions utilizing known effects. The mechanical axiomization process offered by the fluent calculus operates under the assumption that the effect axioms given constitute a complete set of effects. The presence of indirect effects gives rise to the *Ramification Problem*. Such indirect effects exist through environment state constraints which impose certain circumstances that are not directly specified. Causal propagation [128] approaches are the most general approaches used to address the Ramification Problem [130].

B.4 Probabilistic Situation Calculus

Probabilistic Situation Calculus (PSC) was created to handle dynamic knowledge relating to worlds in which actions have uncertain results [85]. When the outcome of a particular action is not certain, the transition from one state to another goes beyond the capabilities of current representations of the situation calculus, as such representations only deal with discrete state transitions where the effects of actions are known. PSC operates under the assumption that, in a realistic environment, the result of a particular action may only be probabilistically approximated, hence, this framework was created to handle the complexity of such probabilistic distributions. This task is accomplished through the inclusion *probabilistic temporal projection*, which revolves around the prediction of world changes that might take place as a group of actions, or a plan, is put into effect [89].

PSC deviates from traditional situation calculus methods for specifying action preconditions by partitioning them into the following two components:

- Preconditions for inputs, which are analogous to the situation calculus preconditions.
- Probability distribution for the reactions resulting from the processing of an input.

B.4.1 Induction Axioms

Several foundational axioms are used in PSC based on the induction axioms introduced by reiter [112]. The first is an existence axiom which states that the initial situation S_0 is in the path for reaching any other situation. The existence axiom is stated as follows:

$$(\forall \varphi).\varphi(S_0) \wedge (\forall a,s)[\varphi(s) \supset \varphi(do(a,s))] \supset (\forall s)\varphi(s). \tag{B.1}$$

This axiom is comparable to the induction axiom for natural numbers:

$$(\forall \varphi).\varphi(0) \wedge (\forall x)[\varphi(x) \supset \varphi(succ(x))] \supset (\forall x)\varphi(x).$$

The next axiom relates to the reachability of situations from other situations. Since S_0 is the initial state, it does not have any other state in its history, hence

$$(\forall s)\neg s \sqsubset S_0. \tag{B.2}$$

The $\sqsubset$ operator is used for state ordering. The interpretation of $s \sqsubset s'$ is that state s is in the history of state s', which means that state s' could be reached from state s by applying some set of actions.

The name uniqueness axiom is stated as:

$$do(a_1,s_1) = do(a_2,s_2) \supset a_1 = a_2 \wedge s_1 = s_2, \tag{B.3}$$

which means that each action applied in a situation must have a unique name identifier.

PSC also introduces the *legal* predicate which identifies the status of situations after the execution of actions that are not possible in a particular situation. If an illegal action is executed, the holding value of fluents remains the same. The axiom is presented as follows:

$$\neg legal(do(a,s)) \supset (\forall f)holds(f,s) \equiv holds(f,do(a,s)). \tag{B.4}$$

B.4.2 Cumulative Distribution

In order to operate over a domain of uncertain reactions, a cumulative distribution function, denoted as c(i,s), is used, where *i* represents the input in situation *s*. The following cumulative distribution axioms are used:

$$\lim_{x \to +\infty} c(i,s)(x) = 1, \tag{B.5}$$

$$\lim_{x \to -\infty} c(i,s)(x) = 0, \tag{B.6}$$

$$x < y \supset c(i,s)(x) \leq c(i,s)(y), \tag{B.7}$$

$$\lim_{y \to x^{+}} c(i,s)(y) = c(i,s)(x). \tag{B.8}$$

The probability that the fluent f holds after a particular input i is given in situation s is given by the predicate *prob* where

$$prob(f,i,s) = \int_{-\infty}^{+\infty} \varphi(holds(f, do(\langle i,x \rangle, s)))c(i,s)(dx). \tag{B.9}$$

where $\varphi(x) = 1$ if x is *true*, and $\varphi(x) = 0$ if x is $false$.

The PSC also includes support for cases where multiple inputs contribute to a particular reaction. The formulation is based on a *Randomly Reactive Automata*. The reasoning mechanism is extended to allow for multivariate reactions by extending the language of the situation calculus.

B.4.3 Expansion of the Situation Calculus

The frameworks presented in this chapter for the formal description of situations offer methodologies based on first and second order logic. However, in their current state, none of the frameworks presented address the complexities of autonomous agent design, specially in the presence of articulated structure control. The situation calculus operates on a system of deterministic actions and states where the consequences of events and actions are well defined. Fluent calculus, as well as event calculus, follow the same general structure as that of the situation calculus with the addition of constructs for the description of

non-effects. Probabilistic situation calculus introduces the element of system uncertainty yet only in relation to the effects of well defined system inputs.

A formal state determination system for autonomous articulated agents must include the ability to describe the dynamic and uncertain nature of actions, reactions as well as state memberships. In order to expand current methodologies to allow for the comprehensive depiction of agent state, the following elements must be incorporated:

- Determination of the agent's internal state vector described by the articulation status at a given moment in time. The determination system must be designed to handle agent parameters whose values are presented on a continuous scale.
- Determination of the agent's external state vector described by the environment state vector as well as the interactivity vector between the agent and the environment. The determination system must be designed to handle environment parameters whose values are presented on a continuous scale.
- The management of agent goals based on current system status. The agent's goal vector will be constantly changing depending on the current status of the agent as well as the priority values of long-term and short-term goals, so the management system must incorporate support for the dynamic re-prioritization of goals.

Due to the dynamic nature of system interactivity, an agent can be described as being in multiple states at the same moment in time. For example, an agent will never be in an absolute state of being balanced or an absolute state of being off-balance. State membership levels must be incorporated to describe the belongingness of the agent to different simultaneous states.

B.5 Rational Agents

The efficient control of robotic agents requires a level of rational behavior where knowledge of the environment maximizes the agent's chances to achieve successful results. A rational agent would be required to maximize its performance measures by using its past experiences, current information available, as well as current actions available in the current situation. The design of an autonomous robotic agent that relies on rational decision making requires the existence of a framework for the modeling of agent perceptions, as well as short-term and long-term goals.

The Belief-Desire-Intention (BDI) model was developed by Bratman [19] as a framework for practical reasoning. The BDI model has come to be possibly the best known and most studied platform for practical reasoning agents [44]. As shown in Figure B.1, A BDI system consists of the following main units:

- *Beliefs*: The agent belief system revolves around its perception of the environment and itself at a given point in time. Beliefs represent the agent's collective knowledge of the world achieved through sensors, inputs and possibly deductive reasoning. An agent could have different levels of certainty about each of its beliefs. As beliefs possibly represent imperfect information, adherence to belief logics is essential even though the computational representation may not be logical [44].

- *Desires*: Agent goals are represented as desires for achieving certain states or tasks. Desires may be immediate representing short-term goals, or they could represent a structured agenda including long-term goals. Desires may have priority levels attached to them to signify the urgency of the associated goals.

- *Intentions*: Once an agent commits to a particular goal, the goal becomes an intention that the agent is working towards materializing. An intelligent system would weigh the priorities of goals on the agenda then commit to goals that seem appropriate at a given point in time.

- *Actions*: An agent must utilize feasible actions in order to achieve goals it has committed to. The feasibility of actions differs from one situation to the next, so a dynamic feasibility evaluation must be performed to decide on the most appropriate plan of action that would constitute the most efficient method for achieving the goal.

An agent would also be required to handle the causality associated with the execution of actions. In an uncertain environment, the outcome of actions may not be fully determined until an action, or part of it, has taken place. Hence, mechanisms are needed for handling the error associated with the execution of actions. Furthermore, intelligent monitoring is needed in order to compensate for error buildup while the action is being executed by possibly making dynamic changes to the action.

An essential part of effective planning is the ability to re-plan when necessary. Although a particular plan of action might be the most appropriate at a particular point in time, it might not be a time step later. The commitment to a particular action will need to be dynamic within itself. Although a belief might exist that a specific action would

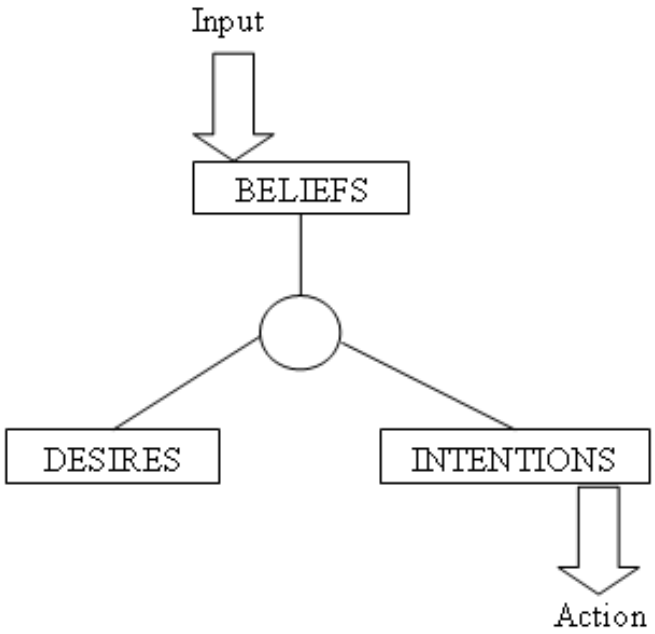


Figure B.1: The belief-desire-intention model.

yield a specific result, the system offers no guarantee of the resultant state of any action. An agent should be able to detect when an intended result seems to be unachievable given the current action plan. Re-planning could then take place taking into consideration knowledge of the previous failure in order to create an alternative course of action.

B.5.1 BDI Relationships

Rao and Georgeff [108] describe the relationship between beliefs, desires and intentions as that of consistency. An agent should have conviction that a goal is achievable in order to classify it as a goal. This property is called *realism* as described by Cohen and Levesque [27]. The world consistency requirement is stated as follows:

$$\forall b \exists g, g \subseteq b$$
$$\forall g \exists i, i \subseteq g$$

This means that given the belief-accessible world set b, a goal-accessible world set g must exist that is a sub-world of b at time t. Similarly, for each goal-accessible world set g, an intention-accessible world set i must exist that is a sub-world of g at time t. The fact that each belief-accessible world must have an associated goal-accessible world does not imply that the reverse must also hold. Belief in the inevitability of a particular fact need not translate to a goal to achieve that fact [108].

Given the desire *d* and potential side-effect *s*, as shown in Figure B.2, an agent believes that by executing any of the two actions *a*1 or *a*2, *d* will be fulfilled with the potential occurrence of *s* as well. Action *a*3 is an alternate course of action causing the non-fulfillment of *d* guaranteeing also that *s* will not occur. The figure shows two of the possible goal-accessible worlds *g*1 and *g*2 associated with the belief-accessible world *b*1. The two intention-accessible worlds *i*1 and *i*2 associated with *g*1 and *g*2 are also shown.

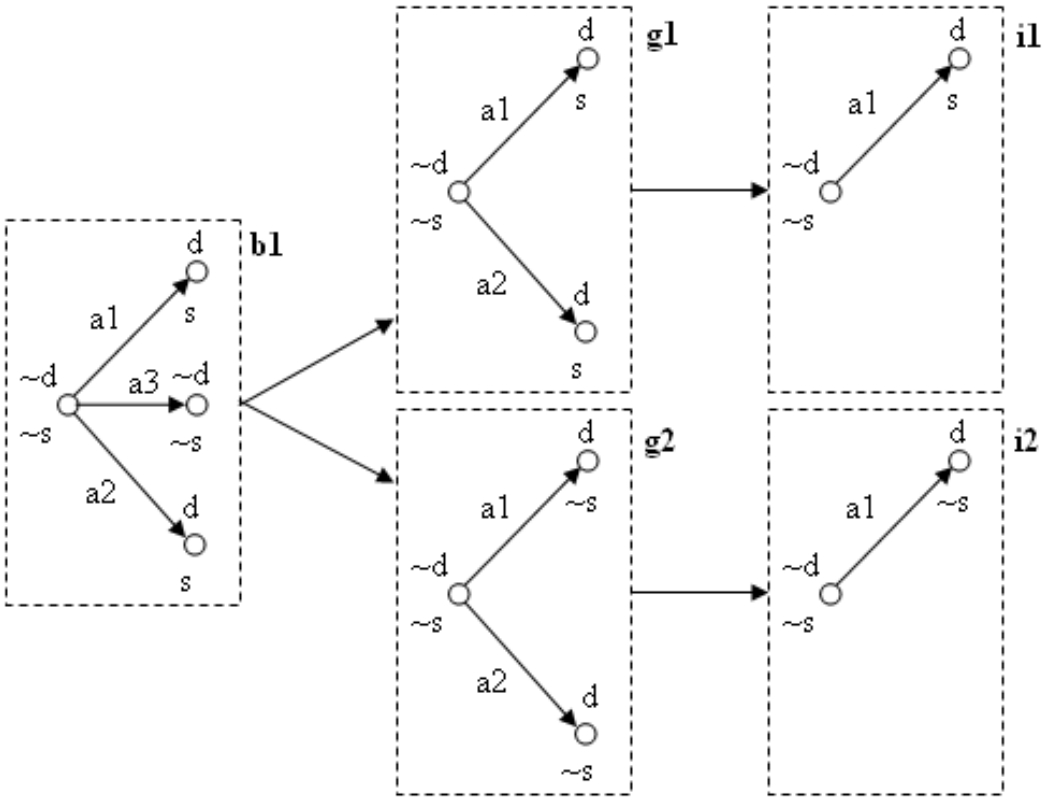


Figure B.2: The relationship between belief, goal, and intention-accessible worlds. The symbols *a*1, *a*2, and *a*3 represent three possible actions. The symbol *d* represents a particular desire, while *s* represents a possible side-effect.

The Computation Tree Logic (CTL), presented by Emerson and Srinivasan [35], is a propositional branching-time logic utilized for program reasoning. Here, we follow the CTL utilization as extended by Rao and Georgeff [108] for the purpose of describing a first-order variant of the logic. The following sections present the constructs defined in [108] for describing the relationships between world entities as well as for system axiomization.

B.5.2 World Semantics

World semantics are presented using the following elements:

- W is a set of worlds.
- E is a set of primitive events that could take place in a particular world.
- T is a set of time points to be related to the existence of worlds on the time line.
- τ defines a binary relation between elements and time points.
- B, G and I represent mapping between the current situation and the agent's set of beliefs, goals and intentions respectively.
- R_t^w is used to denote the set of worlds accessible from w at time t.
- U defines the universe of discourse, and ϕ defines the mapping of entities in U for a given world at a specific point in time.
- Each world w is represented by the tuple $\langle T_w, \tau_w, S_w, F_w \rangle$, where T_w is a set of time points such that $T_w \subseteq T$, and τ_w is a subset of τ restricted to time points in T_w. S_w represents successful occurrence of events between adjacent time points, while F_w represents failure of those events.
- The sub-world w' is defined as a sub-tree of the world w denoted by $w' \sqsubseteq w$.

In addition to the elements listed, the modal operators BEL, $GOAL$ and $INTEND$ are used creating a first-order logic framework for the formal representation of world states. The formulation given is later used to define system axiomization.

B.5.3 Semantics of Events

Events include actions used by an agent to achieve goals. Events transform the agent environment along the time line according to the time point set T. The event e might or might not be successfully executed denoted by $succeeded(e)$ or $failed(e)$ respectively. The predicate $done(e)$ represents an attempt at executing event e. The three predicates are defined as follows:

$$succeeded(e) \text{ iff } \exists t_0, S_w(t_0, t_1) = e$$

$$failed(e) \text{ iff } \exists t_0, F_w(t_0, t_1) = e$$

$$done(e) \equiv succeeded(e) \vee failed(e)$$

B.5.4 BDI Compatibility

Since in order for a goal to exist, it must stem from an existing belief, each identified goal must have an associated belief-accessible world leading to it. This compatibility is formalized using the following axiom:

$$GOAL(\alpha) \supset BEL(\alpha).$$

The same compatibility applies for goals leading to intentions. Hence, each identified intention must stem from a goal-accessible world. The axiom of goal-intention compatibility is given by:

$$INTEND(\alpha) \supset GOAL(\alpha).$$

In order for the entire system to progress forward, intentions must be transformed into actions that may or may not succeed. Such transformation is signified by a commitment on the part of an agent to a particular intention. The axiom of intention to action is given by:

$$INTEND(does(e)) \supset does(e).$$

In addition to the BDI formalization discussed , the framework also offers categorization of agent commitment strategies as it relates to rational agents. Although beyond the scope of this overview, the framework also offers additional axioms, propositions and theorems that assist in describing the inevitability of actions and side effects as well as the success or failure of actions attempted by an agent. For a more detailed discussion of the framework, refer to [108].

B.6 Summary

In this chapter, we discussed several platforms for the description of agent situations as well as the relationships between different agent components. We have also described

the Rao and Georgeff [108] framework for the formal representation of the belief-desire-intention model. The framework offers a significant logical framework for formulating the transition of knowledge about the agent's environment into actions that change the environment.

Although the logical foundations of the situation calculus as well as BDI are quite significant, more work will still have to be done to bridge the gap between theory and applications [109]. The situation calculus does not provide the constructs needed for the dynamic control of articulated structures, while a significant criticism of the BDI model is that it is not appropriate for modeling various types of behaviors. For example, both frameworks do not include constructs for building systems that rely on learning and adaptation in their behavioral patterns. This is a crucial, yet missing, component needed for the creation of autonomous robotic controllers.

BIBLIOGRAPHY

[1] Charles W. Anderson. Strategy Learning with Multilayer Connectionist Representations. *Proceedings of the Fourth International Workshop on Machine Learning* , pages 103-114, 1987.

[2] F. Bacchus. *Representing and Reasoning with Probabilistic Knowledge.* MIT Press, Cambridge, MA , 1990.

[3] F. Bacchus, J. Halpern, and H. Levesque. Reasoning About Noisy Sensors and effectors in the Situation Calculus. *Artiffcial Intelligence* , pages 171-208, 1999.

[4] W. Banzhaf, P. Nordin, R. E. Keller and F. D. Francone. *Genetic Programming: An Introduction: On the Automatic Evolution of Computer Programs and Its Applications.* Morgan Kaufman , 1998.

[5] D. Baraff. Analytical methods for dynamic simulation of nonpenetrating rigid bodies. *Proceedings of SIGGRAPH 89* , 23, 3, 223-232, 1989.

[6] D. Baraff. Curved surfaces and coherence for non-penetrating rigid body simulation. *Proceedings of SIGGRAPH 90* , 24, 4, 19-28, 1990.

[7] D. Baraff. Coping with friction for non-penetrating rigid body simulation. *Proceedings of SIGGRAPH 91* , 25, 4, 31-40, 1991.

[8] D. Baraff. Issues in computing contact forces for non-penetrating rigid bodies. *Algorithmica* , 10, 292-352, 1993.

[9] D. Baraff. Fast contact force computation for nonpenetrating rigid bodies. *Proceedings of SIGGRAPH 94* , 23-34, 1994.

[10] D. Baraff. Interactive simulation of solid rigid bodies. *IEEE Compututer Graphics and Applications* , 15, 3, 63-75, 1995.

[11] D. Baraff. Physically Based Modeling. *ACM SIGGRAPH 2001 Course Notes* , 2001.

[12] John Bares, Martial Hebert, Takeo Kanade, Eric Krotkov, Tom Mitchell and Reid Simmons. Ambler: An Autonomous Rover for Planetary Exploration. *IEEE Computer Society - Computer* , pages 18-26, June 1989.

[13] A. G. Barto, R. S. Sutton and C. W. Anderson. Neurolike Elements that Can Solve Difficult Learning Control Problems. *IEEE Transactions on Systems, Man, and Cybernetics* , 13, pages 835-846, 1983.

[14] Forrest H. Bennett, III and Eleanor G. Rieffel. Design of Decentralized Controllers for Self-Reconfigurable Modular Robots Using Genetic Programming. *The Second NASA/DoD Workshop on Evolvable Hardware (EH'00)* , page 43, July 2000.

[15] Gino Van Den Bergen. *Collision Detection in Interactive 3D Environments*. Morgan Kaufmann Publishers, San Francisco, CA 94111, 2004.

[16] Tobias Blickle and Lothar Thiele. A Mathematical Analysis of Tournament Selection. *Proceedings of the 6th International Conference (ICGA95)* , 1995.

[17] Ulrich Bodenhofer. *Genetic Algorithms: Theory and Applications.* Lecture Notes - Third Edition, 2003.

[18] David M. Bourg. *Physics for Game Developers*. O'Reilly and Associates, Inc.,Sabastopol, CA 95472, 2002.

[19] M. E. Bratman *Intention, Plans and Practical Reason.* Harvard University Press: Cambridge, 1987.

[20] F. Brazier, B. Dunin-Keplicz, J. Treur, R. Verbrugge. Modelling Internal Dynamic Behaviour of BDI Agents. *Proceedings of the Third International Workshop on Formal Models of Agents, MODELAGE'97* , 1997.

[21] C. Breazeal and B. Scassellati Challenges in Building Robots That Imitate People. *Imitation in Animals and Artifacts, MIT Press* , 2001.

[22] R. Brooks. Toward a Practice of Autonomous Systems. *Proceedings of the First European Conference on Artificial Life* , 1992.

[23] G.S. Bullock. Co-evolutionary design: Implications for evolutionary robotics. *Technical Report CSRP384, University of Sussex School of Cognitive and Computing Sciences.* , 1995.

[24] A. Acosta-Calderon and H. Hu. Robotic Societies: Elements of Learning by Imitation. *Proceedings of of the 21st International Conference on Applied Informatics* , pages 315-320, 2003.

[25] Y.U. Cao, A.S. Fukunaga, A.B. Kahng and F. Meng. Cooperative mobile robotics: antecedents and directions. International Conference on Intelligent Robots and Systems , Volume 1, pages 226-234, August 1995.

[26] Michael B. Cline. *Rigid Body Simulation with Contact and Constraints.* Masters Thesis, The University of British Columbia , 2002

[27] P. R. Cohen and H. J. Levesque. Persistence, Intention and Commitment. *Proceedings of the 1986 Workshop on Reasoning About Actions and Plans* , pages 297-340, 1987.

[28] K. Dautenhan and C. L. Nehaniv. *Imitation in Animals and Artifacts* The MIT Press, Cambridge, MA , 2002

[29] Michael R.W. Dawson. From Embodied Cognitive Science To Synthetic Psychology. *First IEEE International Conference on Cognitive Informatics (ICCI'02)* , page 13, August 2002.

[30] Judith Devaney, John Hagedorn, Olivier Nicolas, Gagan Garg, Aurelien Samson and Martial Michel. A Genetic Programming Ecosystem. *15th International Parallel and Distributed Processing Symposium (IPDPS'01)* , page 3013b, April 2001.

[31] G. Dudek, M. Jenkin, E. Milios and W. Wilkes. Experiments in Sensing and Communication for Robot Convoy Navigation. *Proceedings of the International Conference on Intelligent Robots and Systems (IROS95)* , pages 268-273, 2004.

[32] Roman Durikovic and Katsuhiro Numata. Human Hand Model based on Rigid Body Dynamics. *Eighth International Conference on Information Visualisation (IV'04)* , pages 853-857, July 2004.

[33] David H. Eberly. *Game Physics*. Morgan Kaufmann Publishers, San Francisco, CA 94111 , 2004.

[34] A.E. Aiben and J.E. Smith. *Introduction to Evolutionary Computing Theory*. Springer, New York, 10013 , 2003

[35] E. A. Emerson and J. Srinivasan. Branching Time Temporal Logic. *Branching Time and Partial Order in Logics and MOdels for Concurrency* , pages 123-172, 1989.

[36] M. R. Endsley. Theoretical Underpinnings of Situation Awareness: A Critical Review. *Situation Awareness Analysis and Measurement, Lawrence Erlbaum Associates, Publishers, Mahwah, New Jersey* , 2000.

[37] C. Ferreira. Gene Expression Programming: A New Adaptive Algorithm for Solving Problems. *Complex Systems* , pages 87-129, 2001.

[38] C. Ferreira. Gene Expression Programming in Problem Solving. *Proceedings of the Sixth Online World Conference on Soft Computing in Industrial Applications* , September 10-24, 2001.

[39] D. Floreano, S. Nolfi and F. Mondada. Competitive Co-Evolutionary Robotics: From Theory to Practice. *Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior* , 1998.

[40] D. Floreano and J. Urzelai. Evolutionary Robotics: Coping with Environmental Change. *Proceedings of the Genetic and Evolutionary Computation Conference* , 2000.

[41] D. Fogel and Zbigniew Michalwicz. *How to Solve It: Modern Heuristics.* Springer, , 1994.

[42] Alex Freitas. A Genetic Programming Framework for Two Data Mining Tasks: Classification and Generalized Rule Induction. *Proceedings of the Second Annual Conference.* , pages 96-101, 1997.

[43] M. P. Georgeff and A. L. Lansky. Reaktive reasoning and planning. *Proceedings of the Sixth National Conference on Artificial Intelligence* , pages 677-81, 1987.

[44] M. P. Georgeff, B. Pell, M. Pollack, M. Tambe and M. Wooldridge. *The Belief-Desire-Intention Model of Agency.* Springer Publishers, 1998.

[45] Murdoch Gabbay and James Cheney. A Sequent Calculus for Nominal Logic. *19th Annual IEEE Symposium on Logic in Computer Science (LICS'04)* , pages 139-148, July 2004.

[46] P. Godefroid and S. Khurshid. Exploring Very Large State Spaces Using Genetic Algorithms. *8th Conference on Tools and Algorithms for the Construction and Analysis of Systems* , April 2002.

[47] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison Wesley, Reading, MA , 1989.

[48] D. E. Goldberg and K. Deb. *A Comparative Analysis of Selection Schemes Used in Genetic Algorithms.* Morgan Kaufmann, San Mateo , pages 69-93, 1991.

[49] J. Y. Halpern. An Analysis of First-order Logics of Probability.. *Artificial Intelligence* , pages 311-350, 1990.

[50] I. Harvey. Species Adaptation Genetic Algorithms: the Basis for a Continuing Saga. *Proceedings of the Fourth European Conference on Artificial Life* , pages 346-354, 1992.

[51] I. Harvey. *The Artificial Evolution of Adaptive Behaviour.* PhD Thesis, School of Cognitive and Computing Sciences, University of Sussex , 1993.

[52] I. Harvey. Evolutionary Robotics and SAGA: the Case for Hill Crawling and Rournament Selection. *Artificial Life III, Vol. XVI* , pages 299-326, 2004.

[53] I. Harvey, P. Husbands, D. Cliff, A. Thopson and N. Jakobi. Evolutionary Robotics: the Sussex Approach. *Robotics and Autonomous Systems* , 1996.

[54] I. Harvey. Cognition is not Cmputation; Evolution is Not Optimization. *Artificial Neural Networks - ICANN97* , pages 685-690, 1997.

[55] G. Hayes and J. Demiris A Robot Controller Using Learning by Imitation. *Proceedings of the 2nd International Symposium on Intelligent Robotic Systems* , 1994.

[56] F. Herrera and M. Lozano. Two-Loop Real-Coded Genetic Algorithms with Adaptive Control of Mutation Step Sizes. *Technical Report, Dept. of Computer Science and Artificial Intelligence, University of Granada, Spain* , 1997.

[57] F. Herrera and M. Lozano. Tackling real-coded genetic algorithms: Operators and tools for the behavioural analysis. *Artificial Intelligence Review* , 1998.

[58] Haym Hirsh. Genetic Programming. *IEEE Intelligent Systems* , pages 74-84, May 2000.

[59] Robert R. Hoffman and David D. Woods. Toward a Theory of Complex and Cognitive Systems. *IEEE Intelligent Systems* , pages 76-79, January 2005.

[60] John Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, USA, 1975.

[61] Dean Hougen, John Fischer, and Deva Johnam. A neural network pole-balancer that learns and operates on a real robot in real time. *In Proceedings of the MLC-COLT Workshop on Robot Learning* , pages 73-80, 1994

[62] P. Husbands, I. Harvey. Evolution Versus Design: Controlling Autonomous Robots. *Proceedings of the Third Annual Conferences on Articial Intelligence* , 1992.

[63] P. Husbands, I. Harvey and D. Cliff An Evolutionary Approach to Situated AI. *Proceedings of the 9th Bi-annual Conference of the Society for the Study of Artificial Intelligence and the Simulation of Behaviour* , pages 61-70, 1993.

[64] P. Husbands. Evolving Robot Behaviours with Diffusing Gas Networks. *Proceedings of Evorobot* , 1998.

[65] N. Jakobi. Half-baked, Ad-Hoc, and Noisy: Minimal Simulations for Evolutionary Robotics. *Proceedings of the Fourth European Conference on Artificial Life* , 1995.

[66] N. Jakobi, P. Husbands and I. Harvey. Noise and the Reality Gap: The Use of Simulation in Evolutionary Robotics. *Proceedings of the Third European Conference on Artificial Life* , pages 704-720, 1995.

[67] N. Jakobi. *Evolutionary Robotics and the Radical Envelope of Noise Hypothesis.* COGS Reprints, University of Sussex, , 1995.

[68] N. Jakobi and M. Quinn. Some problems and a Few Solutions for Open-ended Evolutionary Robotics. *Proceedings of Evorob98* , 1998.

[69] De Jong, E.D., Dirk Thierens, and Richard A. Watson. Hierarchical Genetic Algorithms. *Proceedings of the 8th International Conference on Parallel Problem Solving from Nature PPSN-04* , pages 232-241, 2004.

[70] John R. Koza. *Genetic Programming.* MIT Press, Cambridge, MA , 1992.

[71] John R. Koza. *Genetic Programming II.* MIT Press, Cambridge, MA , 1994.

[72] John R. Koza. *Genetic Programming III.* Morgan Kauffman, Cambridge, MA , 1999.

[73] John R. Koza and P. Riccardo. A Genetic Programming Tutorial. *Stanford University, Stanford, California* , 2002.

[74] John R. Koza, Martin A. Keane and Matthew J. Streeter. What's AI Done for Me Lately? Genetic Programming's Human-Competitive Results. *IEEE Intelligent Systems* , pages 25-31, May 2003.

[75] Michael D. Kramer and Du Zhang. GAPS: A Genetic Programming System. *The Twenty-Fourth Annual International Computer Software and Applications Conference* , page 614, October 2000.

[76] Robert Laddaga, Mark L. Swinson and Paul Robertson. Seeing Clearly and Moving Forward. *IEEE Intelligent Systems* , pages 46-50, November 2000.

[77] Kristof Van Laerhoven, Kofi A. Aidoo and Steven Lowette. Real-time Analysis of Data from Many Sensors with Neural Networks. *Fifth International Symposium on Wearable Computers (ISWC'01)* , page 115, October 2001.

[78] Eric Lengyl. *Mathematics for 3D Game Programming and Computer Graphics*. Charles River Media, Inc., Hingham, MA 02043, 2002.

[79] Letizia Leonardi, Marco Mamei and Franco Zambonelli. A Physically Grounded Approach to Coordinate Movements in a Team. *22nd International Conference on Distributed Computing Systems Workshops (ICDCSW '02)* , page 373, July 2002.

[80] Henry Leung and Vinay Varadan. System Modeling and Design using Genetic Programming. *First IEEE International Conference on Cognitive Informatics (ICCI'02)* , page 88, August 2002.

[81] H. Levesque, F. Pirri, and R. Reiter. Foundations For the Situation Calculus. *Electronic Transactions on Artificial Intelligence* , pages 159-178, August 1998.

[82] J. Liu and J. Wu. Multi-Agent Robotic Systems. *CRC Press, London* , 2001.

[83] Xin Lu and Shunichiro Oe. Recognizing and Modeling Non-rigid Human Body Actions in Space-time. *Third International Conference on Image and Graphics (ICIG'04)* , pages 501-506, December 2004.

[84] Marco Mamei, Franco Zambonelli and Letizia Leonardi. Co-Fields: A Physically Inspired Approach to Motion Coordination. *IEEE Pervasive Computing* , pages 52-61, April 2004.

[85] P. Mateus, A. Pacheco, J. Pinto, A. Sernadas, and C. Sernadas. Probabilistic Situation Calculus. *Annalysis of Mathematics and AI* , pages 393-431, 2001

[86] John McCarthy and Patrick J. Hayes. *Some Philosophical Problems from the Standpoint of Artificial Intelligence.* Machine Intelligence 4, Edinburgh University Press , pages 463-502, 1969.

[87] John McCarthy and T. Costello. Combining Narratives. *Proceedings of the Sixth International Conference (KR'98)* , pages 48-59, 1998.

[88] John McCarthy. Notes on Self-awareness. *DARPA Workshop on Self-awareness* , April 2004.

[89] D. McDermott. An Aalgorithm for Probabilistic Totally-ordered Temporal Projection *Technical Report YALEU/CSD/RR 941, Yale University* , 1994.

[90] Nik A. Melchior and William D. Smart. Autonomic Systems for Mobile Robots. *International Conference on Autonomic Computing (ICAC'04)* , pages 280-281, May 2004.

[91] A. N. Meltzoff and R. Brooks. *Intention and Intentionality, Chapter Like Me as a Building Block for Understanding Other Minds: Bodily Acts.* Atention, and Intention, The MIT Press, Cambridge, MA , 171-191, 2001.

[92] A. N. Meltzoff. *Elements of a developmental theory of Imitation.* The Imitative Mind: Development, Evolution, and Brain Bases, Cambridge University Press , pages 19-41, 1999.

[93] Neville Melvin, Robert Soricone and James Waslo. On the Automaticity of Genetic Programming. *14th International Conference on Electronics, Communications and Computers* , page 236, February 2004.

[94] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer - Verlag, New York , 1992

[95] Orgazio Miglino, Henrik Lund and Stefano Nolfi. Evolving Mobile Robots in Simulated and Real Envionment. *Massachusetts Institute of Technology Artificial Life 2* , pages 417-434, 1995.

[96] R. Miller and M. Shanahan. The Calculus in Classical Logic - Alternative Axiomatizations. *lectronic Transactions on Artificial Intelligence* , pages 77-105, 1999.

[97] Stefano Nolfi and Dario Floreano. *Evolutionary Robotics - The Biology, Intelligence, and Technology of Self-Organizing Machines.*. The MIT Press, Cambridge, Massachusetts, 2000.

[98] T. J. Norman and D. P. Long. Goal creation in motivated agents. *Intelligent Agents* , Volume 890, pages 277-290, 1995.

[99] Masaki Oshita and Akifumi Makinoushi. A Dynamic Motion Control Technique for Human-like Articulated Figures. *Eurographics 2001* , Volume 20, Issue 3, 2001.

[100] Oztop and M.A. Arbib. Schema Design and Implementation of the Grasp-related Mirror Neuron System. *Biological Cybernetics* , pages 116-140, 2002.

[101] L.E. Parker. The effect of action recognition and robot awareness in cooperative robotic teams. *International Conference on Intelligent Robots and Systems* , pages 212-219, August 1995.

[102] Linda Dailey Paulson. Biomimetic Robots. *IEEE Computer Society - Computer* , pages 48-53, September 2004.

[103] F. Pirri and R. Reiter. Some Contributions to the Metatheory of the Situation Calculus. *Journal of the ACM* , 1999.

F. Pirri and R. Reiter. Some contributions to the metatheory of the situation calculus. , 1999. To appear. http://www.cs.toronto.edu/ cogrobo/. 18 http://citeseer.ist.psu.edu/pirri99some.html

[104] J.B. Pollack, H. Lipson, S. Ficici, P. Funes, G. Hornby and R. Watson. Evolutionary Techniques in Physical Robotics. *Proceedings of the Third International Conference on Evolvable Systems* , pages 175-186, 2000.

[105] D. Pratihar. Evolutionary robotics A Review. *Sadhana Vo. 28, Part 6* , pages 999-1009, December 2003.

[106] C. Ramsey and J. Grefenstette. Case-based Initialization of Genetic Algorithms. *Fifth International Conference on Genetic Algorithms* , pages 84-91, 1993.

[107] A.S. Rao and M.P. Georgeff. A formal Model of Intention. *Pacific Rim International Conference on Artificial Intelligence,* , November 1990.

[108] A.S. Rao and M.P. Georgeff. Modeling Rational Agents Aithin a BDI-architecture. *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning* , pages 473-484, 1991.

[109] A.S. Rao and M.P. Georgeff. BDI Agents from Theory to Practice. *Technical Note 56, AAII* , April 1995.

[110] Rajesh P. Rao, Aaron P. Shon and Andrew N. Meltzoff. *A Bayesian Model of Imitation in Infants and Robots*. Imitation and Social Learning in Robots, Humans, and Animals, Cambridge University Press , 2004.

[111] R. Reiter. The Frame Problem in the Situation Calculus: A simple Solution (sometimes) and a Completeness Result for Goal Regression. *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy, Academic Press* , pages 359-380, 1991.

[112] R. Reiter. Proving Properties of States in the Situation Calculus *Artificial Intelligence* , 64, pages 337-351, 1993.

[113] Paul Robertson and Robert Laddaga. The GRAVA Self-Adaptive Architecture: History; Design; Applications; and Challenges. *24th International Conference on Distributed Computing Systems Workshops - W2: DARES (ICDCSW'04)* , pages 298-303, March 2004.

[114] Justinian P. Rosca. Analysis of complexity drift in genetic programming. *Proceedings of the Second Annual Conference* , pages 286-294, July 1997.

[115] Alfred A. Schmitt. A Dynamical Simulation of Linked Rigid Body Systems using Impulse Technique. *Fakultt fr Informatik, University Karlsruhe* , 2003.

[116] S. Schaal. Is imitation learning the route to humanoid robots? *Trends in Cognitive Science* , pages 233-242, 1999.

[117] S. Schaal, A. Ijspeert and A. Billard. Computational approaches to motor learning by imitation. *Transaction of the Royal Society of London: Series B, Biological Sciences* , pages 537-247, 2003.

[118] S.P. Singh and R.S. Sutton. Reinforcement Learning With Replacing Eligibility Rraces. *Machine Learning* , pages 123-158, 1996.

[119] Martin John Baker. Physics - Jointed structures. *Euclidean Space* , 2006.

[120] Jean Scholtz, Brian Antonishek and Jeff Young. Evaluation of a Human-Robot Interface: Development of a Situational Awareness Methodology. *Proceedings of the 37th Annual Hawaii International Conference on System Sciences* , pages 50130c, January 2004.

[121] Xiong Shengwu, Wang Weiwu and Li Feng . A New Genetic Programming Approach in Symbolic Regression. *15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'03)* , page 161, November 2003.

[122] Schmidl and Victor J. Milenkovic. A Fast Impulsive Contact Suite for Rigid Body Simulation. *IEEE Transactions on Visualization and Computer Graphics* , pages 189-197, April 2004.

[123] Aaron Sloman and Ron Chrisley. Virtual Machines and Consciousness. *Journal of Consciousness Studies* , pages 4-5, October 2004.

[124] Russel Smith. Open Dynamics Engine. *Open Dynamics Engine v0.5 User Guide* , 2004.

[125] Raymond So and Liz Sonenberg. Situation Awareness in Intelligent Agents: Foundations for a Theory of Proactive Agent Behavior. *Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'04)* , pages 86-92, September 2004.

[126] Roy Sterritt and Mike Hinchey. Why Computer-Based Systems Should be Autonomic. *Proceedings of the 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS05)* , 2005.

[127] R.S.Sutton. *Temporal aspects of credit assignment in reinforcement learning.* Doctoral Dissertation, Department of Computer and Information Science, UniversityofMassachusetts, Amherst, MA. , 1984.

[128] Michael Thielscher. Ramification and Causality. *Artificial Intelligence* , 89, pages 317-364, 1997.

[129] Michael Thielscher. Introduction to the Fluent Calculus. *Electronic Transactions on Artificial Intelligence* , pages 179-192, 1998.

[130] Michael Thielscher. From Situation Calculus to Fluent Calculus: State Update Axioms as a Solution to the Inferential Frame Problem. *Artificial Intelligence* , 111, pages 277-299, 1999.

[131] M. Walker and C. H. Messom. A Comparison of Genetic Programming and Genetic Algorithms for Auto-tuning Mobile Robot Motion Control. *The First IEEE International Workshop on Electronic Design, Test and Applications (DELTA '02)* , page 507, January 2002.

[132] Rachel Weinstein, Joseph Teran, and Ron Fedkiw. Dynamic Simulation of Articulated Rigid Bodies with Contact and Collision. *IEEE Transactions on Visualization and Computer Graphics* , 2004.

[133] W. Westenhofer and J. Hahn. Using Kinematic Clones to Control the Dynamic Simulation of Articulated Figures. *Computer Graphics International 1996 (CGI'96)* , page 26, June 1996.

[134] P. H. Winston. *Artificial Intelligence.* Addison-Wesley, Reading, MA , 1993.