

Computer-controlled Human Body Coordination

A THESIS
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
OF THE UNIVERSITY OF STELLENBOSCH
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE



By
Henri Hakl
December, 2003

Supervised by: Dr. L. van Zijl

DECLARATION:

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and has not previously in its entirety or in part been submitted at any university for a degree.

Abstract

A need for intelligent robotic machines is identified. Research and experiments have focussed on stable, or relatively stable, dynamic simulated systems to demonstrate the feasibility of embedding advanced AI into dynamic physical systems. This thesis presents an attempt to scale the techniques to a dynamically highly unstable system — the coordination of movements in a humanoid model. Environmental simulation, articulated systems and artificial intelligence methods are identified as three essential layers for a complete and unified approach to embedding AI into robotic machinery. The history of the physics subsystem for this project is discussed, leading to the adoption of the Open Dynamics Engine as the physics simulator of choice. An approach to articulated systems is presented along with the EBNF of a hierarchical articulated system that was used to describe the model. A revised form of evolution is presented and justified. An AI model that makes use of this new evolutionary paradigm is introduced. A variety of AI variants are defined and simulated. The results of these simulations are presented and analysed. Based on these results recommendations for future work are made.

Opsomming

Die beheer van dinamiese masjiene, soos intelligente robotte, is tans beperk tot fisies stabiele — of relatief stabiele — sisteme. In hierdie tesis word die tegnieke van kunsmatige intelligensie (KI) toegepas op die kontrole en beheer van 'n dinamies hoogs onstabiele sisteem: 'n Humanoïede model. Fisiese simulاسie, geartikuleerde sisteme en kunsmatige intelligensie metodes word geïdentifiseer as drie noodsaaklike vereistes vir 'n volledige en eenvormige benadering tot KI beheer in robotte. Die implementاسie van 'n fisiese simulator word beskryf, en 'n motivering vir die gebruik van die sogenaamde “Open Dynamics Engine” as fisiese simulator word gegee. 'n Benadering tot geartikuleerde sisteme word beskryf, tesame met die EBNF van 'n hierargiese geartikuleerde sisteem wat gebruik is om die model te beskryf. 'n Nuwe interpretاسie vir evolusie word voorgestel, wat die basis vorm van 'n KI model wat in die tesis gebruik word. 'n Verskeidenheid van KI variasies word gedefinieer en gesimuleer, en die resultate word beskryf en ontleed. Voorstelle vir verdere navorsing word gemaak.

Acknowledgements

The financial assistance of the Department of Labour (DoL) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the DoL.

I would like to thank Dr. Lynette van Zijl and the Stellenbosch Writing Laboratory team for their kind assistance, their help was instrumental in ensuring the quality of this thesis. Furthermore, I would like to express my appreciation to my mother for her encouragement and her willingness to help in any way she can.

Contents

Abstract	iii
Opsomming	iv
Acknowledgements	v
1 Introduction	1
2 Prior research	3
2.1 The origin	3
2.2 Biped stalkers	4
2.3 Rise of the machines	5
2.4 This project in relation to prior work	7
3 A foundation for AI robotic dynamics	9
3.1 Notation	10
3.2 Function implications and variants	11
4 Physical simulation	13
4.1 Initial attempts	14
4.2 Adopting the Open Dynamics Engine	15
5 Articulated systems	16
5.1 Additional considerations	17

5.2	An EBNF to describe a body	18
5.3	Joint control in ODE	20
6	AI model	21
6.1	Revised thoughts on evolution	23
6.2	An alternate evolutionary AI model	25
6.3	Implications of the AI model	29
7	Implementation	31
7.1	An implemented foundation	31
7.1.1	Physics and environmental simulation	31
7.1.2	Articulated design	32
7.1.3	AI model parameters	33
7.1.4	A question of fitness	35
7.1.5	Choice of parameters	36
7.2	Neural input and output and AI variation	37
7.2.1	The first dimension	38
7.2.2	The higher dimensions	40
7.2.3	The output	41
7.2.4	AI variations	42
8	Results	44
8.1	Qualitative Sin variant	45
8.1.1	Image sequences	45
8.1.2	Graphs	46
8.2	Quantity Vector variant	49
8.2.1	Image sequences	49
8.2.2	Graphs	50
8.3	Quality Vector variant	52

8.3.1	Image sequences	52
8.3.2	Graphs	53
8.4	Quantity Joint variant	55
8.4.1	Image sequences	55
8.4.2	Graphs	55
8.5	Quality Joint variant	58
8.5.1	Image sequences	58
8.5.2	Graphs	59
8.6	Comparative analysis	61
8.6.1	Characteristic results	61
8.6.2	Comparative performance	62
8.6.3	Reliability of results	64
8.6.4	Closing words and recommendations	64
9	Conclusion	66
	Bibliography	67
A	In-house physics system	73
A.1	Physics	73
A.1.1	Coordinate systems and orientation	73
A.1.2	Inertia	74
A.1.3	3D kinematics and dynamics	76
A.2	Integrators	78
A.2.1	Numeric integrators	78
A.2.2	Physical and numeric instability	81
A.3	Collisions	82
A.3.1	Collision detection	82
A.3.2	Collision response	85

A.4	Final remarks	86
A.4.1	Forces	86
A.4.2	The implementation	86
B	Basic articulated model	87
B.1	A segment and child definition	87
B.2	Explanation	87
B.3	A complete listing of the body definition	90
C	More examples of evolution outside nature	96

Chapter 1

Introduction

The bold promises of artificial intelligence (AI) around fifty years ago have largely not come to fruition. No AI has been crafted that convincingly displays a noteworthy degree of mental prowess that could compare to human skills in analysis, imagination, creativity and insight.

Artificial intelligence has, however, made great strides in certain tasks such as pattern recognition and system control. This progress has allowed AI techniques to be employed in a diverse multitude of applications ranging from flight stability in aircraft to risk assessment of insurance applicants. Though the advent of AI has not heralded a shimmering new era of human history, it has nevertheless become an essential, if mostly transparent, part of life.

The avenues of application for AI have steadily increased with the passage of time. Most recently a surge of effort from academic researchers and a community of enthusiasts has highlighted the field of AI robotics.

AI robotics is concerned with coordinating the motion of a physical system. Potential benefits include the development of intelligent prosthetics in the field of medical technology. Application to graphical systems for cinematography in the entertainment sector. Increased independence and versatility of industrial robotics as well as natural motion and design for androids in the field of emerging robotics. The inclusion of realistic self-controlled actors in industrial and scientific computer simulations.

However, most research activity to date has only demonstrated the feasibility of AI robotics in dynamically stable, or relatively stable, physical systems. *The purpose of this thesis is to present an attempt to embed artificial intelligence coordination in a dynamically highly unstable system. The system that is modelled is a simplified representation of a human being.*

In the next chapter prior and related work is presented. This is followed by a description of a unified approach to AI robotics — broken down into three aspects: Environmental simulation, articulated systems and artificial intelligence modelling. Implementation details and

development are discussed before presenting an analysis of results. Finally recommendations for continued research are made and the thesis is concluded.

Chapter 2

Prior research

2.1 The origin

The notion of artificial life has for a long time lent wings to human-kinds flights of fancy. Religious ideas of creation and myths have filled man's imagination over millennia with golems, automatons, metallic beasts, walking contraptions and life shaped out of clay.

Crude attempts at creating just a small spark of artificial life have existed even before computer simulations in the form of wind-up toys, marionettes, zoetropes and simple mechanical automata. Though devoid of life, these toys were filled with vigor and vitality through the imagination of a playing child or a captivated audience.

Progress and invention have since opened new dimensions to artificial life and nothing more so than the advent of computers. One of the earliest and simplest examples of artificially simulated life on a computer is the *Game of Life* as pioneered by John Conway [16].

Since the *Game of Life* was published by *Scientific American* in 1970 numerous approaches and techniques were applied to simulate and visualize artificial life. Less abstract than the *Game of Life* were Reynold's animations [61] of flocking and herding behaviour. Animal-like locomotion based on physical systems has been demonstrated as early as 1990 by McKenna and Zeltzer [39], as well as Raibert [56] in 1991.

In the years that followed a great number of results were presented that firmly established genetic algorithms as a powerful tool to evolve goal-orientated motion in physical locomotion systems. Examples include the work by Ngo and Marks [54] (1993), Terzopoulos, Tu and Grzescuk [75] (1994), and more recently Paul Urban [78] (2001). Jeffrey Ventrella presents a short, well written history of animated artificial life in chapter 3 of the book *Virtual Worlds: Synthetic Universes, Digital Life, and Complexity* [79].

The evolved creatures of Karl Sims [68, 69] in 1994 are probably the most well known and

popular early results in the field. Inspired by Sims’s virtual creatures a community of enthusiasts [19, 62] and academic researchers have been evolving their own creatures — a rather novel approach was demonstrated by Thomas Ray [57] who applied “*aesthetic, emotional, and emphatic selection*” to Sims’s techniques resulting in aesthetically pleasing, strange and evocative virtual pets.

2.2 Biped stalkers

The work mentioned above has in common that it targets physically stable, or relatively stable, systems. This implies that the physical body that is simulated experiences no, or little, difficulty to maintain an orientation and layout that optimizes the body’s ability to perform actions that optimise a given target function.

The task of successfully controlling bipedal motion has received a considerable amount of attention by researchers. This is in part due to the high degree of difficulty of the problem — as biped walking is dynamically highly unstable — and in part due to the perceived superiority of bipeds over other forms of locomotion — as bipedal movement is extremely adaptive and versatile.

The challenges inherent in biped locomotion are partially due to the complexity of human locomotion: Merely standing straight tasks the human body with a great number of minute, ever-changing adjustments that are necessary to maintain equilibrium. Furthermore the difficulties are aggravated as failure in two-legged walking is usually catastrophic, as no artificial system at the time of writing possesses the coordination to generally recover from falling down.

No accurate models exist to describe human locomotion — though elementary bipedal walking has been successfully modelled analytically by a number of studies: Kajita and Tani [32], for example, modelled leg motion physically based on the principles of a linear inverted pendulum, whereas Pannu *et al* [55] demonstrate how analytical μ -synthesis¹ control can be applied to walking.

A variety of implementations of bipedal robots exist, such as Raibert [56] and the improvements by Boone [11], that give an indication of the possibilities inherent in biped locomotion. Most such attempts expect a hard flat surface and thus fail to adapt to a changing environment including changes in slope and surface friction.

Miller [41, 42], as well as Stitt and Zheng [73], presented implementations that, with some success, cope with changing environmental conditions by adapting system control parameters. Nonetheless no results on bipedal locomotion exist that convincingly survive in unstructured environments and no biped walker has been deployed for practical applications.

¹ μ -Synthesis is a design approach for control applications that incorporates both structured and unstructured uncertainty. It can be applied to control problems that are hard, or impossible, to solve analytically by making use of efficient numerical techniques to solve linear matrix inequalities [12].

2.3 Rise of the machines

One recent and two current projects demonstrate a promising degree of mastery in artificial humanoid motion: *Intelligent Motion Control with an Artificial Cerebellum* by Russell Smith [70], ASIMO, Honda's prototype humanoid [30] and *Active Character Technology* by NaturalMotion [51].

Russell Smith completed his doctoral dissertation, *Intelligent Motion Control with an Artificial Cerebellum*, in 1998. His work focussed on developing the algorithm FOX — a biologically motivated, intelligent controller system that can be used to adaptively and optimally control a variety of systems.

The foundation of FOX is a variant of the Cerebellar Model Articulation Controller (CMAC) neural network, first introduced by Albus [1, 2] in 1975. CMAC was originally devised as a simple biological model of the cortex of the cerebellum. It operates very fast, which has made it a popular choice for a number of applications — Smith utilized this feature of CMAC to create an efficient controller that runs adaptive control in real-time systems.

FOX² implements an eligibility-based reinforcement learning technique. In reinforcement learning³ a learner must make use of delayed rewards and penalties to select sequences of actions that maximise a future reward. Eligibility-based methods maintain an eligibility value for each parameter to indicate their overall impact on the system output and state. Eligibility values in reinforcement learning are used to assign credit for the results of the system's actions and thereby form the basis by which the system is optimised.

Smith demonstrates that FOX is, in fact, a generalization of feedback-error (FBE) control, originally introduced by Kawato [24, 33, 49] in 1987. FBE makes use of neural networks to learn control strategies that maintain a desired system trajectory. According to Smith FOX supersedes FBE in that it covers a larger class of problems, whilst offering less restraints and more flexibility in design and implementation.

In his dissertation Smith describes a number of successful applications of the FOX algorithm, including the implementation of a walking biped. According to Smith the walker was successfully taught to “... *walk in a straight line with a steady gait ... walk in a circle [and] walk up and down a ramp*”.

The Smith walker is endowed with joints that grant it eighteen degrees of freedom — comparable to similar studies on two-legged walking such as ASIMO and Laszlo *et al* citeLaszlo. The walker was initialized with stereotyped periodic walking motions. These motions are not in themselves sufficient to allow the robot to walk — to achieve a stable walking gait the walker had to be taught adaptive control using FOX.

²Not without a sense of humour Russell Smith dubbed his version of CMAC “FOX”, or **F**airly **O**bvious **e**Xtension of CMAC

³For a survey of reinforcement learning refer to [31], for a good introduction refer to [48].

Although Smith's robot is a convincing demonstration of adaptive AI control for walking, he notes that "*it is still unsuited to any practical application*". An alternative, real-world example of a walker in action is Honda's ASIMO.

ASIMO⁴ is a celebrated humanoid robot — a prototype from Honda's technical department. ASIMO is the third generation android (preceded by the prior prototypes P2 and P3) in an ongoing effort by Honda to perfect the science of human-like robotics.

Honda hopes that in the foreseeable future the technology can mature sufficiently to make commercial deployment of robots like ASIMO viable. Though initially ASIMO's offspring will probably be little more than a curiosity in a few households, given time such robotic helpers may become invaluable assistants in human life much like computers.

An elementary AI has been added to ASIMO that enables it to respond to some outside stimuli: Voice recognition allows ASIMO to respond to approximately fifty spoken commands, and since December 2002 ASIMO incorporates the technology to interpret a limited repertoire of human postures and gestures and respond to them.

ASIMO furthermore has limited control of his arms and hands. He is capable of simple manipulations of his arms, which allows him to wave his arms or grasp objects.

The AI elements responsible for ASIMO's reactions to human input are quite simple and serve primarily to lend ASIMO a likeable, human quality. These features make ASIMO an effective spokes figure for Honda's marketing strategy, but offer little substance to AI research at large.

However, the AI that governs ASIMO's mobility has progressed remarkably in the past few years. The control mechanisms can adaptively respond in real-time to changing environmental conditions — the android can successfully change direction and speed while walking, navigate obstacles as well as ascend and descend stairs and slopes in a manner that is closely akin to human motion.

ASIMO's mobility AI still faces several challenges before being able to take a small step that would mark a giant leap for robotkind: A truly robust biped is required to be able to cope with rapidly changing, extreme environmental conditions — and, when failing to maintain footing, should be able to recover by independently standing up. ASIMO's progress along these lines is significant, but still falls far short of the light-footed versatility of human motion.

⁴ASIMO stands for **A**dvanced **S**tep in **I**nnovative **M**obility; though the author was not able to ascertain this, ASIMO may have been named in tribute to one of the spiritual fathers of robotics, Isaac Asimov (1920–1992). Isaac penned down the *Three Laws of Robotics* [5]:

1. "A robot may not injure a human being, or, through inaction, allow a human being to come to harm.
2. A robot must obey the orders given it by human beings except where such orders would conflict with the First Law.
3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Law."

In addition to purely locomotive shortcomings, ideally ASIMO's descendants should be capable of using their bodies with human-like expressiveness: Androids should be able to convey a sense of urgency, leisure, enthusiasm, apathy, joy, dejection, and more in the way they stand, move and place themselves in their environment. A possible approach that in the long run could resolve ASIMO's deficiencies and offer a rich palette of physical expressiveness is presented in the form of Active Character Technology.

NaturalMotion patented Active Character Technology (A.C.T.), which is based on a series of research results into human and animal locomotion by Oxford University [52]. NaturalMotion's algorithms are proprietary and in spite of the author's repeated attempts little information regarding A.C.T. techniques or foundations was made available. Nonetheless it should be safe to assume that the prior work by the A.C.T. development team featured heavily in the creation of Active Character Technology; this work includes articles by Torsten Reil and Colm Massey [58, 59, 60], core and lead technology developers respectively for NaturalMotion.

According to the scant information available Active Character Technology makes use of a model that is brought to life with AI techniques such as "*neural networks, artificial evolution, fuzzy logic, and FSM hybrids*". The original emphasis was placed on creating a successful walker, but since has expanded to include additional behaviors such as "*balancing, jumping, staggering, writhing, [and] active falling*".

In essence NaturalMotion has created an intelligent procedural graphics system (PGS). Procedural graphics systems have existed for quite some time, especially in media intensive sectors such as the movie industry. A PGS is usually implemented as a passive shader that performs some form of animation such as, for example, the fur of the blue hero Sully in Disney and Pixar's classic *Monster's Inc.*

Whereas the fur on Sully's body is animated in a physically accurate manner, the process is nevertheless a passive procedural technique. An intelligent PGS as pioneered by NaturalMotion can interact pro-actively with its environment. The animation is not purely a result of external forces applied by a simulation, but includes responses based on AI computations.

Currently NaturalMotion is applying A.C.T. to applications for the animation industry — *endorphin*, a virtual motion capture studio, is the first of these that is commercially available. With *endorphin* virtual actors and environments can interact in a physically accurate manner, the actors are exposed to AI controlled behaviors, as well as bio-mechanical and environmental simulations.

2.4 This project in relation to prior work

In closing, bipedal locomotion has received extensive research interest and to date has yielded several promising results. Projects such as ASIMO demonstrate not only the feasibility of

two-legged walkers, but also hint at a genuine industry interest in achieving biped motion.

Though such preliminary results are encouraging, they fail in two categories: No universally successful walker has been presented that can navigate unstructured environments — and recover from control errors should they occur — and no project proposes a model that can, potentially, exhibit a human-like versatility of motion.

Interest in bipedal motion is, at least partially, based on man's fascination with himself. Purely navigational problems can be solved efficiently and robustly without requiring the use of two-legged locomotion. In that sense a simple robotic walker does little more than satisfy idle curiosity and an innate need for diversion.

On a more philosophical level, however, research into humanoid robotics is ultimately a desire for companionship, to create a being made in man's image. As such it is only natural to expect such a creation to display at least near human motor skills.

With the exception of NaturalMotion's Active Character Technology the author is not aware of any research that attempts to model AI control on a body that, in terms of mechanical constraints, possesses even remotely human-like characteristics. Furthermore, techniques currently employed in walkers may fail to scale well to the complexity of human bodies, or may impose control constraints that are not justified by the kinetic constraints of a human body.

The purpose of this project is to examine evolved computer control on a model with constraints that closely resemble human parameters.

The upcoming chapter broadly discusses two-legged motion and proposes a unifying foundation that is a prerequisite for any successful implementation of a walker. The elements that constitute the foundation are examined in turn with a description of the author's solution to accommodate these elements in his work.

Chapter 3

A foundation for AI robotic dynamics

Russell Smith [70] accurately observed that the “*field of autonomous robots is very broad, but shallow with few unifying threads*”. Nearly all projects in this field assert their own set of design principles, implementation strategies and learning methods. Smith comments that the techniques currently used “*are drawn from so many areas that it is sometimes difficult to make meaningful comparisons between different systems*”. The result is that a great variety of functional roboters¹ exist, but no single approach has yet yielded a truly satisfying solution.

The principles described in the following foundation for AI roboters are not novel, however they have not — to the author’s knowledge — been explicitly presented and described. This chapter remedies this lack by advancing the notion of a foundation to AI robotic dynamics, followed by chapters that detail the foundation’s elements and offer insights into the author’s implementation of such a foundation.

Every successful computer-controlled roter implements, inadvertently or by design, a foundation consisting of three elements:

- **Physical simulation** provides an environment for the roter within which to function, as well as establishing the physical laws that govern the way the body physically interacts with itself and the environment. Physical simulation is discussed in greater detail in Chapter 4.
- An **articulated system** defines the parameters that determine how the body is designed, as well as how it is controlled and interacts with its environment — articulated systems are reviewed in Chapter 5.

¹For the purposes of this thesis descriptive items such as “robot”, “roter”, “embodied agent” and similar terms refer to the body and physical manifestation described by an articulated system.

- **Artificial intelligence** controls the voluntary behaviour of the body within the system, as well as describing any learning mechanisms by which control efficiency within the system is improved. The AI model used within the project is described in Chapter 6.

These subsystems form both a sufficient as well as a minimal requirement for the successful implementation of a computer-controlled robot. They are a minimal requirement in the sense that an AI-controlled roboter would be incomplete without a physical body and environment within which to (virtually) exist and a mind to control its actions.

To show that they are also a sufficient requirement consider the arguably most complex bio-mechanical system known, a human being: The human body forms an articulated system, the physical simulation is shaped by the universe and the laws that govern it, and the human brain represents the intelligence that directs the body's motions.

It could be argued that the articulated system is merely a subset of the data and tasks present in physical simulation and that it therefore does not merit being explicitly mentioned. However, the task of creating a suitable articulated system is neither trivial nor is it a passive feature of the physics subsystem. Furthermore, making articulated systems an explicit part of the foundation for AI robotics emphasizes the mediating role the articulated system plays between the physical and AI simulations, as it establishes a common ground for AI and physics to act upon.

3.1 Notation

AI robotics can be described as a discrete, time-variant system, where s_t denotes the state of the system at time interval t . The state of the system contains all the changeable parameters of the system, including physics, AI and body variables. A symbolic representation of the foundation can then be expressed as follows:

Given

- s_0 – an initial state
- b – a description of an articulated body
- $\mathbf{A}(x, y)$ – an AI function that maps an input state x and body y to an output state
- $\mathbf{P}(x, y)$ – a physics function that maps an input state x and body y to an output state

then for any time interval t the system of an AI-controlled roboter is described as

$$s_t = \mathbf{P}(\mathbf{A}(s_{t-1}, b), b) .$$

In summary: The task in autonomous robotics is to design an AI function, \mathbf{A} , that optimizes

the behaviour of a body, b , within a world described and controlled by a physics function, \mathbf{P} .

To maximise system performance the AI function often incorporates a learning method, such as back-propagation or evolution. Symbolically this is described with a learning function, $\mathbf{B}(x)$, that maps an input state, x , to an output state. The learning function \mathbf{B} outputs a state that at most differs from the input state in the AI parameters of the input state, in other words, the learning function can only alter AI variables. Thus the AI function described earlier is represented as follows for a system that incorporates a learning mechanism:

$$\mathbf{A}(s_t, b) \rightarrow \mathbf{A}(\mathbf{B}(s_t), b) .$$

For example, in the case of an evolutionary approach to learning the input state to the learning function typically includes the state details of an entire generation of individuals. The learning function uses these to compute the fitness of individual AI parameters and subsequently creates a new generation of AI settings based on the fitness evaluations.

3.2 Function implications and variants

The symbolic representation presented above offers some insight into the nature of the problem of AI robotics; more specifically, it demonstrates that rag-dolls and intelligent agents may be considered subsets of autonomous robots. This suggests that techniques that are successfully employed in those areas could be equally effective for computer-controlled roboters.

Consider the special case where the AI function is in fact an identity function, $\mathbf{A}(x, y) = \mathbf{I}(x) = x$, that is the input state is mapped onto the output state without changes:

$$\begin{aligned} s_t &= \mathbf{P}(\mathbf{A}(s_{t-1}, b), b) \\ &= \mathbf{P}(\mathbf{I}(s_{t-1}), b) \\ &= \mathbf{P}(s_{t-1}, b) . \end{aligned}$$

In this case no voluntary control is exerted on the body and it is reduced to a passive procedural system of applied physics — a rag-doll in the case of a humanoid body. Rag-doll physics are used in the computer gaming and the movie industry to simulate the motion of bodies under different conditions such as falling through a window or down a flight of stairs.

A separate special case applies when the physics function is an identity function, that is $\mathbf{P}(x, y) = \mathbf{I}(x) = x$. In this case

$$\begin{aligned} s_t &= \mathbf{P}(\mathbf{A}(s_{t-1}, b), b) \\ &= \mathbf{I}(\mathbf{A}(s_{t-1}, b)) \\ &= \mathbf{A}(s_{t-1}, b) \end{aligned}$$

thus only the AI function influences the state of the system. The result is an intelligent, embodied agent. Such systems are commonly used in virtual avatars, to convey a more life-like persona, and in certain computer games where entities are moved via means such as waypoints rather than physical simulation — the ghosts in *Pac-man* are a classic, if minimalist, example of such a system.

The elements that form the foundation described in this chapter are discussed at greater length in the following chapters: Chapter 4 describes numerical approaches to physical simulation on computing systems, this description is accompanied by a short history on the development of a physics engine for the project. Chapter 5 presents design considerations and a formal description for simple articulated systems. Finally, Chapter 6 introduces and justifies the AI model used within the project.

Chapter 4

Physical simulation

The simulation of accurate physics enjoys a growing emphasis in modern computer applications. Roaring explosions, falling dice, the descent of snow flakes and the flight of a bullet should not only be functional and attractive to the eye — but they should also behave in a manner consistent with real world experiences.

In AI robotics the simulation of physics is paramount to the creation of successful applications. A given implementation can only achieve its purpose if the physical environment in which it was trained is adequate to describe the physical environment in which it is to be deployed. More precisely, the physics simulation represents the environment and laws which act on an embodied agent and which the agent acts upon — and the simulation determines both the limitation and the potential ability that can be taught to an agent.

Correspondingly a well-evolved field of analytic and numeric techniques to perform physically accurate computations exists. Such techniques cover concepts as diverse as rigid body mechanics and deformation physics to fluid dynamics, capillary flow and realistic fur modelling.

Although these methods encompass and authentically describe most common phenomena, for the purposes of this project it is assumed that constrained rigid body physics offer a sufficiently accurate description of articulated systems to model the motion of a human body. Though this assumption cannot withstand close scrutiny, it adequately satisfies the intentions of this project.

Rigid body physics, as the name suggests, is concerned with the physical characteristics of rigid bodies¹. Methods of computing rigid body dynamics include explicit, impulse-based methods, as presented by Brian Mirtich [46, 47], and implicit methods such as, for example, Stewart and Trinkle [71, 72].

¹By definition, a rigid body is neither compressed nor deformed during any instance. Though this is hardly realistic, it is a simplifying assumption that allows the approximation of a broad range of physical results. For an outline of approaches applicable to rigid body physics refer to the *Online Siggraph '97 Course notes* by Andrew Witkin and David Baraff [81].

Explicit methods reduce the problem of physical simulation to ordinary differential equations which are solved directly using numerical methods such as Euler's integrator², $\mathbf{w}_{new} = \mathbf{w}_{old} + hf(\mathbf{w}_{old})$, or a fourth order Runge-Kutta integrator. However, occasionally a physical problem may be *stiff*, meaning that, to satisfy the differential equation, the equation must be solved in extremely small steps. This renders explicit methods useless for certain problems.

In many cases a stiff differential equation can still be solved explicitly by reformulating the equation to eliminate stiffness — this is, however, not universally possible. Implicit methods are a way to solve such differential equations. An example of an implicit solver is a backwards Euler solver, $\mathbf{w}_{new} = \mathbf{w}_{old} + hf(\mathbf{w}_{new})$ — in essence the system is solved by finding a point \mathbf{w}_{new} such that if that point was regressed in time it would end up at \mathbf{w}_{old} .

Unfortunately \mathbf{w}_{new} cannot be generally solved for, unless the derivative function f happens to be linear. This problem is solved by substituting f with a linear approximation³. Noting that \mathbf{w} is generally a vector, this yields

$$\begin{aligned}\Delta\mathbf{w} &= \left(\frac{1}{h}\mathbf{I} - f'(\mathbf{w}_{old})\right)^{-1} f(\mathbf{w}_{old}) \\ \mathbf{w}_{new} &= \mathbf{w}_{old} + \Delta\mathbf{w}\end{aligned}$$

where \mathbf{I} is the identity matrix and $f'(\mathbf{w}_{old})$ is a matrix.

Thus an implicit solver is required to solve a linear system of equations at every step. Although this appears to be a serious computational requirement, in many cases the matrix is sparse and can be solved in linear time.

Implicit methods lend themselves well to constraint-based approaches to rigid body mechanics, which in turn allow the convenient implementation of limitations, such as joints, in articulated systems. A variety of constraint-based algorithms exist, including penalty methods [50], Lagrangian multipliers [7] and Featherstone's method [20, 21]. Additionally, a hybrid approach that combines explicit and constrained techniques has been forwarded by Mirtich [45].

4.1 Initial attempts

For this project a physics simulation engine was written. The simulator was inspired by Chris Hecker's tutorial [28] on three-dimensional kinematics and dynamics and makes use of explicit computation of dynamics using impulse-based contact resolution as presented by Mirtich in his doctoral dissertation [46]. The reader is referred to Appendix A for an overview of relevant physics equations and algorithms used.

The engine offers an efficient simulation of rigid bodies in a simple environment, emphasising

²For more information on explicit integrators and the dynamics algorithm refer to Appendix A, specifically focusing on section A.2.

³For a full derivation of the implicit Euler solver refer to Baraff's description of implicit techniques in the *Online SIGGRAPH '97 course notes* [81].

speed and stability over accuracy, furthermore the engine was optimised for cuboidal bodies. These choice were made to accommodate the long-term strategy of the project: An AI that learns to control an articulated system with a high degree of freedom requires considerably more simulation time to converge to a solution than an AI that optimises the behaviour of a low-fidelity articulated system.

The physics engine's performance was improved by implementing a variety of integrators, including the Euler integrator, the midpoint method and the fourth order Runge-Kutta integrator. Appendix A.2 discusses the nature of these methods.

All physics engines must internally rely on the efficiency and accuracy of a collision detection system to impose suitable restraints or impulses within the system. For a description of the methods used within the physics engine refer to A.3 in Appendix A.

4.2 Adopting the Open Dynamics Engine

The in-house physics simulator required the ability to compute joint constraints between bodies — this proved to be a formidable problem, as it is difficult to describe axial and positional constraints universally and in a way that can be directly and explicitly solved for. Though not intractable, as illustrated by Mirtich's hybrid approach [45], only scant progress was made in achieving a suitable extension of the existing engine.

During research of related literature and existing implementations the author became aware of an open source physics simulator, *Open Dynamics Engine* (ODE). ODE is an open source project by Russell Smith. In addition to his work in the field of AI robotics [70], his efforts include *Kea*, the core of MathEngine's current product line, as well as *GlowWorm*, *SJL3D* and the unreleased *Lateral Dynamics Engine* — all of which implement some form of physics simulation.

Initial experimentations with ODE revealed that it incorporates a stable, fast engine that supports all the features required for this project. It was subsequently decided to abandon the in-house simulator in favour of the Open Dynamics Engine. This decision was justified by noting that the intentions were to achieve familiarity with implementing rigid body dynamics, rather than the creation of a complete physics simulation package.

The adoption of ODE encouraged the development of, and experimentation with, an articulated system for the project. The design process, considerations and results are detailed in the following chapter.

Chapter 5

Articulated systems

Within AI robotics the purpose of an articulate system is to define a body. This body forms the physical manifestation upon which both passive environmental forces and voluntary AI forces act.

The part of the articulated system that is exposed to the physics simulation need not be the same as the one used for AI manipulations or the part used for rendering purposes. For example, it is possible to use a simple articulated system for physical calculations and AI control, and then use the low-resolution results to render a high-definition body that mimics the behaviour of the simple body.

Simple articulated systems are of little interest academically — indeed in many implementations, including this project, the articulated system is little more than a data structure that houses the description of the body which is used by the physics simulation, as well as the parameters that are used to interface the AI with the body.

Despite this apparent simplicity the creation of an articulated system should not be taken lightly. An articulated system is created with a purpose in mind and consequently the design should reflect this purpose.

For example, the roboter bodies used in most two-legged walker systems are specified in such a manner that they comply with both the demands of bipedal motion and the desire to facilitate the ease with which the AI can learn to control the body: The body is created in a manner that allows bipedal locomotion and simultaneously minimizes the overall degree of freedom (DOF) of the body. Examples include the walkers by Smith [70] and Laszlo *et al* [36], with eighteen and nineteen degrees of freedom respectively, as well as Honda's ASIMO roboter, which is endowed with 26 degrees of freedom¹.

¹A truly minimalist approach was adopted in one of the earlier experiments at NaturalMotion — a successful two-legged walker was implemented sporting as little as six degrees of freedom. The robot was simply modelled after a vastly simplified human body, incorporating only physical features from the hips down [53].

In contrast, the articulated system designed for this project possesses 43 degrees of freedom. The DOF was chosen in a manner to convey a human-like freedom of movement within a body that is build from only eighteen rigid segments, as demonstrated in Figure 1 below.

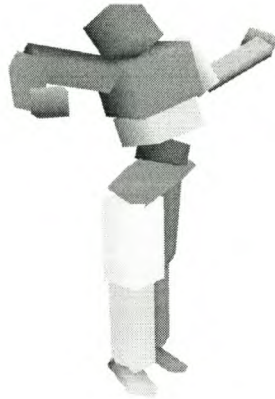


Figure 1: Human-like expressiveness

The benefits of a high degree of freedom to an articulated system are the ability to produce a high degree of varied motion. Physically such systems can perform a much broader range of actions and expressive behaviours than systems with a smaller degree of freedom — an essential requirement for a humanoid robot.

However, a high degree of freedom in an articulated system imposes a great computational burden on an AI that is required to learn to control it. Therefore most research to date has focused on systems with a relatively low degree of freedom, as it makes successful AI control easier and faster to achieve. This project attempts to introduce larger, more human-like articulated systems to AI robotic research.

5.1 Additional considerations

Bare articulated systems may occasionally be augmented to accommodate particular stipulations or features, such as visualization requirements or remote manipulation. To illustrate: One of the greatest challenges in computer graphics is the realistic modelling and animation of human bodies. Nonetheless, the avatars that inhabit a virtual environment may be subject to the visualization requirement that they appear to be human-like.

This implies that subjects should not only appear natural in static scenes but also when in motion. The problem is composed of the complexity of the human body and in the way it interacts with itself, as well as the high sensitivity of human perception to familiar objects such as human figures. Specifically the deformation of human skin, anatomy and clothing under

motion is difficult to reproduce in practise².

Such visual realism need not be considered within the physical simulation or collision detection schemes; it suffices to render the graphics output with the visual cues of flexing muscles and deformed skin. Thus it becomes independent of the physical subsystem and is solely part of the articulated system.

5.2 An EBNF to describe a body

The project makes use of an Extended Backus-Naur Form (EBNF) description in conjunction with a scanner and parser to load an articulated model at run time. The EBNF is presented below and the articulated model description is listed in Appendix B.

The EBNF used in this project can describe articulated systems by specifying body segments, dimensions, positions and connections to other segments. This is sufficient to describe arbitrarily complex systems, provided that segments do not require individual initial orientations and that each segment in that system is joined to at least one other segment. In other words: A human with a chain around her neck represents two separate articulated systems.

The EBNF first declares a base segment number; functions that perform computations on the articulated system will start at the base segment and then follow the logical connections as established during parsing of the body description. Following the base segment are one or more segments.

Each segment possesses:

- an identifier,
- a type definition (limited to rectangular prisms in the project),
- a position, and
- possibly one or more children.

Each child possesses:

- an identifier (that determines which segment the joint connects to),
- a joint type (limited to ball-and-socket joints and hinge joints in the project),
- a position at which the joint acts,
- a principle local axis along which the joint acts, and

²For results in musculature modelling and visualization, refer to Turner [77], Shen [67], Zuo *et al* [82] and Scheepers *et al* [65].

- joint constraints that describe the limitations of motions at that joint.

For more details and a complete listing of the body description used in the project refer to Appendix B.

A basic EBNF for articulated systems

```

system      = initial body
initial     = kwBaseS INT
body        = segment {segment}*

segment     = kwSegm id type dim pos [children]
id          = kwSgid INT
type        = kwStyp INT
dim         = kwSdim VECTOR
pos         = kwSpos VECTOR

children    = kwSchl child {child}*
child       = kwChld cid jtype jpos jaxis jcon
cid         = kwChid INT
jtype       = kwJtyp INT
jpos        = kwJpos VECTOR
jaxis       = kwJaxs VECTOR [“;” VECTOR]
jcon        = kwJcon INT “,” INT [“,” INT “,” INT]

kwBaseS    = “basesegment”
kwSegm     = “segment”
kwSgid     = “segmentID”
kwStyp     = “segmentType”
kwSdim     = “segmentDimension”
kwSpos     = “segmentPosition”
kwSchl     = “segmentChildren”
kwChld     = “child”
kwChid     = “childID”
kwJtyp     = “jointType”
kwJpos     = “jointPosition”
kwJaxs     = “jointAxis”
kwJcon     = “jointConstraint”

VECTOR     = REAL “,” REAL “,” REAL
REAL       = [ “+” | “-” ] VALUE “.” VALUE
INT        = [ “+” | “-” ] VALUE

VALUE      = DIGIT {DIGIT}*
DIGIT      = “0” . . “9”

```

5.3 Joint control in ODE

Each joint, both ball-and-socket as well as hinge joints, within the project is accompanied by an angular motor joint. An angular motor joint allows the control of relative angular velocities of two bodies. ODE enables the user to specify high and low stops along each joint axis of the angular motor beyond which motion is completely inhibited, furthermore ODE allows the user to specify a desired angular velocity around each joint axis and a maximum force with which the simulator will try to achieve that velocity.

The use of angular motors allows a convenient interface of AI control to the articulated system — AI results are interpreted as motor velocities along the axes of angular motors. There are several benefits to such an approach to applying AI behaviour to an articulated system:

- Varying power at different joints can be abstracted by specifying a maximal torque for each joint. The AI need then only be concerned with issuing desired angular velocity orders and the physics simulation will attempt to implement these orders as best as possible given the peak force that it may apply.
- Angular motor parameters act not as instantaneous but continuous forces, thus AI commands will remain in effect until they are changed by the AI. This implies that it is possible to separate AI computations from physics simulations, in other words, it is possible to perform several iterations of physical simulation before new AI behaviour is calculated. This may be desirable in circumstances where changes in AI behaviour do not occur frequently but physical simulation must progress in small increments to maintain dynamic stability.

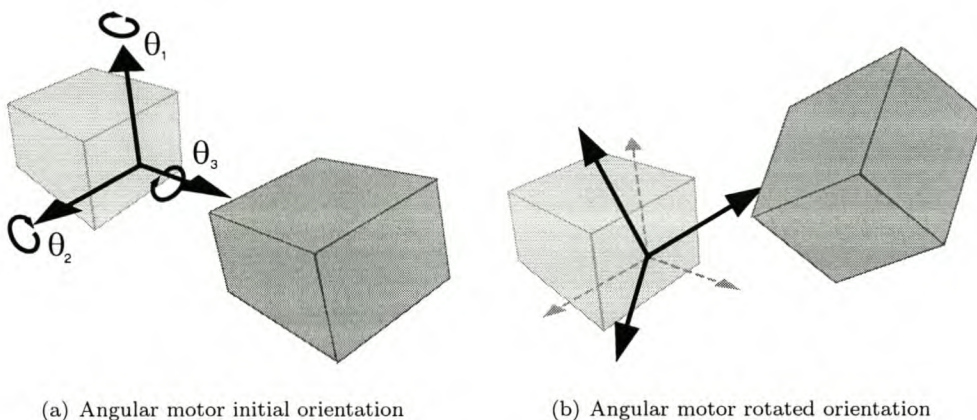


Figure 2: An angular motor with axial rotations θ_1 , θ_2 and θ_3

The creation of an articulated system for the project gave rise to an AI to control it. The following chapter introduces a revised notion of evolution which forms the basis of the artificial intelligence model within the project. This AI model is formally described and its implications are discussed.

Chapter 6

AI model

The field of artificial intelligence has brought forth an astounding variety of approaches to solve a wide range of problems. Problems as structured as chess, as critical as computer-aided flight stability and as subtle as human dialogue have been addressed with varying degrees of success.

Nonetheless the richest fruits of AI research have not been solutions to difficult problems, but the myriad algorithms, techniques and methods that have been created in an attempt to lend to machines some measure of human cognition. These techniques include algorithms to perform concept learning, decision tree learning, Bayesian learning, artificial neural networks, evolutionary programming, analytical learning and reinforcement learning¹.

The first task in designing an AI for a given problem is to determine which of these AI techniques are suitable to solve the problem. In the case of AI robotics this yields a list of algorithms that have been shown to be successful in the field of control problems.

Classically AI control problems are approached with a functionality paradigm: Problems are divided into functional segments such as sensory analysis, planning and execution [13]. However, it has been argued [8] that classical artificial intelligence is too narrow to produce robust control structures for real world environments — instead it has been proposed to approach control problems with a modularity paradigm.

A modular approach to control implements a host of smaller modules, each of which performs a simple, task-orientated function. The behavior of the system is not explicitly specified but emerges from the interaction of the collective modules. Such a modular approach could be likened to the functioning of a colony of bees — individual members only perform simple tasks and are expendable, however these individuals form a collective that performs tasks and functions that supersede the tasks and functions of the individual. Examples of such modular control solutions include work by Brooks [13, 14] and Beer [8, 9].

¹For a comprehensive overview of AI techniques refer to, for example, *Machine Learning* by Tom M. Mitchell [48].

Irrespective of the design paradigm, artificial neural networks (ANN), genetic algorithms (GA) and reinforcement learning techniques are popular in control implementations. Specifically feed-forward neural networks have received considerable academic attention [43, 80]: The network forms an adaptable component that is used as the basis for a learning controller and the network weights are optimised using techniques such as back-propagation, genetic algorithms or stochastic search algorithms.

Genetic algorithms, often in conjunction with artificial neural networks [3, 40], are an efficient tool to perform unsupervised learning. Reinforcement learning techniques allow control problems to be solved using only scalar reward-penalty feedback to the system, such as, for example, the controller implemented by Russell Smith [70].

It was decided that the AI model of this project would make use of artificial neural networks that would be optimized with a genetic algorithm. ANNs were chosen on the grounds that they offered the ability to

- easily adapt to various input encodings,
- accept input data that is only approximately accurate,
- output probabilistic results, and
- evaluate past data by using a recurrent architecture.

Genetic algorithms, on the other hand, have proven to be a reliable means of optimizing learning parameters for ANNs [3, 40] — furthermore the choice of GAs is motivated with strong arguments:

- Evolution is a successful and highly robust method for adaptation in biological systems.
- Genetic algorithms can successfully search through solution spaces that contain complex interacting parts which are difficult to model using analytic or numeric techniques.
- Genetic algorithms can easily be implemented to take advantage of parallel computing.

Reinforcement learning techniques, though powerful learning mechanisms, tend to converge to solutions very slowly [48, 70] — especially in large and complex systems. Therefore it was decided to adopt the use of genetic algorithms, as opposed to reinforcement learning, as an aid in optimizing the artificial neural networks that would be used.

The following sections first present a modified model of evolution that will form the basis of the AI model that will be described afterwards. This is followed by a description of the neural input and output models that are used in this project and concludes with a section on different AI variations that are used in the implementation.

6.1 Revised thoughts on evolution

Even though man has made use of genetic and evolutionary theory in the form of selective breeding for millennia, it was not until *On the Origins of Species* by Charles Darwin [17] in 1859, and the contributions of Alfred Russel Wallace, that the concept of evolution was introduced to the world². The theory of evolution postulates that genetic variations in individuals of a species enable some of its members to become more successful within their environment than others, these individuals are more likely to pass on their genes to the next generation thereby ensuring that the qualities that make them successful will gradually become intrinsic to the species as a whole.

Lamarck³ was one of the first scientists to publish thoughts on evolutionary change in nature. In computer science, specifically in genetic algorithms, Lamarckian evolution refers to the idea that knowledge acquired by specimens in a generation can be passed on to subsequent generations. This is an attractive proposition, as it presumably allows more efficient evolutionary progress than is perpetuated through a simple generate-and-test approach.

Unfortunately, for the progress of evolution, current understanding of evolution states that the genetic framework of an individual is not altered through experiences in its lifetime. However, various mechanisms have been suggested that do allow individual learning to influence future generations, such as the Baldwin effect [6].

Baldwin observed that in a changing environment evolutionary pressure favors individuals capable of learning to adapt to the changes. These individuals in turn need to rely less on their genetic encoding, thus enabling them to support a more diverse gene pool. This facilitates more rapid evolutionary adaptation, thus the individual's experience can indirectly influence the rate of evolutionary change.

In machine learning practise the Baldwin effect has seen some exposure and experimental evidence shows that convergence of genetic algorithms can indeed be accelerated by making use of this mechanism [22, 29]. The Baldwin effect can be implemented by allowing certain genes to be changeable over the course of an individuals lifetime — in effect allowing it to perform its own localised optimization.

In contrast to the Baldwin effect, Mayley introduced the Hiding effect [38] which demonstrates that learning can also hinder evolutionary progress. The Hiding effect manifests itself when learning obscures genetic differences; this can occur, for example, when different genotypes

²Only recently have the contributions of Alfred Russel Wallace to the theory of evolution been given due credit — Darwinian evolution is now also commonly referred to as the Darwin/Wallace theory of evolution.

³Jean Baptiste Pierre Antoine de Monet, Chevalier de Lamarck (1744–1829), though discredited at his time, was acknowledged by Charles Darwin and other early evolutionists as a great zoologist and a forerunner of evolution. His description of evolution, most prominent in *Philosophie zoologique*, is surprisingly similar to Darwin's and Wallace's findings — and in some cases more complete. Unfairly Lamarckian evolution, in biology, refers in rather derogatory terms to the theory that acquired traits may be inherited; this does not reflect Lamarck's ideas, who instead suggested that in a changing environment the needs, and thus behavior, of an organism change and that consequently certain structures and organs would see greater use or disuse. Over generations these structures would correspondingly grow or shrink.

map onto similar phenotypes due to learning, resulting in equivalent fitness scores. This makes it difficult to distinguish between more and less successful genotypes and as a consequence evolutionary progress is slowed.

McLean [40] demonstrates that global search methods, such as genetic algorithms based on Darwinian evolution, offer effective strategies for highly dynamic environments, whereas genetic algorithms emphasising local learning (Lamarckian evolution) perform better in stable environments. Nonetheless, the results by McLean [40], the work by Belew, McInerney and Schraudolph [10], as well as the papers by Sasaki and Tokoro [63, 64], conclude that a hybrid approach combining both global and local search methods offers the best overall performance. In fact, it has been demonstrated by Littman [37], as well as Todd and Miller [76], that a combination of genetic and learning methods can solve problems that cannot be solved optimally using either genetic or learning algorithms exclusively.

The evolutionary principles discussed to this point have only considered evolution in a biological context and how it applies to computer science. The remainder of this section attempts to expand the notion of evolution to include all systems that adapt or change over time.

Evolution is the process of change within certain systems. Tentatively we define evolutionary systems to include all systems in which changes in time yield results that cannot be predicted deterministically. This includes evolution, as observed in nature, as well as in other systems such as languages, cultures, music, industries and sport — even highly abstract concepts, such as freedom, continuously evolve and change over time. There is no requirement that evolutionary progress be propagated through a genetic framework, nor for evolution to be guided with purpose and intent.

Evolutionary changes may be of the form of adaptations, as observed in nature, that improve the odds that members of a given species may succeed. Some changes are forced upon a system from outside forces, such as selective breeding of domesticated animals (domesticated animals are diminished in size compared to their original, wild counterparts). However, changes need not necessarily have a purpose beyond change itself, indeed in some cases variation may be caused by as little as a whim or a haphazard mutation.

For example, in the case of languages, some changes reflect geo-political conditions when a language adopts terms of a different, pre-dominant language, whereas other changes reflect technological invention and social development. Still, some changes are hard to account for by means other than a lingual form of Brownian motion: Thirty years ago teenagers may have exclaimed delight with terms such as *marvy* or *far out*, today the offspring of those teenagers are more likely to use terms such as *wicked*, *phat* and *fly*⁴.

Another example⁵: The entertainment industry thrives on a form of evolution that emphasizes imitation. A given song, movie or computer game might be particularly successful, due to

⁴On a lighter note, some changes are more subtle — “cool” is not cool anymore, but “kewl” is still cool.

⁵Additional examples of evolutionary systems outside of nature are presented in Appendix C.

an original concept or a breakthrough in technology — subsequently a great variety of similar songs, movies or games are released that attempt to capitalize on the success of the former. Such exploitation may be short-lived or may, eventually, form the basis of a new genre.

In conclusion, we assert that evolution is not a concept intrinsically linked with the evolutionary theories for nature introduced by Darwin and Wallace. Instead evolution is a far more universal process that may be influenced by a host of direct, as well as subtle, forces.

The consequence for evolutionary programming techniques used in computer science is that genetic algorithms need not necessarily conform to a biological context of evolution. An AI model based on alternate evolutionary principles is presented next.

6.2 An alternate evolutionary AI model

This section describes the AI model used within the project. The basis of the AI is a simplified description of evolutionary processes in society:

A society consists of a set of communities, referred to as families within the project. Each family consists of a set of individuals. Each family possesses its own set of ideas, values and opinions and these values are represented by the dominant member of that family, the head of the family. The remaining family members possess values and opinions similar — though not identical — to the head of the family.

The process of evolution over generations within a family functions as follows: Family members try new variations of values and opinions and the most successful of these variations are mimicked by the remainder of the family. After a few iterations the most successful variation is determined and that family member becomes the head of the family. Each family is rated by the success of its family head.

A new generation of families is generated from the old set of families. Each new family is created by the exchange of ideas, values and opinions between two existing families using the cross-over operator of classic genetic algorithms. The values and opinions of more successful families are more likely to be chosen for subsequent generations than the ideas of less successful families.

To put the above analogy into a symbolic context we first define a notational convention:

$$Z_{i,j,k}^{\alpha,\beta}$$

refers to a variable Z at generation α , iteration β , family i , family member j and opinion k , and to easily distinguish variables associated with a family representative we define

$$\hat{Z}$$

which refers to a variable Z that belongs to a family head. Note that $Z \equiv \hat{Z}$, in other words the emphasis of a variable only serves to make the underlying equations more readable.

Then define

- b – a description of an articulated body, as presented in Appendix B
- $\mathbf{A}(x, y)$ – an AI function (a neural network) that maps an input state x and body y to an output state
- $\mathbf{P}(x, y)$ – a physics function (ODE) that maps an input state x and body y to an output state
- $\mathbf{C}(x)$ – a cross-over function; maps a set of input elements onto a single output element
- $\mathbf{rnd}(x, y)$ – a function that returns a random, normally distributed value with mean x and deviation y
- ω – the number of generations to simulate
- α – the generation index, $\alpha \in \{1, 2, 3, \dots, \omega\}$
- η – the number of iterations
- β – the iteration index, $\beta \in \{1, 2, 3, \dots, \eta\}$
- δ – the number of families
- i – the family index, $i \in \{1, 2, 3, \dots, \delta\}$
- γ – the number of family members
- j – the family member index, $j \in \{1, 2, 3, \dots, \gamma\}$
- ϵ – the number of opinions — each opinion is, in fact, a network weight that is used by the neural network $\mathbf{A}(x, y)$
- k – the opinion index, $k \in \{1, 2, 3, \dots, \epsilon\}$
- λ – the imitation rate, $\lambda \in [0, 1]$.

We define the starting conditions of the AI model with an initial set of family heads

$$\hat{F}^{0,\eta} = \{\hat{H}_1^{0,\eta}, \hat{H}_2^{0,\eta}, \dots, \hat{H}_\delta^{0,\eta}\}.$$

Note the use of η as iteration index; this ensures a consistent notation for the creation of subsequent generations (see Equation 1). Each initial family head possesses a set of initial opinions

$$\hat{H}_i^{0,\eta} = \{\hat{\sigma}_{i,1}^{0,\eta}, \hat{\sigma}_{i,2}^{0,\eta}, \dots, \hat{\sigma}_{i,\epsilon}^{0,\eta}\} \text{ with } i \in \{1, 2, 3, \dots, \delta\}$$

and each initial opinion is a random value

$$\hat{\sigma}_{i,k}^{0,\eta} = \mathbf{rnd}(0, 1) \text{ with } i \in \{1, 2, 3, \dots, \delta\}, k \in \{1, 2, 3, \dots, \epsilon\}.$$

Given such conditions we can generally define the initial set of family heads for generation α as follows:

$$\hat{F}^{\alpha,0} = \{\hat{H}_1^{\alpha,0}, \hat{H}_2^{\alpha,0}, \dots, \hat{H}_\delta^{\alpha,0}\} \quad \text{with } \alpha \in \{1, 2, 3, \dots, \omega\} \quad (1)$$

where each family head is determined by the cross-over function, $\mathbf{C}(x)$

$$\hat{H}_i^{\alpha,0} = \mathbf{C}(\hat{F}^{\alpha-1,\eta}) \quad \text{with } \alpha \in \{1, 2, 3, \dots, \omega\}, i \in \{1, 2, 3, \dots, \delta\}$$

such that the cross-over function takes a set of family representatives as input and produces a new family head by randomly selecting two family heads (the random selection is not uniform but gives preference to more successful representatives), randomly exchanging opinions between them, and finally outputting one of the two modified family heads as result.

A society, G , is a set of δ families. We determine the initial society for generation α as follows:

$$G^{\alpha,0} = \{F_1^{\alpha,0}, F_2^{\alpha,0}, \dots, F_\delta^{\alpha,0}\} \quad \text{with } \alpha \in \{1, 2, 3, \dots, \omega\}$$

where each family consists of a family head and $\gamma - 1$ additional family members:

$$F_i^{\alpha,0} = \{\hat{H}_i^{\alpha,0}, H_{i,1}^{\alpha,0}, H_{i,2}^{\alpha,0}, \dots, H_{i,\gamma-1}^{\alpha,0}\} \quad \text{with } \alpha \in \{1, 2, 3, \dots, \omega\}, i \in \{1, 2, 3, \dots, \delta\}$$

each non-head family member possesses a set of ϵ opinions

$$H_{i,j}^{\alpha,0} = \{\sigma_{i,j,1}^{\alpha,0}, \sigma_{i,j,2}^{\alpha,0}, \dots, \sigma_{i,j,\epsilon}^{\alpha,0}\} \quad \text{with}$$

$$\alpha \in \{1, 2, 3, \dots, \omega\}, i \in \{1, 2, 3, \dots, \delta\}, j \in \{1, 2, 3, \dots, \gamma - 1\}$$

such that each opinion is a deviation (mutation) of the family head's opinions:

$$\sigma_{i,j,k}^{\alpha,0} = \hat{\sigma}_{i,k}^{\alpha,0} + \mathbf{rnd}(0, 0.1) \quad \text{with}$$

$$\alpha \in \{1, 2, 3, \dots, \omega\}, i \in \{1, 2, 3, \dots, \delta\}, j \in \{1, 2, 3, \dots, \gamma - 1\}, k \in \{1, 2, 3, \dots, \epsilon\}.$$

We define the success rating of a given family member with a fitness function. The fitness function performs an AI and physics simulation and evaluates the resultant state:

$$f(H_{i,j}^{\alpha,\beta}) \rightarrow f(\mathbf{P}(\mathbf{A}(H_{i,j}^{\alpha,\beta}, b), b)).$$

The fitness function can be used to determine the overall fitness of a family:

$$g(F_i^{\alpha,\beta}) = \max\{f(\hat{H}_i^{\alpha,\beta}), \max_{j=1}^{\gamma} f(H_{i,j}^{\alpha,\beta})\}.$$

Each society progresses through a series of iterations before a new generation is created. The iterative process is described with the following equations: For each iteration in a generation there exists a set of family heads

$$\hat{F}^{\alpha,\beta} = \{\hat{H}_1^{\alpha,\beta}, \hat{H}_2^{\alpha,\beta}, \dots, \hat{H}_\delta^{\alpha,\beta}\} \quad \text{with } \alpha \in \{1, 2, 3, \dots, \omega\}, \beta \in \{1, 2, 3, \dots, \eta\}$$

to assist in computing these family heads we introduce temporary variables

$$\hat{j}_i^{\alpha,\beta} = \left(\exists j \mid f(H_{i,j}^{\alpha,\beta-1}) = g(F_i^{\alpha,\beta-1}) \right) \quad \text{with}$$

$$\alpha \in \{1, 2, 3, \dots, \omega\}, \beta \in \{1, 2, 3, \dots, \eta\}, i \in \{1, 2, 3, \dots, \delta\}$$

and use these to compute the family heads of the iterations in a generation

$$\hat{H}_i^{\alpha,\beta} = H_{i,\hat{j}_i^{\alpha,\beta}}^{\alpha,\beta-1} \quad \text{with}$$

$$\alpha \in \{1, 2, 3, \dots, \omega\}, \beta \in \{1, 2, 3, \dots, \eta\}, i \in \{1, 2, 3, \dots, \delta\}$$

furthermore we explicitly assign the family heads' opinions

$$\hat{\sigma}_{i,k}^{\alpha,\beta} = \sigma_{i,\hat{j}_i^{\alpha,\beta},k}^{\alpha,\beta-1} \quad \text{with}$$

$$\alpha \in \{1, 2, 3, \dots, \omega\}, \beta \in \{1, 2, 3, \dots, \eta\}, i \in \{1, 2, 3, \dots, \delta\}, k \in \{1, 2, 3, \dots, \epsilon\}$$

and reinsert the old family head into the list of non-head family members to ensure that the genetic material is not lost

$$H_{i,\hat{j}_i^{\alpha,\beta}}^{\alpha,\beta-1} = \hat{H}_i^{\alpha,\beta-1} \quad \text{with}$$

$$\alpha \in \{1, 2, 3, \dots, \omega\}, \beta \in \{1, 2, 3, \dots, \eta\}, i \in \{1, 2, 3, \dots, \delta\}.$$

Note that not every $\hat{j}_i^{\alpha,\beta}$ exists, as in cases where the former family head is still the most successful member of a family, in such cases the index simply falls away; the equations still hold and describe the model correctly. For example, due to notational convention

$$\hat{j}_i^{\alpha,\beta-1} = \emptyset \quad \rightarrow \quad H_{i,\hat{j}_i^{\alpha,\beta}}^{\alpha,\beta-1} \equiv H_i^{\alpha,\beta-1} \equiv \hat{H}_i^{\alpha,\beta-1}.$$

To compute the sets of family heads for each iteration we make use of the following descriptions for societies, families, family members and opinions: Each iteration produces a society consisting of δ families

$$G^{\alpha,\beta} = \left\{ F_1^{\alpha,\beta}, F_2^{\alpha,\beta}, \dots, F_\delta^{\alpha,\beta} \right\} \quad \text{with } \alpha \in \{1, 2, 3, \dots, \omega\}, \beta \in \{1, 2, 3, \dots, \eta - 1\}$$

each of these families consists of a family head and $\gamma - 1$ additional family members

$$F_i^{\alpha,\beta} = \left\{ \hat{H}_i^{\alpha,\beta}, H_{i,1}^{\alpha,\beta}, H_{i,2}^{\alpha,\beta}, \dots, H_{i,\gamma-1}^{\alpha,\beta} \right\} \quad \text{with}$$

$$\alpha \in \{1, 2, 3, \dots, \omega\}, \beta \in \{1, 2, 3, \dots, \eta - 1\}, i \in \{1, 2, 3, \dots, \delta\}$$

and each (non-head) family member possesses a set of ϵ opinions

$$H_{i,j}^{\alpha,\beta} = \left\{ \sigma_{i,j,1}^{\alpha,\beta}, \sigma_{i,j,2}^{\alpha,\beta}, \dots, \sigma_{i,j,\epsilon}^{\alpha,\beta} \right\} \quad \text{with}$$

$$\alpha \in \{1, 2, 3, \dots, \omega\}, \beta \in \{1, 2, 3, \dots, \eta - 1\}, i \in \{1, 2, 3, \dots, \delta\}, j \in \{1, 2, 3, \dots, \gamma - 1\}$$

such that these opinions mimic the family heads' opinions

$$\sigma_{i,j,k}^{\alpha,\beta} = \sigma_{i,j,k}^{\alpha,\beta-1} + \lambda(\hat{\sigma}_{i,k}^{\alpha,\beta} - \sigma_{i,j,k}^{\alpha,\beta-1}) \quad \text{with } \alpha \in \{1, 2, 3, \dots, \omega\},$$

$$\beta \in \{1, 2, 3, \dots, \eta - 1\}, \quad i \in \{1, 2, 3, \dots, \delta\}, \quad j \in \{1, 2, 3, \dots, \gamma - 1\}, \quad k \in \{1, 2, 3, \dots, \epsilon\}$$

Once $\hat{F}^{\alpha,\eta}$ has been determined the next generation set of family heads $\hat{F}^{\alpha+1,0}$ can be computed using Equation 1 and subsequently the next generation society, $G^{\alpha+1,0}$ can be determined. This process may be continued for an arbitrary number of generations or until a suitably successful AI has been found.

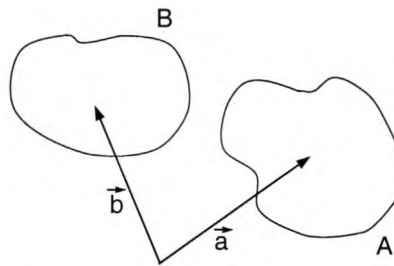
6.3 Implications of the AI model

The AI model discussed in Section 6.2 does not fundamentally alter established principles of genetic algorithms, however, it does offer several decisive advantages over traditional genetic algorithms:

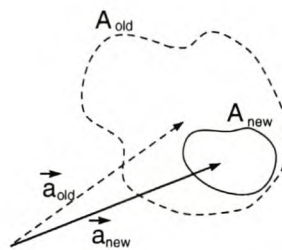
- The philosophical arguments on evolution upon which the model is built are less inflexible than existing theories on biological evolution. It is thus possible to extend current constraints in genetic algorithms in a manner that is justified by the underlying principles.
- The AI model elegantly integrates local learning in terms of the Baldwin effect [6, 22, 29] whilst partially mitigating the effects of Hiding [38]. This is accomplished by performing localised learning on distinct subsets (families) as opposed to a complete generation (society) set: Individual families will tend to locate local optima within their vicinity, whereas globally families will tend to be dispersed due to the effect of genetic cross-overs.
- The AI model offers several tiers within which solution searching may take place. On the upper-most level global searching is performed by a genetic algorithm through the use of a cross-over function, on an intermediate level localised learning is performed through the process of mimicking, and, on a lower-most level⁶, local learning could be performed by individual members of a family.
- Learning through the use of mimicking spans the gap between truly global and truly local learning methods (see Figure 3 for a graphic representation). Global searching is performed by a classic genetic algorithm through the use of a cross-over function and local learning is implemented by individual family members. Mimicking differs substantially from these methods as:

⁶Such a lower-most layer of solution searching is performed by the mutation function within the AI model, however, more effective methods of individual learning can be implemented by noting that on the most elementary level a family member is represented as an artificial neural network. Thus any ANN learning method [48] such as backpropagation, gradient search or other stochastic searching can be used to optimize the behaviour of an individual family member.

- Mimicking, by definition, cannot be performed by an individual and thus is not, in the true sense, a local learning method.
- Although mimicking could be applied globally between any set of two or more individuals it is unlikely that such a technique would yield efficient results — the global search space is generally too unpredictable to make a global form of mimicking any more effective than a random guess, however, applying mimicking globally does encourage the Hiding effect and should thus be avoided.
- Mimicking performs a vicinity-wide, inter-local solution search: The process of mimicking linearly interpolates the solution of one family member towards the solution of a more successful family member. Since members of a family span a relatively localised vicinity in solution space this process effectively searches the vicinity for its local optimum.



(a) Global and local searches for two families, \vec{a} and \vec{b} represent the family head, A and B define the search space for local searching.



(b) A family after the first iteration of mimicking. A new family head \vec{a}_{new} is chosen and the local search space A_{old} is refined to A_{new} .

Figure 3: Global and local search 3(a) and narrowed search vicinity through mimicking 3(b)

The implementation of the project is described in the next chapter, including implementation details of environmental simulation, the articulated body, AI model parameters and the evolutionary fitness function. Furthermore the encoding strategy used as input and output to the artificial neural network is discussed. The encoding strategy forms the basis for AI variations that are used for comparative studies on the project results.

Chapter 7

Implementation

This chapter discusses details of the project's implementation. Physical and environmental simulation, articulated design considerations, AI model parameters and fitness function, neural network input and output and finally AI variations are presented.

7.1 An implemented foundation

7.1.1 Physics and environmental simulation

The Open Dynamics Engine allows a nearly effortless integration of existing code with the physics simulator. Straightforward initialization and deinitialization mechanisms instantiate and release the engine and a physical simulation step is performed with a call to a simulate procedure. The only additional requirements on the programmer's part is to ensure that suitable environments and bodies are created that are subject to simulation.

The project makes use of a minimalistic environment consisting of a flat, infinite plane that forms the ground upon which bodies move. A gravitational force acts perpendicularly onto the plane and all collisions give rise to temporary frictional forces.

Collisions are tested for between every body segment and the environment, as well as between body segments. A collision is ignored if it is detected between two body segments that are connected with each other through a joint, this allows those segments to penetrate each other and only be constrained by the joints and constraints defined over the segments. Body segments still appropriately collide with the rest of the body.

An AI robot body is defined by the specifications of an articulated system. As these specifications are parsed (for a complete listing of the specifications refer to Section B.3) appropriate joint, physics and geometry structures are created within the physics engine. The specifications

designate type, location and properties of these structures.

Any physical property that is not explicitly defined within the specifications are assigned with a default function. This includes the mass of body segments as well as maximum force that can be exerted by angular motor joints. The maximum force that can be exerted by an angular motor joint is set to 750 Newton within the project, in other words, any control command issued by the AI to a joint is accommodated by at most 750 Newton.

The mass of body segments is computed based on the euclidean size of their dimensions. This means that a body with dimensions (x, y, z) is assigned a mass of $\sqrt{x^2 + y^2 + z^2}$. Such a model for the body mass ensures that larger body segments possess a larger mass than smaller segments. However, the mass does not increase linearly, instead the model assumes that larger body segments possess a lower density than smaller body segments. The only exception to this model for mass is the head. Its density is computed as three times higher than that of other body segments with equivalent dimensions. This model for mass has not been derived from research on the properties of the human body, instead it has been determined empirically to satisfy the requirements of the project by the author.

7.1.2 Articulated design

Designing and implementing an appropriate articulated system specification is not a trivial task. For the purposes of this project the articulated body is required to project a human-like expressiveness using a relatively limited amount of physical and visual cues.

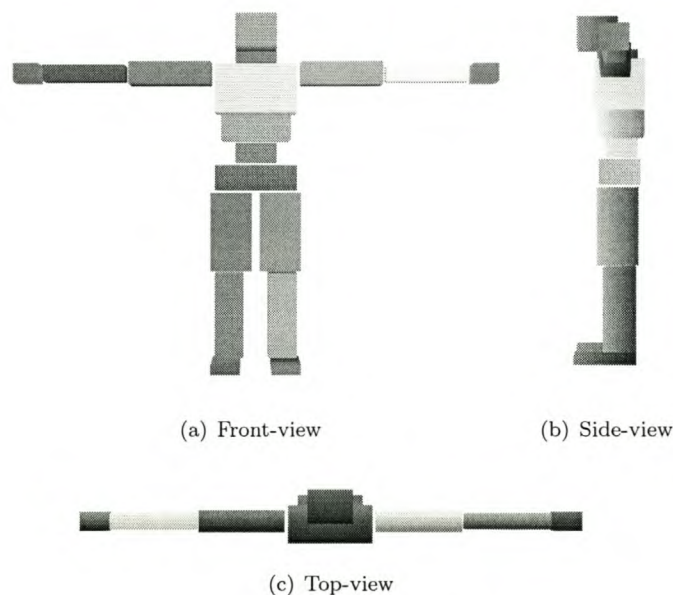


Figure 4: The articulated system

To achieve the required, elusive human quality a combination of realistic segment proportioning and joint constraints was used: Each body segment is specified with dimensions, and is positioned in a manner, that offers a reasonable approximation of the human body, each segment is connected to other segments in a manner that resembles corresponding joints in the human body, and each joint possesses a degree of freedom and constraints that closely matches equivalent human joints.

These specifications were designed around a body consisting of only eighteen rectangular prisms and seventeen joints. Figure 4 shows the resulting articulated body from various perspectives.

Though the resulting articulated body possesses more than just a passing human resemblance the visual representation still is highly abstract. In static images such a representation can only offer a limited selection of visual cues to suggest a human element; this limitation is, however, supplemented in dynamic scenes through the realistic animation of the various body segments.

The animation of the articulated body through simulated physical forces and voluntary AI forces results in a visual experience that creates convincing human poses and motions. Although the sense of motion is greatly limited in static scenes, the images in Figure 5 nonetheless instill a real sense of human motion, despite the restrictions on physical and representational resources that have been placed on the articulated system.

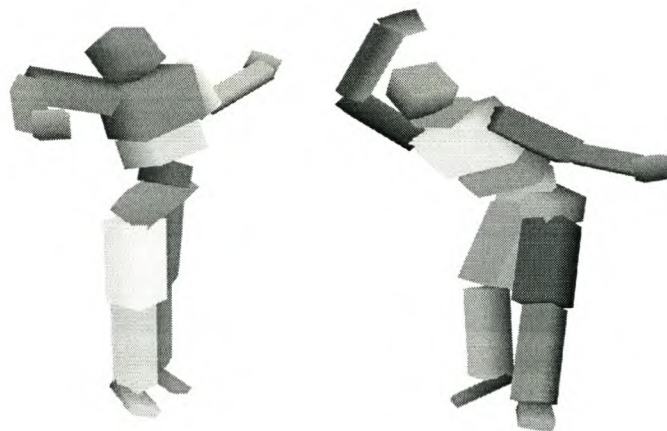


Figure 5: Human-like expressiveness

7.1.3 AI model parameters

The evolutionary AI model used within the project has been extensively detailed in Section 6.2. This section shortly discusses the implementation of that model within the confines of the project.

The AI model implements evolutionary learning over successive generations. Each generation consists of a society of a hundred families ($\delta = 100$). Each family is represented by the most successful family member, the family head. The genetic code of a family head represent ideas and opinions within the model and are used as weights for a neural network that controls the voluntary behaviour of the simulated AI robot.

Each family is processed in turn. The current family head is used to create nine replicas of the family head which are subsequently mutated.

Each of the ten family members ($\gamma = 10$) is simulated and the most successful family member is determined. The remaining nine family members mimic the ideas and opinions of the most successful member before another simulation run is performed. Again the most successful family member is determined and mimicked by the remaining family members ($\eta = 2$). A final simulation run determines the most successful member of the family, which is subsequently declared the head of the family.

The fitness values of the family heads are sorted into a fitness tree (a binary tree). Based on the enumeration of the fitness values in the tree each family head receives tickets which are used in a fitness lottery used for a genetic cross-over function. The least successful candidate receives one ticket, the next receives two tickets, then three, four, and so forth up to the best candidate who receives a hundred tickets.

The next generation is created by a layered process:

- The fittest family head is automatically transferred to the next generation. This is known as elitism, which ensures that the top performers of subsequent generations are at least as successful as the fittest member of the current generation.
- 30% of the next generation's family heads are directly transferred from the current family. The individuals are selected by picking a random ticket and transferring the owner of the ticket to the new generation. Selected tickets are reinserted into the lottery. This process ensures that a selection of established, successful individuals are likely to be present in the next generation. Less successful candidates are less likely to be selected, however, some less successful individuals are likely to be selected nonetheless. This ensures genetic diversity.
- 10% of the next generation are generated randomly. Though the odds are slim that such a random creation will be a successful individual within the generation, it is likely that some of these random additions will contain genetic material that will contribute to subsequent generations. This ensures that a continuous stream of fresh genetic material is present within the project, thereby promoting an effective search of solution space.
- The remainder of the next generation is created by genetic cross-over. Two distinct candidates of the current generation are determined using lottery selection. A new individual,

the child, is created from the two selected candidates, the parents, using a genetic cross-over function. Each gene of the child is determined by randomly selecting one of the corresponding genes of the parents. The parent that was selected first is considered the dominant parent and each of its genes has a 75% chance of being selected for the offspring, the genes of the secondary parent correspondingly have a 25% of being selected for the child:

The majority of the next generation is created using the process of genetic cross-over, a classic approach in genetic algorithms. The cross-over function ensures that the problem's solution space is effectively searched globally.

Within the project the cross-over function is subject to two mutually exclusive constraints: Ideally each parent should, on average, contribute an even amount of genes to its offspring to promote an effective global search of solution space. However, an even contribution of genes from both parents tends to degrade the overall fitness of the offspring (due to the tendency of the resulting genes to display qualities similar to randomly generated genes). Instead, a dominant and a secondary parent are chosen who contribute an uneven distribution of genes to the child (on average 75% and 25% respectively). This ensures that the child maintains strong elements of the solution present in the dominant parent, whilst still possessing a sufficiently large influx of secondary genes to perform an effective global search.

7.1.4 A question of fitness

The choice of fitness function is of paramount importance in genetic algorithms — it determines the learning emphasis that is placed on individuals by promoting and suppressing solution elements. Within this project the fitness function has progressed through several different incarnations before settling on a final function.

All fitness functions implemented emphasized some combination of distance covered by the AI robot and the height of the body during simulation, however, most of these functions also exhibited unexpected behaviour of the AI roboter. The final fitness function possesses the following properties:

During each simulation step the fitness is incremented by the following function:

$$\text{fitness}_{new} = \text{fitness}_{old} + \mathbf{f}(\text{current state})$$

where $\mathbf{f}(\text{current state})$ is a current fitness rating function that behaves as follows:

$$\begin{aligned} h_{prior} &= \text{normalized height achieved during prior simulation} \\ h_{cur} &= \text{normalized height of the body now} \end{aligned}$$

$$\begin{aligned}
 h &= \min(h_{prior}, h_{cur}), \quad h \in [0, 1] \\
 d &= \max(0, \text{distance travelled straight forward}) \\
 \mathbf{f}(\text{current state}) &= h^2 \times (0.001 \times d + 0.01) .
 \end{aligned}$$

In other words, the current fitness rating considers the minimum height achieved to that point of the simulation ($0.01 \times h^2$), as well as the distance travelled forward, emphasizing the height used to travel the distance ($0.001 \times d \times h^2$).

The fitness function is complicated slightly by several early-out tests. These tests allow candidates that perform poorly to be ignored in favour of more promising candidates:

- Each AI robot is simulated for at most five hundred time steps, if at any time during this simulation the body's height falls below 75% of standing height, then simulation of that robot is halted and it is considered a *dropped* AI. No additional fitness evaluations or simulations are performed, the AI learning function proceeds to the next available candidate.
- A comparative early-out test is performed: After 40% of the AI simulation time has passed each simulation step is accompanied by an early-out test. If the current AI does not achieve at least 40% of the fitness of the best AI of the prior generation (scaled according to the percentage of time that has passed), then execution of that AI is halted. Unlike *dropped* AI robots, however, the fitness of AIs that fail a comparative early-out test is scaled to the full simulation time (thus, if the fitness was 2 after 40% of the simulation time, then the final fitness credited to the AI will be $2 \times \frac{100\%}{40\%} = 5$). In addition to this, if less than 70% of the complete simulation time has passed, then that AI is considered *rejected*. The AI learning function proceeds to the next available candidate.

7.1.5 Choice of parameters

The experimental system that has been designed possesses a great number of parameters. Consequently the question of which values for parameters are suitable for simulation purposes is difficult to answer.

An educated guess was used to determine initial parameter settings. This was followed by a lengthy period of experimentation in which parameters and the simulation process were subject to modification.

Early-out rejection tests in particular were exposed to considerable experimentation, as early attempts required in excess of an hour of computational time to simulate a single generation. Simulations therefore had to be able to process individual AI attempts quickly, efficiently eliminating poor AI solutions and encourage the proliferation of strong AI candidates.

Most model parameters saw relatively little variation throughout the testing phase of the

project. Examples of such parameters include the number of families within a generation (one hundred) and the number of iterations over which a family is simulated (three):

The number of families within a generation was chosen to be one hundred to offer a sufficiently large population to efficiently make use of global solution searching through the use of genetic algorithms, whilst ensuring that a given generation could be simulated relatively quickly.

The number of iterations a family is simulated was chosen in a manner to balance the requirement of rapid simulation with the need to find local optima: Only one or two iterations are not sufficient to effectively search the local solution space. On the other hand as little as five iterations shrink the local solution space to approximately 6% of its original radial size through the process of mimicking.

Considering that multiple family members are searching such a reduced space five iterations may seem somewhat excessive. Viable iteration counts were thus determined to be three or four. The parameter was chosen to be three as less iterations implied faster overall simulation turn around times.

Other parameters, such as the mutation rate, were determined through a reasonable initial estimate. These parameters were only subjected to change when experimental results indicated that the value was poorly chosen.

7.2 Neural input and output and AI variation

The choice of input encoding in artificial neural networks directs the form of solution space search that the ANN can perform [48]. This section presents the problem of ANN input and output formats in AI robotics.

Artificial neural networks generally perform best when input is of a qualitative, as opposed to quantitative, form [48]. In other words, given a variable $x \in \{1, 2, 3, 4, 5\}$ a quantitative input representation as shown in Figure 6(a) is more difficult for an ANN to interpret than a qualitative representation, as shown in Figure 6(b), where each possible value for x is encoded explicitly as a separate input weight.

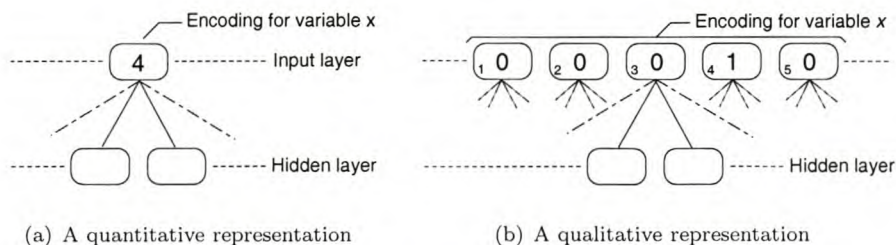


Figure 6: Quantitative and qualitative representations of variable x

Alternatively, instead of a unary encoding a more memory efficient binary encoding may be chosen. Such qualitative representations are easily implemented for sets of discrete values. However, for continuous variables, or even two-, three- or higher-dimensional variables such as vectors, a qualitative encoding represents a compromise between accuracy, memory usage and computational time.

To illustrate, given a continuous variable, for example x such that $x \in [0, 1]$, then the question arises how to encode such a variable qualitatively. In the case of x we may chose to use two inputs, one for $x < 0.5$ and another for $x \geq 0.5$, though such an encoding is accompanied by a considerable loss in fidelity of the data contained in x .

Alternatively we may choose to use a qualitative input of ten, twenty, a hundred or even more divisions for x . In each case we retain a greater resolution of the data within x , but this increase in information is coupled with an increase in memory usage and computational expense.

To minimise the memory footprint, loss of information and computational requirements a qualitative encoding of a variable should always consider the distribution of that variable. If, for example, it is known that $x \in [0, 1]$ and that x tends to be near zero then we might chose an encoding for x that allocates more inputs for values closer to zero than for values closer to one.

7.2.1 The first dimension

In this project a novel encoding strategy is used that forms a compromise between quantitative and qualitative encodings. As far as the author could ascertain no AI model has been published that makes use of such an encoding scheme. The technique can be summarised as a weighted qualitative encoding, the following description considers the one-dimensional case only:

Given

$$\begin{aligned} x \in [\alpha, \beta] & \quad \text{a variable } x \text{ between bounds } \alpha \text{ and } \beta, \text{ where } \alpha < \beta \\ n & \quad \text{number of subdivisions boundaries within } [\alpha, \beta] \end{aligned}$$

and defining

$$\rho = \frac{\beta - \alpha}{n} \quad \text{subdivision step size,}$$

then a set of subdivision boundaries is created as follows:

$$B = \{b_0, b_1, b_2, \dots, b_n, b_{n+1}\}$$

such that

$$b_i = \alpha + \rho \times (i - 0.5) \quad \text{with } i \in \{0, 1, 2, \dots, n, n + 1\} .$$

Note that there are a total of $n + 2$ subdivision boundaries, two of which are outside the boundaries described by α and β : $b_0 < \alpha$ and $\beta < b_{n+1}$.

Using these boundaries we can compute a set of $n + 1$ input weights

$$W = \{w_0, w_1, w_2, \dots, w_n\}$$

that belong to $n + 1$ partitions

$$S = \{s_0, s_1, s_2, \dots, s_n\}$$

where a given w_i refers to a particular subdivision s_i and each subdivision is bounded such that

$$s_i \in [b_i, b_{i+1}) \quad \text{with } i \in \{0, 1, 2, \dots, n\} .$$

Finally we define individual input weights based on the coverage of $x \pm \frac{\rho}{2}$:

$$w_i = \begin{cases} 0 & \text{if } b_i \geq x + \frac{\rho}{2}, \\ \frac{1}{\rho} (x + \frac{\rho}{2} - b_i) & \text{if } b_i < x + \frac{\rho}{2}, \\ 0 & \text{if } b_{i+1} \leq -\frac{\rho}{2}, \\ \frac{1}{\rho} (b_{i+1} - x + \frac{\rho}{2}) & \text{if } b_{i+1} > x - \frac{\rho}{2}, \quad \text{with } i \in \{0, 1, 2, \dots, n\} . \end{cases}$$

For example: Given $x \in [0, 5]$ we may choose to compute a qualitative representation with $n = 5$ and subdivision boundaries and bounds as illustrated in Figure 7.

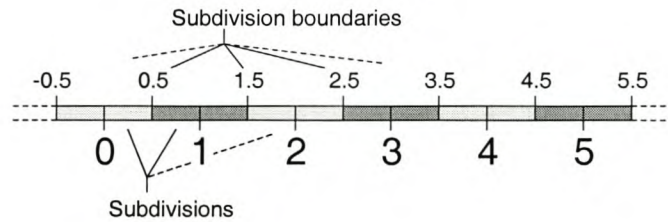


Figure 7: Bounds for $x \in [0, 5]$

Thus $\rho = \frac{5-0}{5} = 1$ and $b_0 = -0.5, b_1 = 0.5, b_2 = 1.5, \dots$, and $b_6 = 5.5$.

Assuming that $x = 1.8$ we can depict the placement of x within the subdivisions as shown in Figure 8 below, which also demonstrates that the input weights are computed based on the coverage of $x \pm \frac{\rho}{2}$ along the value axis:

Thus the input weights are $w_0 = 0, w_1 = 0.2, w_2 = 0.8, w_3 = 0, w_4 = 0$ and $w_5 = 0$.

Such a weighted qualitative encoding has the advantage that no information regarding variable x is lost. This can be demonstrated by showing that it is possible to compute the original value of variable x from the input weights, partition boundaries, n and ρ . If

$$j = \min \{i \in \{0, 1, 2, \dots, n\} \mid w_i > 0\}$$

then

$$x = b_{j+1} + \frac{\rho}{2} - \rho \times w_j .$$

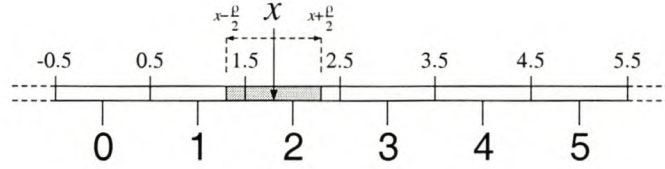


Figure 8: Coverage for $x \in [0, 5]$

Following the example portrayed in Figure 8 the original value of x can be computed as:

$$\begin{aligned}
 j &= \min \{i \in \{0, 1, 2, 3, 4, 5\} \mid w_i > 0\} \\
 &= \min \{1, 2\} \\
 &= 1 \\
 x &= b_2 + \frac{\rho}{2} - \rho \times w_1 \\
 &= 1.5 + \frac{1}{2} - 1 \times 0.2 \\
 &= 1.5 + 0.5 - 0.2 \\
 &= 1.8 .
 \end{aligned}$$

7.2.2 The higher dimensions

The one-dimensional case described above can be extended with little difficulty to higher-dimensional descriptions. Figure 9, Figure 10 and Figure 11 below indicate possible weighted quality encodings of one-, two- and three-dimensional variables using a partition with three subdivisions per value axis.



Figure 9: A one-dimensional 3 subdivision partition with coverage of a variable

When vectors or other forms of higher-order continuous data serve as input to an artificial neural network, the need for an efficient qualitative input increases dramatically. Given, for example, a vector $v \in \mathcal{R}^3$ it is possible to perform a component-wise encoding of v that makes use of five partitions per component ($n = 4$). Since each component is encoded separately a total of fifteen (3×5 , generally $3 \times (n + 1)$) ANN inputs are required.

This appears to be a reasonably inexpensive encoding of three-dimensional data, however, it does not explicitly account for the spatial information contained in v . Instead a geometrical

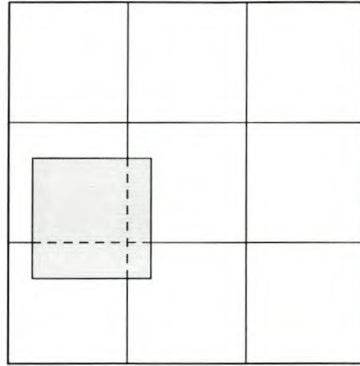


Figure 10: A two-dimensional 3×3 partition with coverage of a variable

qualitative encoding utilizing a three-dimensional partition should be used. In this case, however, if each dimension is encoded with five partitions a total of 125 ($5 \times 5 \times 5$, generally $(n+1)^3$) ANN inputs are required — a cubic increase in computational and storage requirements.

A weighted qualitative encoding for higher dimensions can assist in carrying some of the computation and storage burdens: Since a weighted qualitative encoding performs a lossless encoding of the information within v it is possible to make use of considerably fewer partitions than for a pure qualitative encoding.

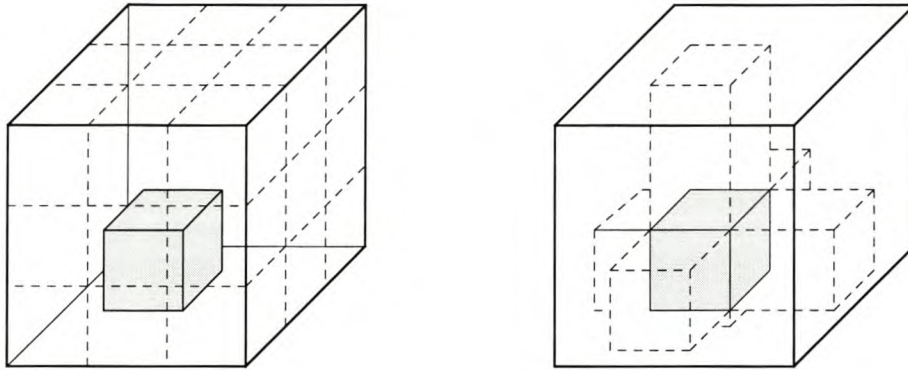
The weighted qualitative encoding forms a compromise between qualitative encodings and quantitative encodings. An artificial neural network is likely to produce better results with a sufficiently large qualitative encoding than with a smaller weighted qualitative encoding. However, at a given partition size a weighted qualitative encoding includes more precise information than a qualitative encoding and can be expected to yield improved performance on artificial neural networks.

7.2.3 The output

All artificial neural networks used within this project have in common that they make use of a probabilistic, weighted qualitative output. This form of output is described next:

Each AI is required to output three desired angular velocities per joint, corresponding to the three principle axis of the joints. Each ANN allocates twenty-seven output nodes per joint in the articulated system. The twenty-seven nodes are interpreted as a $3 \times 3 \times 3$ partition and the weights in the nodes form the probabilistic output of the neural network.

Each $1 \times 1 \times 1$ sub-partition of the larger $3 \times 3 \times 3$ partition is associated with a directional



(a) A three-dimensional partition with coverage of a variable

(b) Depth cues for the coverage of the variable in Figure 11(a)

Figure 11: A three-dimensional $3 \times 3 \times 3$ partition with coverage of a variable, 11(a), and corresponding depth cues, 11(b)

vector:

$$\begin{array}{ccc}
 (-1, -1, -1) & (-1, -1, 0) & (-1, -1, 1) \\
 (-1, 0, -1) & (-1, 0, 0) & (-1, 0, 1) \\
 & \vdots & \\
 (1, 1, -1) & (1, 1, 0) & (1, 1, 1)
 \end{array}$$

An intermediate vector is constructed by computing the sum of all directional vectors, scaled by their corresponding output weights. Finally the output vector is computed by scaling the intermediate vector by the inverse of the sum of all the partitions weights. The three components of the output vector are interpreted as the three desired angular velocities of that joint.

7.2.4 AI variations

This project makes use of several AI variants to broaden the scope of the data collected. Five forms of AI have been used, the difference between them is the form of information that is input to the artificial neural network that is used to control the behaviour of the AI roboter.

Each artificial neural network possesses three layers: An input, a hidden and an output layer. The various AI variants have different-sized input layers, however, to ensure an even comparison between the different AI forms the hidden and output layers are the same size.

The types of AI used distinguish between (weighted) qualitative and quantitative encodings, as well as the type of information — segments or joints — that is provided. Additionally a reference AI was used that only inputs a periodic stimulus to the neural network:

- *Quantitative Vector* — an encoding that inputs direct, quantitative values to the neural

network that controls the AI robot. Three sets of vectors are used as input that specify information on the segments of the body: Two sets of vectors specify position and velocity of segments relative to a reference segment, the base segment. An additional set of vectors is input that feeds the output of the network on the prior step back into the neural network. This makes the ANN a recurrent network [48] that can make use of time-dependent, historic data.

- *Qualitative Vector* — an AI variant similar to the *Quantitative Vector* variant; the only difference is that position, velocity and prior output data is input to the ANN using a weighted qualitative encoding as described in sections 7.2.1 and 7.2.2.
- *Quantitative Joint* — makes use of a quantitative encoding of joint information within the body. Three sets of vectors are used as input that detail information on the joints that are present in the articulated system: Two sets of vectors specify the angles and angular velocities of the angular motors in the body. Finally, equivalent to the *Quantitative Vector* variant, the neural network's prior output is fed back into the network.
- *Qualitative Joint* — inputs the same information to the artificial neural network as the *Quantitative Joint* variant, however, the information is input with a weighted qualitative encoding.
- *Qualitative Sin* — a reference AI variant. Biped walking can, in principle, be interpreted as a periodic function. This implies that the only input required for a simple biped walker is a periodic stimulus (such as a sinus wave). This AI variant makes use of such an interpretation of biped walking and, correspondingly, the input is a qualitative encoding of the sin function. Unlike the other AI variants used, this neural network is not recurrent.

After implementation details had been finalised the project proceeded to gather data on the performance of AI variants. The results are presented in the following chapter.

Chapter 8

Results

This section presents the data that has been generated over the course of AI simulation. Well in excess of a thousand hours of computation time have been spent in an attempt to find an optimal solution strategy for controlling the AI bodies.

To understand the significance of the results in this thesis, both worst and best case scenarios need to be considered. A worst case scenario can be defined as the result that occurs when no AI governs the host body. In this case only physical forces act on the body and it falls to the ground. Figure 12 illustrates the corresponding result within the project.



Figure 12: A fall without AI guidance. Each image denotes a 0.8 second interval.

Ideally an AI should be able to achieve sufficient control over its designated articulate system to mimic human motion. It should be able to stand, walk at varying speeds, turn, ascend and descend stairs, recover from falling down — and demonstrate human grace while doing so. For the purposes of this project such an idealised result is utopian; instead this project defines a more realistic best case scenario that requires the AI roboter to be able to stand and to walk.

The most capable AIs that have been evolved in the course of this project fall considerably short of such a best case scenario. They do, however, progress well beyond the worst case scenario by demonstrating intent and attempts at balance.

Each AI variant will be discussed in turn, starting with the Sin variant. The discussion presents progressive AI results, illustrated by image sequences similar to Figure 12 above, four graphs detailing AI characteristics over the course of several generations, and an analysis of the data. Finally, once all AI variants have been presented, a comparative analysis is performed.

The graphs are presented in a unit-less format to emphasis relative results:

- The first graph depicts the best and average fitness of the AI over several generations. Due to the elitist approach to evolution the graph of the fittest artificial intelligences is monotonically increasing.
- The second graph shows the fractions of dropped and rejected AIs, note that the following relation, $dropped + rejected \leq 1$, always holds. This is true since any given AI solution may either be dropped from a simulation, rejected from a simulation or it may complete a simulation.
- The third graph illustrates what fraction of AI families in a given generation have learned through mimicking.
- The last graph represents the amount of time that was spent on computing progressive generations. This graph may fluctuate considerably as computation of the corresponding AI variant may have been conducted on different computer systems. Furthermore, computational resources were occasionally shared with other, concurrently running programs. Nonetheless, the time graph suggests trends in computational requirements that should not be ignored.

8.1 Qualitative Sin variant

As described in the prior chapter the qualitative Sin variant serves as a reference AI variant; its input to the AI is a qualitative encoding of the sinus wave. The Sin variant received the greatest amount of exposure to simulation, the graphs in Figure 14 depict the progress of the sinus-based AI variant over the course of a thousand generations.

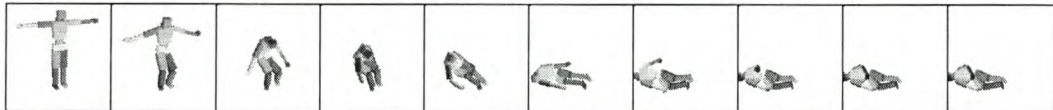
8.1.1 Image sequences

Figure 13 illustrates the behaviour of the fittest AI members for various generations within the Sin variant. Except for Figure 13(a), each AI generation that is depicted demonstrates artificial intelligence control over the articulated system.

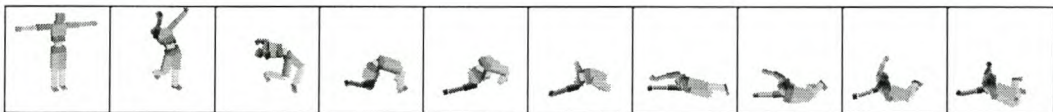
The AI behaviours in image sequences 13(c) and 13(e) convey a discernable measure of intent to maintain balance. The control that is exerted by each artificial intelligence attempts to maximise its performance against the evolutionary fitness function.

Correspondingly the AI tends to encourage behaviour that attempts to keep the torso of the articulated system as high as possible for as long as possible. This ensures that AI members are not dropped from simulation prematurely. Furthermore, evolution encourages the AI to fall forward, as forward motion is rewarded by the fitness function.

It is noteworthy that, subjectively, image sequence 13(d) behaves in a manner that is inferior to the behaviour of the AI in an earlier generation, as shown in image sequence 13(c). Such examples of subjective performance evaluations indicate that, perhaps, it is possible to evolve successful AI control through human evaluation of performance, as opposed to a simple fitness function.



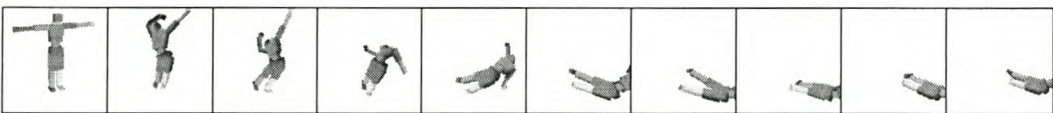
(a) Plain fall, a fall without AI intervention



(b) Generation 1



(c) Generation 36



(d) Generation 137



(e) Generation 685

Figure 13: Sin variant image sequences, images represent 0.8 second intervals.

8.1.2 Graphs

The fitness graph, Figure 14(a), of the Sin variant demonstrates a unique anomaly within the project. The average fitness of generations spontaneously increases over the course of several generations. This spontaneous rise in average fitness culminates after a new elite AI is found that supercedes the prior best of the AI variant.

It is possible that generations are created with an unusually high proportion of extremely fit AI individuals. However, it is highly unlikely that such a selection is generated continuously

over the course of more than a few consecutive generations, let alone a prolonged period of time.

A somewhat more plausible explanation is that the genes of the elite AI were propagated into a disproportionate amount of AI individuals over the course of several generations. This theory is supported by two coinciding factors:

- The graph representing the success of mimicking, Figure 14(c), falls with the rise of the graph of average fitness. This indicates that learning through mimicking was impaired — this is likely due to the difficulty to learn through mimicking if a local optimum, such as the current AI elite, dominates the genepool.
- The average fitness declines sharply after the discovery of a new elite AI: The genetic material of the successor begins to displace the gene code of the prior elite.

The statistical anomaly raises a philosophical implication. Suppose that the anomaly is a periodic, statistical certainty, that is to say: Prolonged simulation with a set of dominant genes eventually gives rise to an anomaly similar to the one noted in Figure 14(a).

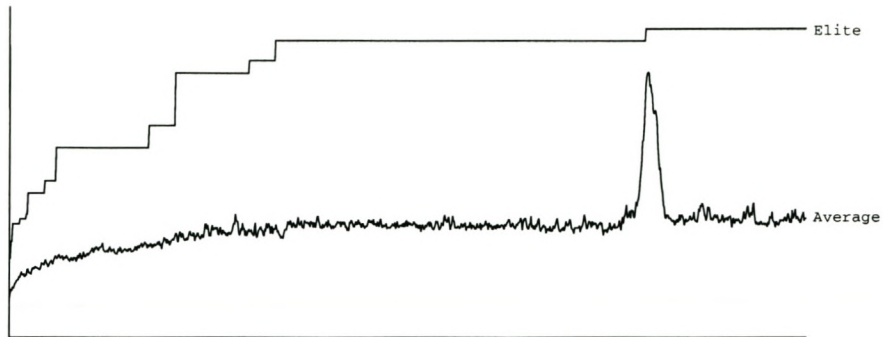
These anomalies effectively would initiate a thorough search of solution space biased towards a variation of the current elite AI. Such a search would continue until a successor to the current elite AI is found.

This in turn suggests that evolution, in general, does not only favour diverse gene pools for evolutionary progress, but that it can also very efficiently make use of a highly specialised gene-pool. Evolution applied to such circumstances attempts to find successful solution strategies through a focussed search, and — as a by product of a successful search — enriches the diversity of the gene pool in the process.

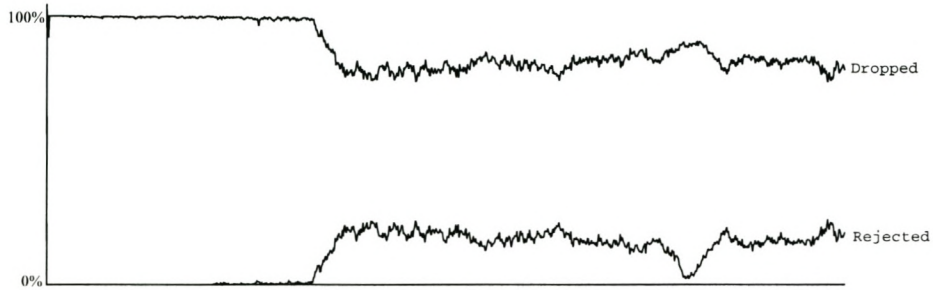
The graphs of dropped and rejected AI members indicates that initially nearly all solutions were dropped from simulation. This process apparently continued until the AI fitness reached a threshold value, after this threshold was reached the fraction of dropped solutions fell to approximately 75 – 80% and the fraction of rejected solutions rose to 20 – 25%. This trend temporarily reversed during the anomaly, but persisted again afterwards.

The efficiency of mimicking as a learning strategy is roughly 33% throughout simulation. This excludes a temporary fall in mimic-based learning during the anomaly.

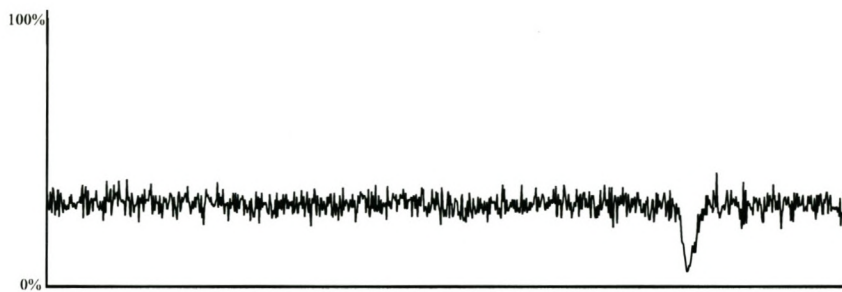
Initially the time graph starts relatively low, as little time is spent on simulation as most early AI solutions are dropped from simulation quickly. The graph reaches a plateau and maintains remarkable stability throughout the lifetime of the AI variant, with the exception of a marked drop that coincides with the fall of dropped and the rise of rejected AI solutions. It is unclear whether this decrease in computational time is due to changes within the AI variant or due to an external coincidence.



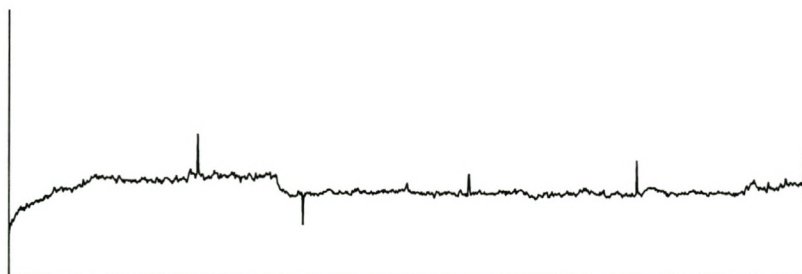
(a) Fitness



(b) Dropped and rejected individuals



(c) Fraction of successful mimes per generation



(d) Time

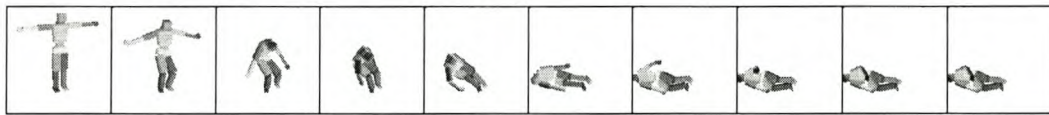
Figure 14: Sin variant graphs, 1000 generations

8.2 Quantity Vector variant

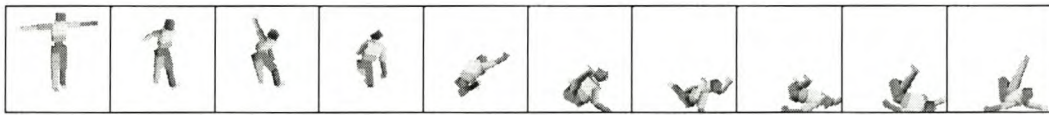
The input of this AI variant is a quantitative encoding of vectors that describe the relative position and motion of the body segments that shape the articulated system.

8.2.1 Image sequences

Figure 15 demonstrates the progress of this AI variant over the course of selected generations. The Quantity Vector variant followed a somewhat unusual pattern of evolution: The first and second generations' fittest solutions are identical, the third and fourth generations' fittest solutions are identical, and all generations afterwards have identical solutions to the fifth generations' elite AI:



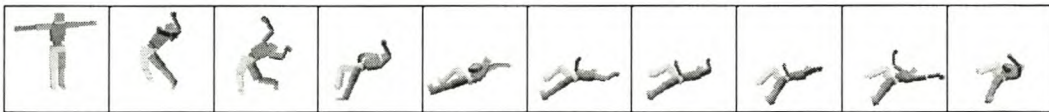
(a) Plain fall, a fall without AI intervention



(b) Generation 1



(c) Generation 3



(d) Generation 300

Figure 15: Quantity Vector variant image sequences, images represent 0.8 second intervals.

The behaviour of early AI solutions (figures 15(b) and 15(c)) are very similar, suggesting that they were acquired through mimicking. A large change in behaviour created the solution that would remain the fittest solution for the remainder of the simulation of this AI variant (Figure 15(d)); it is likely that this solution was found through genetic cross-over and a subsequent localised mimic-based search.

8.2.2 Graphs

The elite fitness graph of the Quantity Vector variant is unusually steep initially, then rapidly settles on a constant value for the remainder of the simulation. As described in Section 8.2.1 it is likely that a particularly successful solution strategy was found early during simulation through a combination of genetic cross-over and mimic-based search.

The strategy of the elite AI proved so successful that no fitter solution was found during simulation. The average AI roughly follows the trend set by the elite solution, however, it appears to move with a slight upwards tendency.

Similar to the graphs of the Sin variant, in Figure 14, the sudden rise of the elite graph appears to have surpassed a threshold value which activated variation in the graph for dropped and rejected solutions. The graph of dropped AIs averages around 80% and the graph for rejected AI solutions averages approximately 20%. This is somewhat less than the corresponding graph of the Sin variant, Figure 14(b), however the graphs in Figure 16(b) appear to possess a slight tendency to fall in the case of dropped AIs and to rise in the case of rejected AIs.

It is possible that the extent by which the percentage of dropped AI solutions falls depends on the simulation's elite AI members:

- The Sin variant and the Quantitative Vector variant produced the most successful AI solutions.
- The Sin and Quantitative Vector variants are the only simulations that produced a noteworthy fall in dropped — and rise in rejected — solutions.
- The Sin variant's solution is more successful than the Quantitative Vector variant's solution; correspondingly the fall in dropped and rise in rejected solutions is greater in the Sin variant.

The graph depicting the performance of mimic-based learning appears to be invariant of the trends displayed in other graphs. Similar to the simulation of the Sin variant mimic-based learning is successful approximately 33% of the time.

The time graph, Figure 16(d), is divided into three parts. The first and third part are similar and suggest that they were exposed to the same computational constraints. However, the second part is vastly different in that the time required for each generation is considerably higher than for generations that were computed during parts one and three.

Additionally, the time graph in the second part fluctuates considerably with sharp peaks and deep troughs. It seems likely that during the computation of the second part the AI simulator competed for processing time with another program.

Irrespective of the computational resources available, each section of the time graph possesses an upwards tendency. This is similar to the trends observed in the fitness as well as the dropped and rejected graphs.

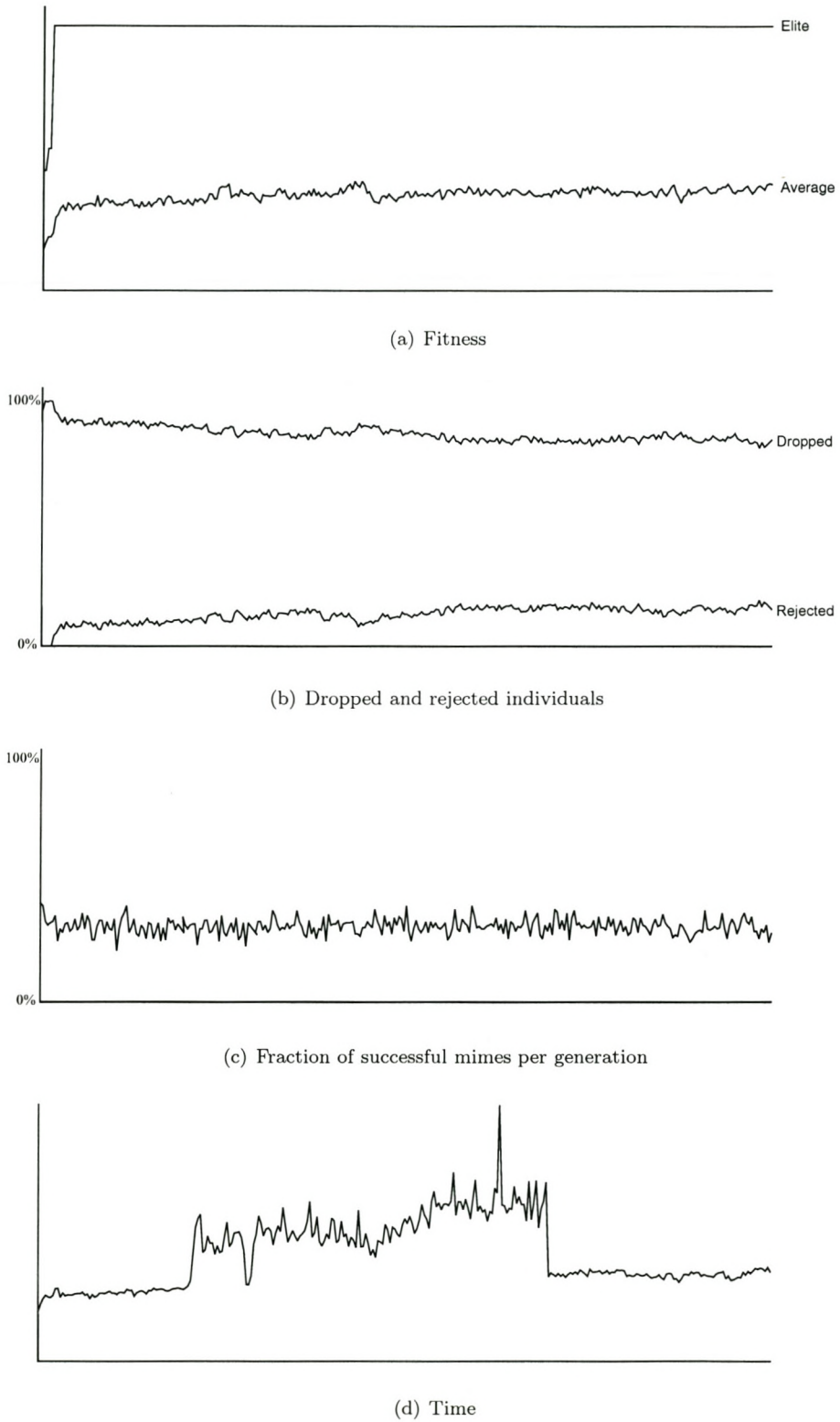


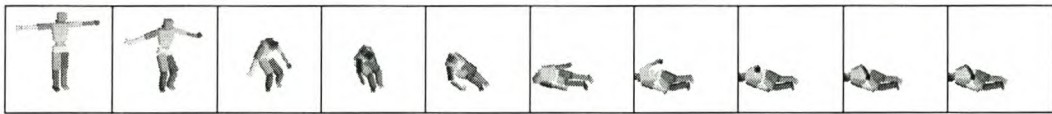
Figure 16: Quantity Vector variant graphs, 300 generations

8.3 Quality Vector variant

8.3.1 Image sequences

The Quality Vector AI variant is the unfortunate victim of circumstance. It was, subjectively, blessed with an uncommonly successful initial solution, as can be seen in Figure 17(b). It could maintain balance significantly longer than other solutions before falling over, however, unfortunately the body fell over backwards.

The fitness function it was subjected to only rewarded forward motion — thus the promising initial attempt was discarded for variations that better satisfied the demands of the fitness function. This caused the loss of an AI that conveyed a considerable degree of skill in maintaining balance in favour of AI solutions that managed to fall forward.



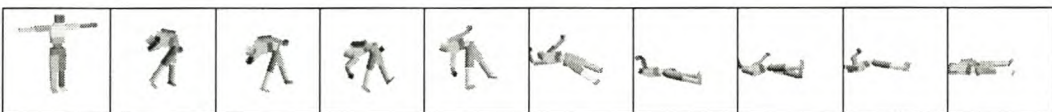
(a) Plain fall, a fall without AI intervention



(b) Generation 1



(c) Generation 30



(d) Generation 100



(e) Generation 300

Figure 17: Quality Vector variant image sequences, images represent 0.8 second intervals.

None of the subsequent generations achieves a result that is subjectively as good as the first generation's solution. The results of the Quality Vector variant strongly questions the validity

of the fitness function; perhaps a more subjective form of evaluation or a less restrictive form of fitness evaluation would have benefited this AI variant.

The solution that best accomodates the fitness function, presented in Figure 17(e), does possess some merit. It cannot match the initial solution's skill at maintaining balance, however, it does attempt to take a step forward. This is, subjectively or otherwise, significant progress towards a best case scenario compared to many other solutions.

8.3.2 Graphs

The graphs of the Quality Vector variant are straightforward and offer little surprises. The elite fitness graph displays a series of rises that become less frequent as the graph rises. The graph of average fitness demonstrates a nearly continuous upwards trend, it only appears to reach a plateau near the end of simulation.

The graph of dropped and rejected individuals, Figure 18(b), remains nearly entirely unchanged. The graph of dropped individuals tends to 100% and the graph of rejected solutions appears to be at 0% throughout the simulation.

However, whereas the graph of rejected AI attempts seems to be constant, the trend displayed in the graph of dropped solutions becomes ever more agitated with the rise of the fitness graphs. This characteristic can be noted in all graphs on dropped and rejected individuals.

The behaviour of the graph of mime-based learning of the Quality Vector variant is analogous to the mime-based learning graphs of other AI variants: Irrespective of the trends displayed in other graphs within this AI variant the mime graph displays a continuous rate of success of approximately 33%.

Finally, the time graph of the Quality Vector AI variant appears to follow the same trend as the graph of average fitness: Barring occasional peaks in computational time, due to competition for CPU time, the time graph continuously rises throughout most of the simulation and only reaches a plateau near the end of the simulation cycle.

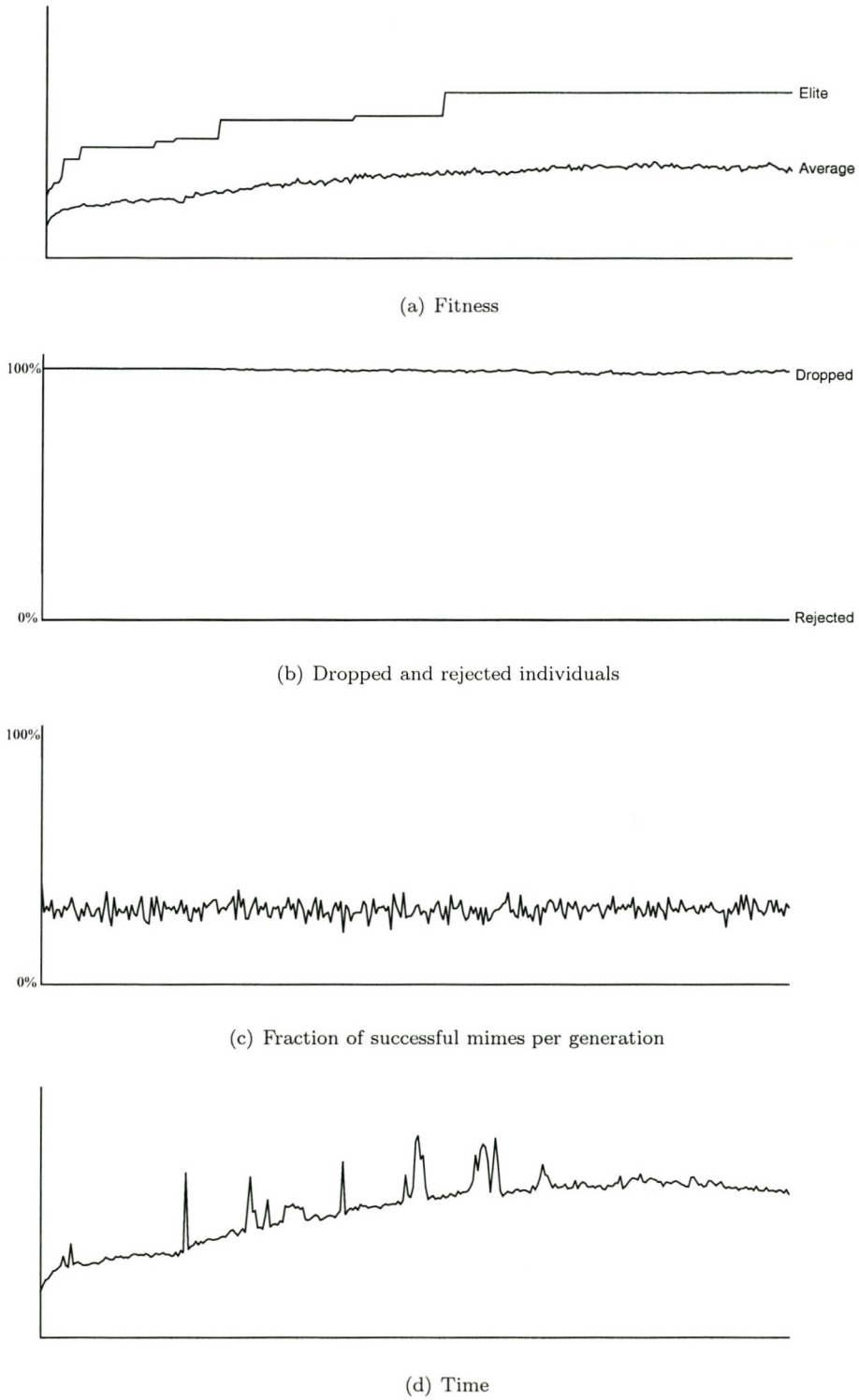
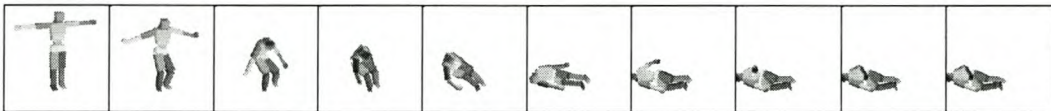


Figure 18: Quality Vector variant graphs, 300 generations

8.4 Quantity Joint variant

8.4.1 Image sequences

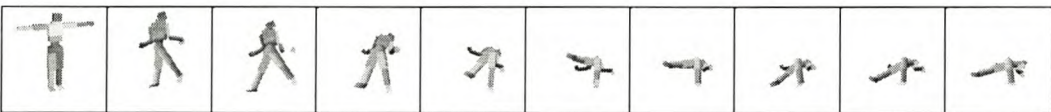
This AI variant possesses certain similarities with the Quality Vector variant: Most of the evolutionary effort of this variant was directed at creating an AI that produced a forward motion. Figure 19 below illustrates the progress that was made to achieve this goal.



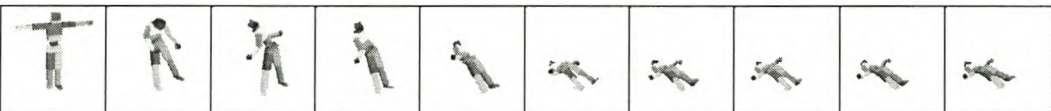
(a) Plain fall, a fall without AI intervention



(b) Generation 1



(c) Generation 100



(d) Generation 200



(e) Generation 300

Figure 19: Quantity Joint variant image sequences, images represent 0.8 second intervals.

8.4.2 Graphs

The solution strategies of elite AI members of the Quantity Joint variant appear to have gone through an unusual number of subtle alterations. Typical to the progress of other AI variants the first quarter of the simulation saw the creation of the greatest diversity of elite AI candidates. However, after the initial rush to find a successful AI strategy most variants settle into a period

of relative stability.

Surprisingly, this does not hold for the Quantity Joint variant, instead the graph of elite fitness shows that the variant produced a continuous stream of subtle improvements to the AI strategy. This gives rise to many small increases in graph of elite fitness.

Perhaps as a result of this the graph of average fitness shows a tendency to rise continuously throughout the course of the simulation. Unusually, the average fitness of the Quantity Joint variant does not show any signs of slumping performance.

Other graphs of average fitness display a definite series of larger peaks and troughs, however, the corresponding graph in this variant only demonstrates relatively subtle variation in troughs and peaks. This may be due to the continuous rise in elite fitness: Genes of elite AI members do not possess sufficient time to fully expose themselves to the population of AI individuals. Thus — statistically — it is less likely that several successive generations possess uncommonly high numbers of individuals that are very similar to, or identical to, the fittest solution (which would explain the presence of large peaks and troughs in the graph).

The graph depicting the fractions of dropped and rejected individuals, Figure 21(b), is rather similar to the corresponding graph of the Quality Vector variant, Figure 18(b). Both share a trend line of dropped individuals that remains at, or near, 100% throughout the simulation, and furthermore both graphs display an agitation of that trend line towards the later part of the simulation.

Careful observation of the behaviour the average fitness graph and the graph of dropped individuals indicates a relationship between the two. Figure 20 below displays an enlarged version of the later parts of the average fitness and dropped individuals graphs.

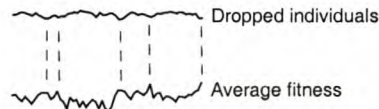


Figure 20: Excerpts of Quantity Joint fitness and dropped individuals graphs

Figure 20 portrays the subtle relationship between average fitness and dropped individuals: A peak in average fitness usually coincides with a trough in dropped solutions. Since the vast majority of solutions are dropped, this is an indication that only a small fraction of exceptionally successful solutions, such as elite solutions, are not dropped from simulation. Therefore a rise in average fitness and a fall in dropped individuals suggests an increased presence of the fittest solutions in the genepool.

Both the mime and the time graphs are similar to corresponding prior graphs. An average mimic-based success of 33% is maintained throughout simulation and the continuous rise in average fitness is accompanied by a continuous rise in the time spent on computing generations.

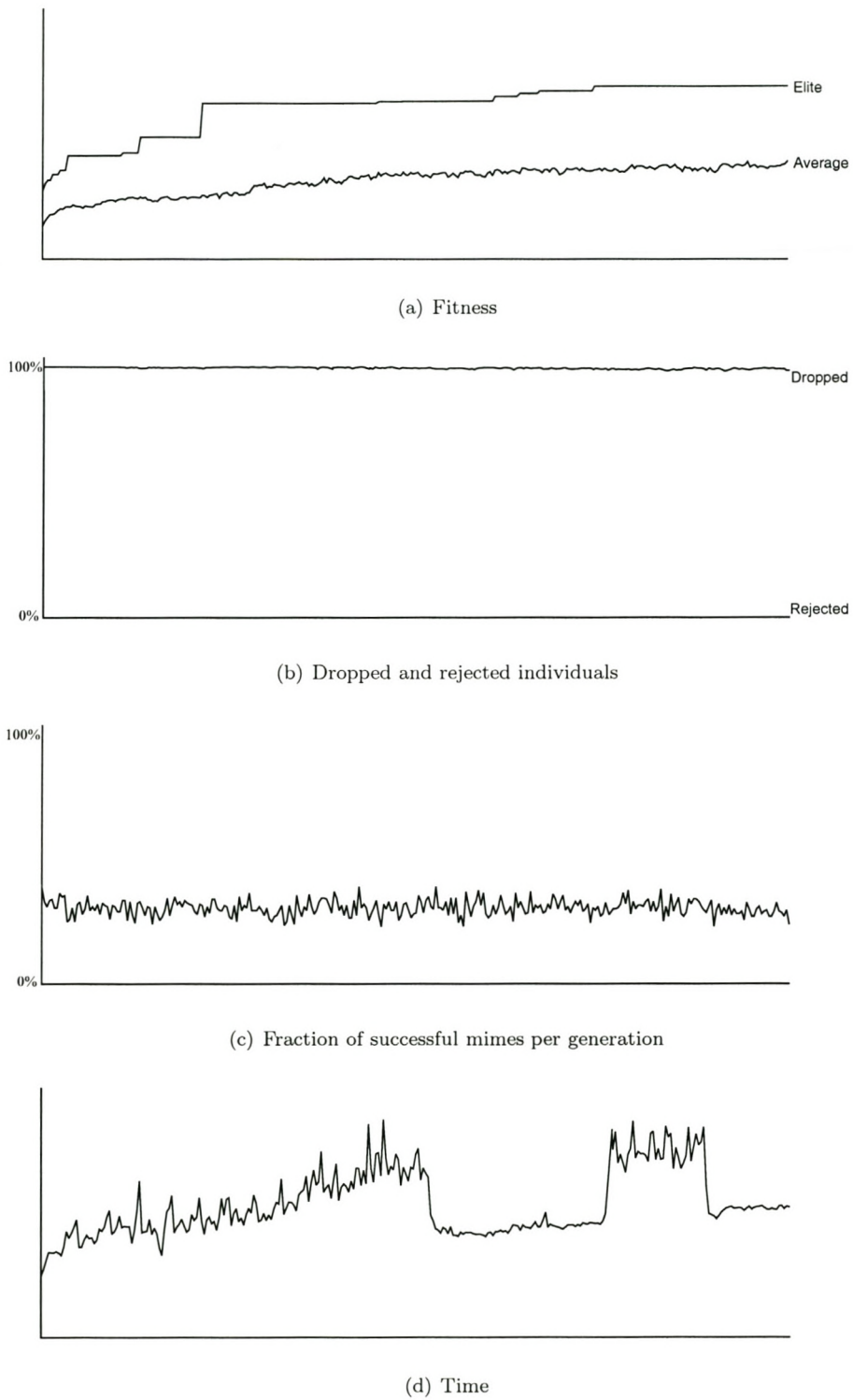


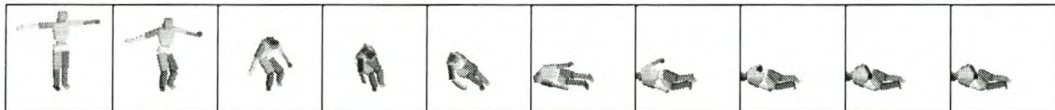
Figure 21: Quantity Joint variant graphs, 300 generations

8.5 Quality Joint variant

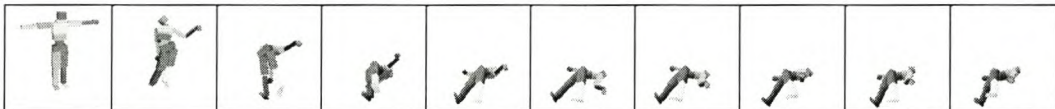
8.5.1 Image sequences

The image sequences of figures 22(b), 22(c) and 22(d) demonstrate a remarkable evolutionary trend: Each successive elite AI manages to maintain balance for a slightly longer period of time.

The AI behaviour illustrated in Figure 22(e) is slightly less adapt at standing than the AI solution shown in Figure 22(d). However, the nature of the fitness function promotes the forward motion that is exhibited by the AI, which results in the AI surpassing earlier solutions as the fittest AI solution of the simulation.



(a) Plain fall, a fall without AI intervention



(b) Generation 1



(c) Generation 100



(d) Generation 200



(e) Generation 300

Figure 22: Quality Joint variant image sequences, images represent 0.8 second intervals.

8.5.2 Graphs

The Quality Joint AI variant received a great amount of simulated attention (second only to the Sin variant) with 530 generations of simulated data. Compared to other AI variants the Quality Joint variant produced several uncharacteristic results that stand out from the results of other AI variants.

The graph of elite fitness only displays increases in the performance of elite AI individuals within the first 300 generations. This in itself is not particularly noteworthy — as most AI variants were only simulated for the first 300 generations — however, it is unusual that the increase in elite fitness in the first half of the simulation is nearly linear with evenly dispersed rises of similar magnitude. On the other hand, the trend line of elite fitness in other simulations appears to be logarithmic.

The graph of average fitness rises steadily during the first third of simulation, but then proceeds to remain approximately constant for the remainder of the simulation — although the graph of elite fitness continues to rise noticeably. This stands in stark contrast to the behaviour of the simulation in other AI variants.

The graph of the fraction of dropped individuals displays features that are comparable to corresponding graphs of the Quality Vector and Quantity Joint variants. The trend of the graph remains near 100% throughout simulation and it is accompanied by a small yet noticeable fluctuation of the graph.

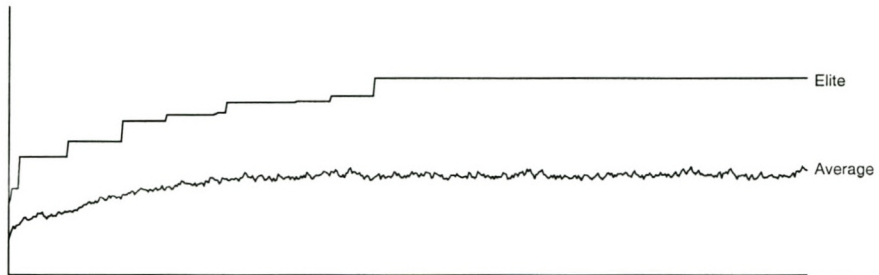
At first glance the graph of dropped individuals in the Quality Joint variant appears to be more agitated than the graphs of the Quality Vector and Quantity Joint variants. This is, however, likely to be the result of a visual artefact of the Quality Joint variant as the graph displays the results of a much larger set of generations.

Consistent with the results of all other AI variants, the graph of mimic-based learning depicts a trendline that is approximately 33% on average. The time graph on the other hand displays an unusual trend.

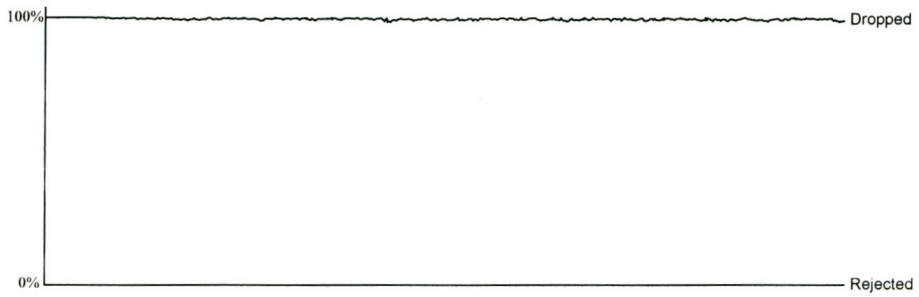
The computational time spent on each generation rises linearly within the first third of simulation in tandem with the graph of average fitness. Afterwards, however, the time graph displays a series of distinct valleys and peaks that only modestly follow the trend displayed in the graph of average fitness, as shown in Figure 23 below.



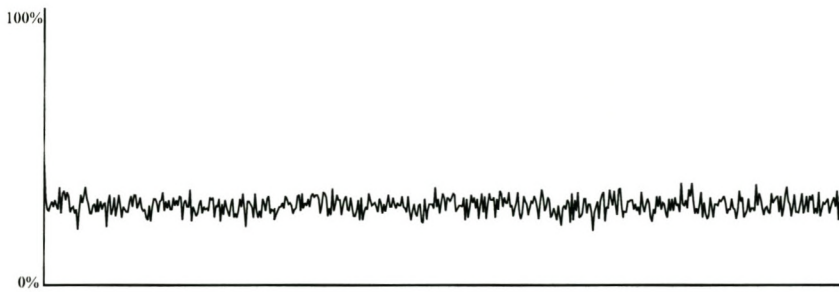
Figure 23: Graph comparing fitness and time over generations



(a) Fitness



(b) Dropped and rejected individuals



(c) Fraction of successful mimes per generation



(d) Time

Figure 24: Quality Joint variant graphs, 530 generations

8.6 Comparative analysis

This section briefly compares and describes trends and results common to the AI variants. This includes a general discussion of each primary recorded characteristic (elite fitness, average fitness, dropped fraction, rejected fraction, mime-based learning and required computational time) as well as a comparison of the performance of resulting AI solutions of the simulation variants.

8.6.1 Characteristic results

Common to each simulation is a rapid initial growth of elite fitness. This growth generally subdues slowly over time, with two notable exceptions:

- The Quantity Vector variant, Section 8.2, displayed an exceptional growth of elite fitness over the first five generations and no growth afterwards.
- The Quality Joint variant, Section 8.5, produced a continuous, linear increase in elite solutions throughout the first half of simulation. The second half of simulation was accompanied by no growth.

The Sin variant demonstrates that, although the graphs of elite fitness grow until they appear to reach a steady state, given sufficient simulation time better solutions can still be found.

Average fitness generally behaves as a function of elite fitness: An increase in elite fitness is accompanied by a gradual rise in average fitness, before a steady state is reached in which the trendline of average fitness remains constant. However, the results of the Sin variant, Section 8.1, demonstrate that occasionally surprising anomalies may occur (see Figure 14(a)) that are probably due to an unusual dominance of elite genetic material in the genepool of the population.

The fraction of dropped individuals in each generation is initially near 100%, the dropping criterion is relatively high, to ensure rapid early evolutionary cycles. As generations adapt and become more able to satisfy the fitness function the number of dropped individuals tends to decrease slightly, causing an agitation of the graph of dropped solutions — however, it still tends to be near 100%. In most simulations the graphs of rejected solutions remained at 0% throughout simulation.

Two simulations, the Sin variant and the Quantity Vector variant, displayed a considerable change in the behaviour in the graph of dropped and rejected individuals: After the fitness of these variants reached a threshold value the trendlines of the graphs of dropped individuals decreased significantly to approximately 75 – 80%.

The fall in dropped solutions in the Sin and Quantity Vector variants is accompanied with a rise in rejected solutions. The sudden increase in rejected solutions is a significant indicator of the status of evolutionary fitness within the simulation:

- AI individuals are required to possess considerable fitness to survive sufficiently long to be rejected rather than dropped from simulation.
- Solutions are rejected by competitive evaluation, the criterion for rejection is based on how well solutions perform compared to the fittest solution of the prior generation.

These two points demonstrate that a rise in rejected AI solutions is an indication for the presence of relatively fit AI solutions that are different from the elite solution. In other words, the populations of the Sin and Quantity Vector variants possessed numerous distinct solutions that achieved considerably high degrees of fitness.

The graphs of successful mimes display similar results in each of the AI variants — a trendline indicating mimic-based learning was successfully applied 33% of the time. This demonstrates that mimic-based learning can be an effective local learning method.

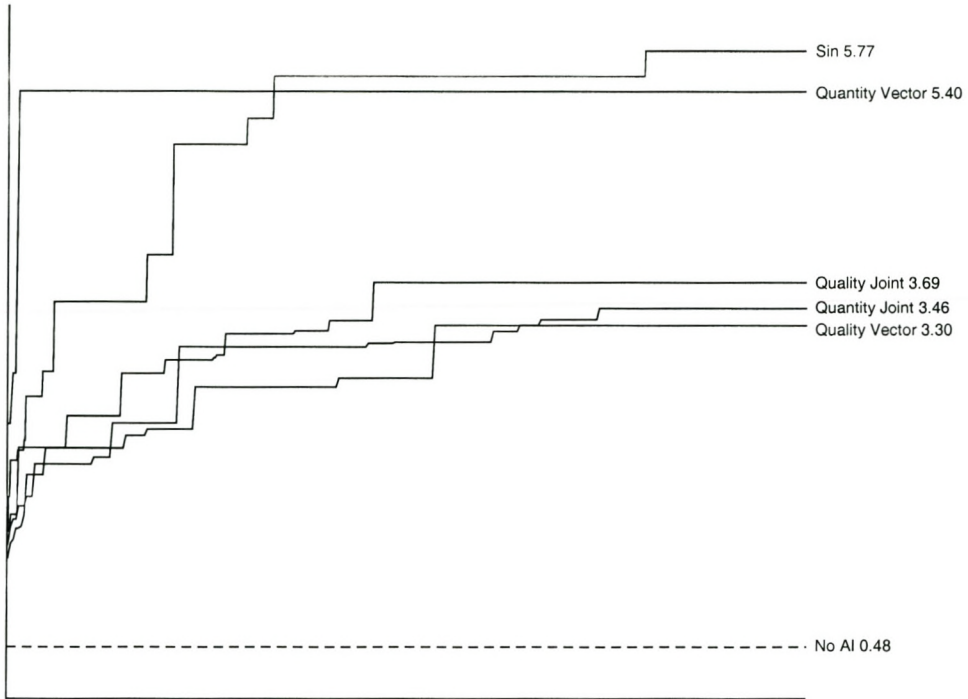
The time required to process a generation depends on a variety of conditions, such as processing power and competition for processing time. Given that such factors remain constant then the graphs of computational time are approximately proportional to the average fitness of the population within each AI variant.

8.6.2 Comparative performance

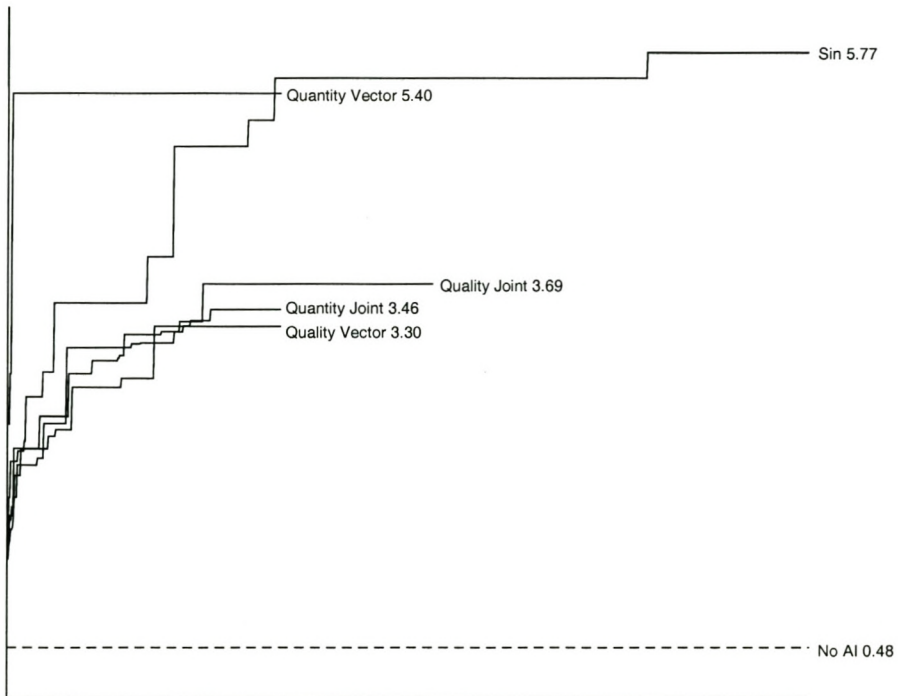
Figure 25 below illustrates the performance of AI variants. For ease of readability Figure 25(a) displays elite fitness of AI variants over the course of one simulation, this ensures that the performance of variants can be easily compared. Figure 25(b) displays elite fitness of AI variants over the course of generations, this ensures that the progress of AI variants can be correctly compared.

The graphs are labelled according to their AI variant and the value of the fittest solution found. Furthermore, note that each figure indicates the fitness of the null-AI with a stippled line.

It is apparent from the graphs in Figure 25 that the fittest solutions of the AI variants are roughly an order of magnitude more effective than the worst case scenario (the null-AI, depicted by the stippled line). This demonstrates that evolution was successful in finding AI solutions that significantly progressed from the worst case scenario.



(a) Fitness of AI variants over one simulation



(b) Fitness of AI variants over generations

Figure 25: Comparative graphs of elite fitness for each AI variant

Both the Sin and the Quantity Vector variant stand out in the figures as particularly successful. The graph of the Sin variant features a continuous series of large leaps, this suggests that a periodic function (as used as input in the Sin variant) is a useful input to an AI that is learning bipedal walking.

The graph of the Quantity Vector AI variant indicates an unusual fitness history in this AI variant. The graph starts considerably higher than other graphs and possesses only two (albeit large) rises in fitness. It is difficult to attribute the characteristics of the graph with particular qualities of the AI variant; it seems likely that the cause of this AI variant's result was a series of statistical flukes.

8.6.3 Reliability of results

The question arises to what extent the results shown in this chapter are typical. Unfortunately only a single set of official results could be made within the time available to complete this thesis. However, the simulator underwent numerous changes and optimizations before the final experiments could begin. These modifications and optimizations (to eliminate coding errors, determine suitable parameters and increase simulation speed) account for approximately a dozen distinct versions of the simulator.

Experimentation with these early builds of the simulator offered results that were quite similar to the final results. Though these early experiments could not run as extensively as final simulations the observable trends (such as initial behavior of the fitness, dropped, mime and time graphs) closely matched the trends that are present in the final results.

8.6.4 Closing words and recommendations

Although this project could not produce an AI that achieves sufficient control on its body to be able to walk, considering the high degree of freedom that the articulated systems are endowed with the results are nonetheless encouraging: When learning bipedal motion the first step, figuratively and physically, is the hardest.

An intelligence that can walk for nineteen steps is likely to complete the twentieth step successfully as well. However, an AI that attempts to learn two-legged walking will find the greatest difficulty in making the first few steps.

Interpreted in this light the fittest AI solutions display impressive progress in achieving those initial steps: AI solutions display intent and discernably attempt to maintain balance and solutions such as displayed in Figure 15(c) and Figure 17(e) demonstrate that the AI is tentatively experimenting with extending a leg to perform a first step.

It should be noted that the articulated system possesses a much higher degree of freedom

than similar systems in other projects on bipedal walking, however, the degree of freedom still falls far short of the fidelity of motion that a human body is endowed with.

Nonetheless, the range of motions and the behaviour that the articulated system displays possesses a recognizable human quality¹. It may be advantageous to investigate the possibility of using a relatively low-fidelity articulated system, as used in this project, to control the motions of an articulated system that is accurately modelled after the human body.

Generally AI systems designed to learn control motion simplify the process by applying precalculated forces to the articulated system which form the basis of the motions that need to be learned; in this thesis these precalculated forces are referred to as force animations. In the case of bipedal walking these precalculated forces enable the articulated system to perform a simple walking motion.

Strictly speaking the AI does not learn to walk, instead it learns to adapt the basic walking motions to allow the body to maintain balance and to successfully walk. Essentially the AI does not learn to control the body, but to control the kinetic animation that has been given to the articulated system. The AI system used within this project, however, makes no such simplifying assumptions, walking is purely an implicit possibility of the fitness function that is applied to the system.

There are drawbacks to each of these approaches:

- Although the method of applying a force animation to the articulated system has been shown to successfully allow an AI to learn to walk in a relatively short time, it also limits the extent to which the body can perform actions — each motion must be predetermined by a corresponding force animation.
- On the other hand, an AI that learns to control the body can potentially produce any motion that the body is capable of through an AI adaptation of inverse body kinematics. However, to control a body to such an extent the AI must be carefully crafted and, given the results of this project, requires a lengthy learning period.

Perhaps a hybrid approach can successfully combine the strengths of each method without suffering from the drawbacks of either. For example, a body could learn to walk using force animations and an AI could learn a vast selection of behaviours that could be applied to the basic technique of walking — such as running, creeping, walking with a swagger or a limp, dancing, jumping, climbing stairs and similar activities.

Finally, the results do not indicate whether a weighted qualitative encoding benefits the AI within this project. The efficiency of a weighted qualitative encoding should be evaluated using less complex AI problems that allow comparison of qualitative, weighted qualitative and quantitative encodings.

¹For a demonstration of AI solutions a video file has been made available for download [27].

Chapter 9

Conclusion

A foundation for AI robotics consisting of three elements — physical simulation, articulated systems and artificial intelligence — was presented. Each of these elements was discussed in turn in context to this project.

Physical simulation was accomplished with the Open Dynamics Engine. An EBNF for simple articulated systems was described and an articulated system possessing a high degree of freedom was implemented within the project using the Open Dynamics Engine to control the physical representation and constraints of the body.

An AI model was devised that makes use of a generalised form of evolution to account for learning mechanisms that are not found in classical interpretations of biological evolution. Within the project the AI model makes use of genetic cross-over operations, mutations and mimicking to represent methods of solution searching that act globally, locally and between these two extremes.

Five AI variants — Quality Sin, Quantity Vector, Quality Vector, Quantity Joint and Quality Joint — were defined. Each variant was subjected to simulation and the resulting data was presented. An analysis of the results indicated that the AI solutions found during simulations displayed rudimentary control over their associated body by attempting to maintain balance and take tentative first steps.

In closing, AI control methods using force animations and implicit learning were compared. It was suggested that additional research should be conducted to create an AI hybrid that could make use of a combination of force animations and implicitly learned behaviours. Such a hybrid might benefit from the ease of learning of the former approach and the dynamic variation in the latter.

Bibliography

- [1] J. S. Albus. A new approach to manipulator control: The cerebellar model articulation controller (CMAC). In *Transactions of the ASME: Journal of Dynamic Systems, Measurement and Control*, pages 220–227, September 1975.
- [2] J. S. Albus. Data storage in the cerebellar model articulation controller (CMAC). In *Transactions of the ASME: Journal of Dynamic Systems, Measurement and Control*, pages 228–233, September 1975.
- [3] P. J. Angeline, G. M. Saunders, and J. B. Pollack. An evolutionary algorithm that constructs recurrent neural networks. In *IEEE Transactions on Neural Networks*, volume 5(1), January 1994.
- [4] H. Anton and C. Rorres. *Elementary Linear Algebra Applications Version*. John Wiley & Sons, Inc., Drexel University, 7th edition, 1994.
- [5] I. Asimov. *I, Robot*. Bantam Books; reprint edition (July 1994), 1950.
- [6] J. M. Baldwin. A new factor in evolution. In *American Naturalist*, volume 30, pages 441–451, 1896.
- [7] D. Baraff. Linear-time dynamics using lagrange multipliers. In *SIGGRAPH '96 Conference Proceedings*, pages 137–146, 1996.
- [8] R. D. Beer. *Intelligence as Adaptive Behavior: An Experiment in Computational Neuroethology*. Academic Press, 1990.
- [9] R. D. Beer, H. J. Chiel, R. D. Quinn, K. S. Espenschied, and P. Larsson. A distributed neural network architecture for hexapod robot locomotion. In *Neural Computation*, volume 4, pages 356–365, 1992.
- [10] R. K. Belew, J. McInerney, and N. N. Schraudolph. Evolving networks: Using the genetic algorithm with connectionist learning. CSE technical report #CS90-174, University of California, San Diego, 1990.
- [11] G. N. Boone and J. K. Hodgins. Slipping and tripping reflexes for bipedal robots. In *Autonomous Robots, Special Issue on Biped Locomotion of Autonomous Robots*, volume 4(3), pages 259–271, 1997.

- [12] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. *Linear Matrix Inequalities in System and Control Theory*, volume 15 of *Studies in Applied Mathematics*. SIAM, Philadelphia, PA, June 1994.
- [13] R. A. Brooks. A robust layered control system for a mobile robot. In *IEEE Journal of Robotics and Automation*, volume RA-2(1), March 1986.
- [14] R. A. Brooks. A robot that walks; emergent behaviors from a carefully evolved network. In *Neural Computation*, volume 1, pages 253–262, 1989.
- [15] R. L. Burden and J. D. Faires. *Numerical Analysis*. Brooks/Cole Publishing Company, 6th edition, 1997.
- [16] J. Conway. Game of life. In *Scientific American*, page 120, April 1970.
- [17] C. Darwin. *The Origin of Species by Means of Natural Selection*. Penguin Classics, London, 1859.
- [18] S. Ehmann. Rigid body simulation tutorial. Online tutorial, accessible at <http://www.cs.unc.edu/~ehmann/RigidTutorial/>, June 1999.
- [19] M. Fagerlund. Evolved behaviour using NEAT. Online results, accessible at <http://www.cambrianlabs.com/mattias/Evolved>, 2002.
- [20] R. Featherstone. A divide-and-conquer articulated-body algorithm for parallel $O(\log(n))$ calculation of rigid-body dynamics. Part 1: Basic algorithm. In *International Journal of Robotics Research*, volume 18(9), pages 867–875, 1999.
- [21] R. Featherstone. A divide-and-conquer articulated-body algorithm for parallel $O(\log(n))$ calculation of rigid-body dynamics. Part 2: Trees, loops and accuracy. In *International Journal of Robotics Research*, volume 18(9), pages 876–892, 1999.
- [22] R. French and A. Messinger. Genes, phenes and the Baldwin effect: Learning and evolution in a simulated population. In R. Brooks and P. Maes, editors, *Artificial Life IV*, Cambridge, Massachusetts, 1994. MIT Press.
- [23] H. Fuchs, Z. M. Kedem, and B. F. Naylor. On visible surface generation by a-priori tree structures. In *SIGGRAPH '80 Conference Proceedings*, volume 14(3), pages 124–133, July 1980.
- [24] H. Gomi and M. Kawato. Neural network control for a closed loop system using feedback-error-learning. In *Neural Networks*, volume 6, pages 933–946, 1993.
- [25] H. Hakl. In-house physics engine. Pre-compiled executable, requires Windows environment and DirectX8 — downloadable at <http://www.cs.sun.ac.za/~henri/physdemo.zip>, December 2002.
- [26] H. Hakl. In-house physics engine. Sources for Delphi6 — downloadable at <http://www.cs.sun.ac.za/~henri/physdemosrc.zip>, December 2002.

- [27] H. Hakl. AI Video. Demonstration video of AI behaviour — downloadable at <http://www.cs.sun.ac.za/~henri/aivideo.avi>, August 2003.
- [28] C. Hecker. Physics, Part 4: The Third Dimension. Published in *Game Developer*, June 1996.
- [29] G. E. Hinton and S. J. Nowlan. How learning can guide evolution. In *Complex Systems*, volume 1, pages 495–502, 1987.
- [30] Honda. *Advanced Step in Innovative Mobility – ASIMO*. Online resource, accessible at <http://world.honda.com/ASIMO/>, May 2003.
- [31] L. P. Kaelbling and A. W. Moore. Reinforcement learning: A survey. In *Journal of Artificial Intelligence Research*, volume 4(Jan-Jun), pages 237–285, 1996.
- [32] S. Kajita and K. Tani. Experimental study of biped dynamic walking. In *IEEE Control Systems Magazine*, volume 16(1), pages 13–19, February 1996.
- [33] M. Kawato, K. Furukawa, and R. Suzuki. A hierarchical neural-network model for control and learning of voluntary movement. In *Biological Cybernetics*, volume 57, pages 169–185, 1987.
- [34] J. Lander. Lone game developer battles physics simulator. Online tutorial, accessible at http://www.gamasutra.com/features/20000215/lander_01.htm, February 2000.
- [35] J. Lander. Trials and tribulations of tribology. Online tutorial, accessible at http://www.gamasutra.com/features/20000510/lander_01.htm, May 2000.
- [36] J. Laszlo, M. van de Panne, and E. Fiume. Limit cycle control and its application to the animation of balancing and walking. In *SIGGRAPH '96 Conference Proceedings*, pages 155–162, 1996.
- [37] M. L. Littman. Simulations combining evolution and learning. In R. K. Belew and M. Mitchell, editors, *Adaptive Individuals in Evolving Populations: Models and algorithms*, Santa Fe Institute Studies in the Sciences of Complexity, Reading, Massachusetts, 1995. Addison Wesley.
- [38] G. Mayley. Guiding or hiding. In P. Husbands and I. Harvey, editors, *Proceedings of the Fourth European Conference on Artificial Life (ECAL97)*, 1997.
- [39] M. McKenna and D. Zeltzer. Dynamic simulation of autonomous legged locomotion. In *SIGGRAPH '90 Conference Proceedings*, volume 24(4), pages 29–38, July 1990.
- [40] C. B. Mclean. *Design, Evaluation and Comparison of Evolution and Reinforcement Learning Models*. Master thesis, Rhodes University, South Africa, April 2001.
- [41] W. T. Miller. Learning dynamic balance of a biped walking robot. In *ICANN 1994*, pages 2771–2777, 1994.

- [42] W. T. Miller. Real-time neural network control of a biped walking robot. In *IEEE Control Systems Magazine*, pages 41–48, February 1994.
- [43] W. T. Miller, R. Sutton, and P. Werbos, editors. *Neural Networks for Control*. MIT Press, Cambridge, Massachusetts, 1990.
- [44] B. Mirtich. Fast and accurate computation of polyhedral mass properties. In *Journal of Graphics Tools*, volume 1(2), 1996.
- [45] B. Mirtich. Hybrid simulation: Combining constraints and impulses. In *Proceedings of First Workshop on Simulation and Interaction in Virtual Environments*, July 1996.
- [46] B. Mirtich. *Impulse-based Dynamic Simulation of Rigid Body Systems*. PhD thesis, University of California, Berkeley, December 1996.
- [47] B. Mirtich and J. Canny. Impulse-based simulation of rigid bodies. In *Proceedings of 1995 Symposium on Interactive 3D Graphics*, April 1995.
- [48] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
- [49] H. Miyamoto, M. Kawato, T. Setoyama, and R. Suzuki. Feedback-error-learning neural network for trajectory control of a robotic manipulator. In *Neural Networks*, volume 1, pages 251–265, 1988.
- [50] M. Moore and J. Wilhelms. Collision detection and response for computer animation. In *SIGGRAPH '88 Conference Proceedings*, volume 22(4), pages 289–298, August 1988.
- [51] NaturalMotion. Active Character Technology. Online home site, accessible at <http://www.naturalmotion.com>, July 2002.
- [52] NaturalMotion. Active Character Technology. Online technology description, accessible at <http://www.naturalmotion.com/technology.htm>, July 2002.
- [53] NaturalMotion. Active Character Technology. Online technology description, a basic walker, accessible at http://www.naturalmotion.com/technology_hiw.htm, July 2002.
- [54] T. Ngo and J. Marks. Spacetime constraints revisited. In *SIGGRAPH '93 Conference Proceedings*, volume 27, pages 343–350, 1993.
- [55] S. Pannu, H. Kazerooni, G. Becker, and A. Packard. μ -Synthesis control for a walking robot. In *IEEE Control Systems Magazine*, volume 16(1), pages 20–25, February 1996.
- [56] M. Raibert and J. K. Hodgins. Animation of dynamic legged locomotion. In *SIGGRAPH '91 Conference Proceedings*, volume 25(4), pages 349–358, July 1991.
- [57] T. S. Ray. Aesthetically evolved virtual pets. In C. C. Maley and E. Boudreau, editors, *Artificial Life 7 Workshop Proceedings*, pages 158–161, 2000.
- [58] T. Reil and P. Husbands. Evolution of central pattern generators for bipedal walking in a real-time physics environment. In *IEEE Transactions on Evolutionary Computation*, 2001.

- [59] T. Reil and C. Massey. Biologically inspired control of physically simulated bipeds. In *Theory in Biosciences*, volume 120, pages 1–13, 2001.
- [60] T. Reil and C. Massey. Facilitating controller evolution in morpho-functional machines - a bipedal case study. To appear in F. Hara, Y. Kakazu, and R. Pfeifer, editors, *Shaping embodied intelligence: The morpho-functional machine perspective*, Berlin, 2001.
- [61] C. Reynolds. Flocks, herds, and schools: A distributed behavioural model. In *SIGGRAPH '87 Conference Proceedings*, volume 21(4), pages 25–34, July 1987.
- [62] G. Ruebsamen. Evolving intelligent behaviors in embodied agents. Online article, accessible at <http://www.erachampion.com/ai/>, 2002.
- [63] T. Sasaki and M. Tokoro. Adaptation towards changing environments: Why darwinian in nature? In *4th European Conference on Artificial Life (ECAL97)*, pages 145–153, 1997.
- [64] T. Sasaki and M. Tokoro. Evolving learnable neural networks under changing environments with various rates of inheritance of acquired characters: Comparison between Darwinian and Lamarckian Evolution. In *Artificial Life*, volume 5(3), pages 203–223, 1999.
- [65] F. Scheepers, R. E. Paren, W. E. Carlson, and S. F. May. Anatomy-based modeling of the human musculature. In *SIGGRAPH '97 Conference Proceedings*, pages 163–172, 1997.
- [66] F. Schiller. *Sämtliche Werke*, volume 5, chapter Über den moralischen Nutzen ästhetischer Sitten. Carl Hanser, Munich, 1958.
- [67] J. Shen. *Human body modeling and deformations*. PhD thesis, École Polytechnique Fédérale de Lausanne, 1996.
- [68] K. Sims. Evolving 3D morphology and behavior by competition. In *Artificial Life*, volume 4, pages 28–39, 1994.
- [69] K. Sims. Evolving virtual creatures. In *SIGGRAPH '94 Conference Proceedings*, volume 28, pages 15–22, July 1994.
- [70] R. Smith. *Intelligent Motion Control with an Artificial Cerebellum*. PhD thesis, University of Auckland, New Zealand, July 1998.
- [71] D. Stewart and J. C. Trinkle. An implicit time-stepping scheme for rigid body dynamics with coulomb friction. In *International Journal of Numerical Methods in Engineering*, volume 39, pages 2673–2691, 1996.
- [72] D. Stewart and J. C. Trinkle. An implicit time-stepping scheme for rigid body dynamics with coulomb friction. In *IEEE International Conference on Robotics and Automation*, San Francisco, CA, April 2000.
- [73] S. Stitt and Y. F. Zheng. Distal learning applied to biped robots. In Y. F. Zheng, editor, *Recent trends in mobile robotics*, pages 333–358. World Scientific, 1993.

- [74] D. Sunday. Distance between lines and segments with their closest point of approach. Online archive, accessible at http://geometryalgorithms.com/Archive/algorithm_0106/, June 2001.
- [75] D. Terzopoulos, X. Tu, and R. Grzesucuk. Autonomous locomotion, perception, behavior, and learning in a simulated physical world. In *Artificial Life*, volume 1(4), pages 327–351, July 1994.
- [76] P. M. Todd and G. F. Miller. Exploring adaptive agency II: Simulating the evolution of associative learning. In J.-A. Meyer and S. W. Wilson, editors, *From animals to animats: Proceedings of the First International Conference on Simulation of Adaptive Behaviour*, pages 306–315, Cambridge, Massachusetts, 1991. MIT Press/Bradford Books.
- [77] R. Turner. LEMAN: A system for constructing and animating layered elastic characters. In R. A. Earnshaw and J. A. Vince, editors, *Computer Graphics: Developments in Virtual Environments*, pages 185–203, England, 1995. Academic Press Ltd.
- [78] P. Urban. Autonomous dynamically simulated creatures for virtual environments. <http://www.cs.ru.ac.za/vrsig/techdocs.html>, 2001.
- [79] J. Ventrella. *Virtual Worlds: Synthetic Universes, Digital Life, and Complexity*. Perseus Books, 1999.
- [80] P. J. Werbos. An overview of neural networks for control. In *IEEE Control Systems Magazine*, January 1991.
- [81] A. Witkin and D. Baraff. Physically based modeling: Principles and practice. Online Siggraph '97 Course notes, 1997. <http://www-2.cs.cmu.edu/~baraff/sigcourse/index.html>.
- [82] L. Zuo, J.-T. Li, and Z.-Q. Wang. Anatomical human musculature modeling for real-time deformation. In *Journal of WSCG*, volume 11(1), February 2003.

Appendix A

In-house physics system

A.1 Physics

Appendix A describes the equations and algorithms used to implement the in-house physics engine. The following subsections detail coordinate systems, orientation, inertia, kinematics- and dynamics equations, integrators, as well as collision detection and response. Throughout it is assumed that bodies are rigid and convex.

A.1.1 Coordinate systems and orientation

A body's coordinate system is defined relative to its center of mass, this coordinate system is referred to as body space. Thus the center of mass represents the origin of body space, relative to which body vertices are defined. The center of mass is chosen as origin to facilitate transformation into world space.

Simulation of physics occurs in a different coordinate system, called world space. A body is transformed from body space to world space as required. Given a point \mathbf{r}_0 on the body in body space its equivalent world space position at a given time, $\mathbf{r}(t)$, can be found using the following transformation:

$$\mathbf{r}(t) = \mathbf{A}(t)\mathbf{r}_0 + \mathbf{r}_{cm}(t)$$

where $\mathbf{A}(t)$ is a rotation matrix representing the body's orientation in world space at time t and $\mathbf{r}_{cm}(t)$ is a translation vector that represents the body's center of mass in world space at time t . The rotation acts through the center of mass of the body, thus the transformation first rotates the body in body space and then translates it into its world space position.

The rotation matrix also represents the orientation of the body. The orientation describes the body's alignment against the three angular degrees of freedom. Many approaches, such as the use of Euler angles (*roll, pitch and yaw*), exist to describe orientation. Stephen Ehmman [18]

describes the use of *quaternions* to represent rotations and orientation in physics simulation, whereas Chris Hecker [28] makes use of *special orthogonal matrices*. Each representation possesses distinct advantages and disadvantages, however, none of these are sufficiently significant to generally prefer one representation over another. The in-house physics simulator adopts the use of special orthogonal matrices.

A special orthogonal matrix is a 3×3 matrix that is not a reflection (meaning it cannot turn a right-handed coordinate system into a left-handed one), furthermore the matrix rows are unit length and $\mathbf{A}\mathbf{A}^T = \mathbf{A}^T\mathbf{A} = \mathbf{I}$, where \mathbf{I} is the identity matrix, and thus $\mathbf{A}^T = \mathbf{A}^{-1}$. Such a matrix is both a rotation matrix and represents an orientation.

As a final note on the orientation matrix: Numerical inaccuracies during simulations may cause the orientation matrix to shear and scale objects as well as rotating them. To ensure that such errors do not adversely affect the simulation the orthogonal constraints on the orientation matrix must be continuously enforced throughout the simulation.

A.1.2 Inertia

Every body that is simulated possesses mass and thus inertia. Inertia is a measure of how a body resists changes in motion. To accommodate the effects of inertia on linear motion is straightforward — as the effects of the resultant force is scaled according to the mass of the body that the force acts upon. Changes in rotary motion, however, are more complex to compute: We make use of an inverse inertia tensor matrix to describe how the shape and mass distribution affects changes in angular motion.

The computation of the inverse inertia tensor in world space is arduous, as every rotation changes the inertia tensor of the body. In body space, however, the inertia tensor is constant, as a body is defined to be rigid and unchanging. The inverse inertia tensor can thus be computed in body space and can be converted to its world space equivalent as necessary using the body orientation in a similarity transform:

$$\mathbf{I}_w^{-1} = \mathbf{A}\mathbf{I}_b^{-1}\mathbf{A}^T$$

where \mathbf{I}_w^{-1} is the world inverse inertia tensor, \mathbf{I}_b^{-1} is the body inverse inertia tensor and \mathbf{A} is the body orientation. Refer to Chris Hecker's article [28] or Section 8.5, Similarity, in [4] for the derivation of similarity transforms.

The inertia tensor matrix is defined as follows:

$$\mathbf{I}_{itm} = \begin{pmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{pmatrix}$$

where

$$I_{xx} = \int_B y^2 + z^2 dV ,$$

$$\begin{aligned}
 I_{yy} &= \int_B z^2 + x^2 dV , \\
 I_{zz} &= \int_B x^2 + y^2 dV , \\
 I_{xy} &= I_{yx} = - \int_B xy dV , \\
 I_{xz} &= I_{zx} = - \int_B xz dV \quad \text{and} \\
 I_{yz} &= I_{zy} = - \int_B yz dV
 \end{aligned}$$

are integrals over the volume of the body. It is easy to analytically solve these integrals for simple shapes such as boxes and ellipsoids, however for arbitrary, oddly shaped bodies these integrals can become arbitrarily complicated and numeric methods need to be applied to solve for the inertia tensor. One such method was presented by Brian Mirtich [44] involving the use of projections and Green's Theorem.

For a box of mass M and dimensions x , y and z the inverse inertia tensor can be computed as follows:

$$\mathbf{I}_{itm}^{-1} = \begin{pmatrix} \frac{3}{M(y^2+z^2)} & 0 & 0 \\ 0 & \frac{3}{M(x^2+z^2)} & 0 \\ 0 & 0 & \frac{3}{M(x^2+y^2)} \end{pmatrix} .$$

The following pseudo code for the computation of the inertia tensor for arbitrary shapes was adapted from Stephen Ehmann's tutorial [18]:

Algorithm 1: Computing the inertia tensor for arbitrary shapes

```

I = [];
hits = 0;
box = BoundingBox(body);
mass = MassOf(body);
for x = box.low.x to box.hi.x do
    for y = box.low.y to box.hi.y do
        for z = box.low.z to box.hi.z do
            if x,y,z within body then
                hits = hits + 1;
                I = I +  $\begin{bmatrix} y \times y + z \times z & -x \times y & -x \times z \\ -x \times y & x \times x + z \times z & -y \times z \\ -x \times z & -y \times z & x \times x + y \times y \end{bmatrix}$ ;
            end
        end
    end
end
I =  $\frac{mass}{hits}$  × I;
    
```

A final note regarding inertia and optimization of a physics simulation: According to *Elementary Linear Algebra* [4], Section 7.2, Diagonalization, since the inertia tensor matrix (and its inverse) are always symmetric they are also orthogonally diagonalizable. Thus it is always possible to align the body space axes in such a manner, that the body inertia tensor and its

inverse are diagonal matrices. Computations with these matrices can therefore be implemented in an optimized manner.

A.1.3 3D kinematics and dynamics

In this subsection we will present various results from the field of rigid body physics. We will briefly indicate how these equations are derived and finally present an algorithm to apply them in a simulation. A more complete derivation of these equations can be found in Chris Hecker's article [28].

It is occasionally necessary to simplify expressions by creating a skew-symmetric matrix from a vector. We use the tilde operator $\tilde{\cdot}$ to that effect:

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \quad \tilde{\mathbf{x}} = \begin{pmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{pmatrix} .$$

The tilde operator can be used to compute cross products using a matrix multiplication:

$$\tilde{\mathbf{x}}\mathbf{y} = \mathbf{x} \times \mathbf{y} .$$

Kinematics of a point on a moving body

As shown earlier the position of a body's point in world space is given as follows:

$$\mathbf{r}(t) = \mathbf{A}(t)\mathbf{r}_0 + \mathbf{r}_{cm}(t) .$$

To make the following derivations more readable we rewrite the equation with time being implicit in the equation.

$$\mathbf{r} = \mathbf{A}\mathbf{r}_0 + \mathbf{r}_{cm} . \quad (2)$$

The velocity of the point is obtained by differentiating the above equation with respect to time, resulting in:

$$\dot{\mathbf{r}} = \dot{\mathbf{A}}\mathbf{r}_0 + \mathbf{A}\dot{\mathbf{r}}_0 + \dot{\mathbf{r}}_{cm} .$$

As \mathbf{r}_0 is a constant vector its derivative is $\dot{\mathbf{r}}_0 = \mathbf{0}$ and thus the middle term falls away. Furthermore the derivative of \mathbf{A} can be shown to be equal to $\dot{\mathbf{A}} = \tilde{\boldsymbol{\omega}}\mathbf{A}$, where $\boldsymbol{\omega}$ is the angular velocity of the body. Finally, $\dot{\mathbf{r}}_{cm}$ simply represents the velocity of the center of mass of the body.

$$\dot{\mathbf{r}} = \tilde{\boldsymbol{\omega}}\mathbf{A}\mathbf{r}_0 + \dot{\mathbf{r}}_{cm} .$$

Furthermore, if we define $\mathbf{A}\mathbf{r}_0 = \mathbf{r}_r$ where \mathbf{r}_r represents the rotated body space vector, then the prior equation simplifies to:

$$\mathbf{v} = \dot{\mathbf{r}} = \boldsymbol{\omega} \times \mathbf{r}_r + \dot{\mathbf{r}}_{cm} . \quad (3)$$

Differentiating equation 3 yields the acceleration of the point.

$$\ddot{\mathbf{r}} = \dot{\boldsymbol{\omega}} \times \mathbf{r}_r + \boldsymbol{\omega} \times \dot{\mathbf{r}}_r + \ddot{\mathbf{r}}_{cm} .$$

And with some manipulation of the equation we can derive:

$$\mathbf{a} = \ddot{\mathbf{r}} = \boldsymbol{\alpha} \times \mathbf{r}_r + \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \mathbf{r}_r) + \ddot{\mathbf{r}}_{cm} \quad (4)$$

where $\boldsymbol{\alpha}$ represents the angular acceleration and the second last term represents the centripetal acceleration of a rotating point and $\ddot{\mathbf{r}}_{cm}$ represents the acceleration of the body's center of mass.

3D Dynamics

The linear momentum of a body is given as

$$\mathbf{P} = M\mathbf{v}_{cm} .$$

In other words, momentum \mathbf{P} is the product of the body's mass M and the velocity of its center of mass \mathbf{v}_{cm} . Differentiating this equation yields the resultant force exerted on the body (where \mathbf{a}_{cm} is the acceleration at the center of mass):

$$\mathbf{F} = \dot{\mathbf{P}} = M\dot{\mathbf{v}}_{cm} = M\mathbf{a}_{cm} .$$

Alternatively we can find the total force exerted on a body by summing the individual forces acting on it.

$$\mathbf{F}_{total} = \sum_i \dot{\mathbf{P}}_i = \sum_i m_i \dot{\mathbf{v}}_i = \sum_i m_i \mathbf{a}_i = M\mathbf{a}_{cm} . \quad (5)$$

This equates the total force on a body as the sum of its momentum derivatives, which is equal to the mass of the whole body M and the acceleration of the body's center of mass \mathbf{a}_{cm} .

In a similar way we will define the angular equivalents of momentum and force, by describing the angular momentum at a point on a body as the cross product between a vector from the body's center of mass to that point, and the momentum at that point:

$$\mathbf{L} = (\mathbf{r} - \mathbf{r}_{cm}) \times \mathbf{P} .$$

The derivative of angular momentum is the angular force, or torque:

$$\boldsymbol{\tau} = \dot{\mathbf{L}} = (\mathbf{r} - \mathbf{r}_{cm}) \times \dot{\mathbf{P}} = (\mathbf{r} - \mathbf{r}_{cm}) \times \mathbf{F}$$

which states that torque at a given point \mathbf{r} is the crossproduct between the vector from the body's center of mass \mathbf{r}_{cm} to that point, and the force vector. The total torque on a body is simply the sum of all individual torques on the body:

$$\boldsymbol{\tau}_{total} = \sum_i (\mathbf{r}_i - \mathbf{r}_{cm}) \times \mathbf{F}_i = \sum_i \mathbf{r}_{(i)} \times \mathbf{F}_i \quad (6)$$

rewriting $\mathbf{r}_i - \mathbf{r}_{cm}$ to $\mathbf{r}_{(i)}$. As a final step we derive a different equation for the total angular momentum on a body. We start by defining the total angular momentum as the sum of all individual angular momentums on the body:

$$\mathbf{L}_{total} = \sum_i \mathbf{r}_{(i)} \times \mathbf{P}_i = \sum_i \mathbf{r}_{(i)} \times m_i \mathbf{v}_{(i)} = \sum_i \mathbf{r}_{(i)} \times m_i \dot{\mathbf{r}}_{(i)} .$$

Taking the above equation as starting point and noting that $\dot{\mathbf{r}} = \boldsymbol{\omega} \times \mathbf{r}$ and that m_i is constant we develop the equation as follows:

$$\begin{aligned} \mathbf{L}_{total} &= \sum_i m_i \mathbf{r}_{(i)} \times (\boldsymbol{\omega} \times \mathbf{r}_{(i)}) \\ &= \sum_i -m_i \mathbf{r}_{(i)} \times (\mathbf{r}_{(i)} \times \boldsymbol{\omega}) . \end{aligned}$$

Rewriting this equation into a matrix form we find that

$$\mathbf{L}_{total} = \sum_i -m_i \tilde{\mathbf{r}}_{(i)} \tilde{\mathbf{r}}_{(i)} \boldsymbol{\omega} = \mathbf{I}_w \boldsymbol{\omega} \quad (7)$$

where \mathbf{I}_w is the inertia tensor matrix.

The kinematics and dynamics results above form the basis for a rigid body simulator: The total force and torque on a body can be easily computed. From the resultant force we obtain the acceleration on the body. The resultant torque can be integrated to obtain the angular momentum on the body. Given the inverse inertia tensor of the body we can compute the angular velocity by manipulating equation 7 above:

$$\mathbf{I}_w^{-1} \mathbf{L}_{total} = \mathbf{I}_w^{-1} \mathbf{I}_w \boldsymbol{\omega} \quad \Rightarrow \quad \boldsymbol{\omega} = \mathbf{I}_w^{-1} \mathbf{L}_{total} . \quad (8)$$

Given the angular velocity and acceleration on the body we can compute the velocity, position and orientation of the body, as demonstrated in Algorithm 2.

A.2 Integrators

A.2.1 Numeric integrators

For an in-depth derivation of numerical solutions to differential equations the reader may refer to a book on numerical methods, such as *Numerical Analysis* [15]. The integration used in Algorithm 2 below makes use of the simplest numeric integrator, known as Euler's integrator. Generally Euler's integrator takes the form:

$$\begin{aligned} w_0 &= \alpha \\ w_{i+1} &= w_i + hf(t_i, w_i) \end{aligned} \quad (9)$$

Algorithm 2: 3D dynamics algorithm

INITIALIZATION

set body constants:

$\bar{\mathbf{I}}^{-1}$	local inverse body tensor
\mathbf{M}	mass of body
e	coefficient of restitution

set initial conditions:

\mathbf{r}_0	initial position
\mathbf{v}_0	initial velocity
\mathbf{A}_0	initial orientation
\mathbf{L}_0	initial momentum

compute initial auxiliary quantities:

$\mathbf{I}_0^{-1} = \mathbf{A}_0 \bar{\mathbf{I}}^{-1} \mathbf{A}_0^T$	world inverse body tensor
$\boldsymbol{\omega}_0 = \mathbf{I}_0^{-1} \mathbf{L}_0$	angular velocity

SIMULATION

compute individual forces and points of application

$\mathbf{F}_{(i)}$	force
$\mathbf{r}_{(i)}$	point of application for $\mathbf{F}_{(i)}$

compute total force and torque

$\mathbf{F}_n = \sum_i \mathbf{F}_{(i)}$	total force
$\boldsymbol{\tau}_n = \sum_i \mathbf{r}_{(i)} \times \mathbf{F}_{(i)}$	total torque

integrate

$\mathbf{r}_{n+1} = \mathbf{r}_n + h\mathbf{v}_n$	new position
$\mathbf{v}_{n+1} = \mathbf{v}_n + h\mathbf{M}^{-1}\mathbf{F}_n$	new velocity
$\mathbf{A}_{n+1} = \mathbf{A}_n + h\boldsymbol{\omega}_n \mathbf{A}_n$	new orientation
$\mathbf{L}_{n+1} = \mathbf{L}_n + h\boldsymbol{\tau}_n$	new momentum
<i>orthonormalize</i> \mathbf{A}_{n+1}	enforce orthogonality constraints

compute auxiliary quantities

$\mathbf{I}_{n+1} = \mathbf{A}_{n+1} \bar{\mathbf{I}}^{-1} \mathbf{A}_{n+1}^T$	current world inverse body tensor
$\boldsymbol{\omega}_{n+1} = \mathbf{I}_{n+1}^{-1} \mathbf{L}_{n+1}$	current angular velocity

where $\frac{dw}{dt} = f(t, w)$. Although Euler's integrator is functional and easy to understand it is slow and inaccurate, requiring very small step sizes (h) to maintain stability.

Observe Taylor's Theorem for function approximation:

$$\begin{aligned} f(x) &= P_n(x) + R_n(x) \\ P_n(x) &= f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \dots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n \\ R_n(x) &= \frac{f^{(n+1)}(E)}{(n+1)!}(x - x_0)^{n+1} \end{aligned}$$

where $x_0 < E < x$. P_n is the Taylor polynomial and R_n is the associated error term. If we write out the Taylor expansion for the first Taylor polynomial we find that

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(E)}{2}(x - x_0)^2$$

finally, substituting $h = x - x_0$ and $f'(t) = g(t, w_i)$, and defining $w_i = w(t_i)$ we obtain:

$$w_{i+1} = w_i + hg(t_i, w_i) + \frac{h^2}{2}g'(E) . \quad (10)$$

The similarity between equations 9 and 10 is immediately obvious and it demonstrates that the Euler integrator possesses an error term of the order $O(h^2)$, furthermore its local truncation error is of order $O(h)$ (refer to Chapter 5.3, Higher-Order Taylor Methods, in Burden and Faires [15]). Several other approaches to numeric integration can be used that offer higher error orders and therefore improved accuracy and stability.

The obvious step to improve the Euler integrator is to add the next term in the Taylor series, thereby reducing the error term to the order $O(h^3)$ (and the local truncation error to $O(h^2)$), this will however require the computation of the derivative for $g(x)$. This derivative computation can be avoided, by substituting a Taylor expansion for the derivative. Although the error is increased slightly, Taylor's error bound is maintained. This technique of expansion and substitution is known as the school of Runge-Kutta methods for differential equations. The first Runge-Kutta expansion of Euler's method, Runge-Kutta Order Two, is also known as the Midpoint Method:

$$w_{i+1} = w_i + h \left[f \left(t_i + \frac{h}{2}, w_i + \frac{h}{2} f(t_i, w_i) \right) \right] + O(h^3) . \quad (11)$$

It is generally accepted that Runge-Kutta Order Four offers the best trade-off between accuracy and computational expense, requiring four function evaluations per iteration and possessing a local truncation error of $O(h^4)$. Runge-Kutta Order Four is usually presented in the

following unnested form:

$$\begin{aligned}
 w_0 &= \alpha \\
 k_{1,i} &= hf(t_i, w_i) \\
 k_{2,i} &= hf\left(t_i + \frac{h}{2}, w_i + \frac{h}{2}k_{1,i}\right) \\
 k_{3,i} &= hf\left(t_i + \frac{h}{2}, w_i + \frac{h}{2}k_{2,i}\right) \\
 k_{4,i} &= hf(t_i + h, w_i + k_{3,i}) \\
 w_{i+1} &= w_i + \frac{1}{6}(k_{1,i} + 2k_{2,i} + 2k_{3,i} + k_{4,i}) .
 \end{aligned}$$

A.2.2 Physical and numeric instability

There exists a distinct difference between physical and numeric instability. Numeric instability is a problem on computers as the floating point numbers used on a computer system cannot accurately represent every number but uses the best possible approximation for numbers. Mathematical formulae should be carefully implemented on a computer to ensure that these rounding errors do not increase unboundedly. Refer to Chapter 1, Mathematical Preliminaries, in Burden and Faires [15] for a detailed explanation.

Physical instability on the other hand refers to inherent instability in simulations. When a simulation produces inaccurate results at a given step size, but is accurate at smaller steps, then physical instability is a possible cause for the poor results. Refer to Jeff Lander's article on integrators [34]. The integration methods discussed in the prior subsection influence the stability in simulations; for example Runge-Kutta Order Four produces accurate results at much larger step sizes than Euler's integrator.

The simple solution is to decrease the step size until the stability requirements are met — unfortunately smaller step sizes cause the simulation to run considerably slower. A better solution is to adapt step sizes as appropriate. The Runge-Kutta-Fehlberg method [15] is an example of an adaptive step size method.

The following should illustrate the idea behind adaptive step sizes: The current integration is performed twice, once with a step size of h , once in two steps using step size $h/2$. If the two answers differ by more than a certain error threshold, then the step size should be decreased. If the answers differ by much less than the error threshold, then the step size can be increased. The Runge-Kutta-Fehlberg method goes as far as to compute an optimal current step size given the current error in the approximation.

A.3 Collisions

The rigid body kinematics and dynamics presented earlier are sufficient to describe the state of a body and how the body will move. For simulation purposes a body needs to be exposed to a variety of continuous forces, such as gravity, drag and friction; occasionally though an instant impulse needs to be applied to a body, usually in the form of a collision. This section will present techniques and equations to detect and respond to collisions within a simulation.

A.3.1 Collision detection

The detection of collisions forms an important part of a physics simulator. A variety of collision types may occur and each requires a different strategy to detect it. The computational expense of collision detection is considerable and correspondingly great effort has been, and still is, invested into finding efficient algorithms to detect collisions.

Definition, technique and requirements

A distinction has to be established between a collision and a penetration. Penetration occurs when two bodies partially or wholly occupy the same space. A collision occurs when two bodies are within a small distance of each other. A body is not allowed to penetrate another body. Collision detection schemes make use of binary elimination to separate cases of collision and cases of penetration:

Given two bodies that are not penetrating or colliding at time $t = 0$ but are penetrating at time $t = \delta$, a test is performed to determine if the bodies are penetrating at time $t = 0.5\delta$. These tests are repeated for $t = 0.25\delta$ or $t = 0.75\delta$, depending on whether penetration did or did not occur at time $t = 0.5\delta$. This binary division scheme is further refined as necessary until a t is found at which no penetration but at least one collision occurs. A collision impulse is then applied to the bodies in question, and the remaining time to $t = \delta$ is simulated.

An algorithm for collision detection must not only determine whether or not a collision has occurred, but must also provide information regarding a collision:

- **Collision bodies** – the actual bodies involved in the collision.
- **Collision normal** – the normal vector along which the collision impulse will be applied.
- **Collision points** – the points of collision between the bodies involved in the collision.
- **Relative velocity** – the relative velocity between the points of collision, if the relative velocity at a potential collision is not negative, then no collision is occurring.

Broad-phase and narrow-phase

Collision detection is usually performed in two phases: Broad-phase detection and narrow-phase detection. The aim of broad-phase detection is the elimination of bodies that cannot possibly collide, for example, if the bounding sphere around a body does not penetrate the bounding sphere of another body, then those two bodies cannot be penetrating each other. Broad-phase detection schemes make use of techniques such as bounding boxes or spheres, or spatial data structures such as octrees.

Narrow-phase detection, on the other hand, establishes collision details – whether penetration, collision or neither has occurred, points of contact, collision normals and relative velocities.

Some implementations use alternative collision detection schemes, such as binary space partition trees (BSP trees) [23]. A BSP tree can be used to efficiently and accurately perform collision detection within urban environments, however, it generally performs poorly in open terrain environments.

Body to plane collisions

Detection of body to plane collisions is relatively simple. A plane is infinite, and since bodies are convex only collisions between body vertices and the plane need to be considered. Body-plane collisions usually occur when the simulation world is defined using infinite planes. Given that \mathbf{r} is a body vertex in world space, \mathbf{p} is a point on the plane in world space, and \mathbf{n} is the plane's unit normal, then

$$d = (\mathbf{r} - \mathbf{p}) \cdot \mathbf{n} . \quad (12)$$

If d is a negative value, then the body has penetrated the plane, if d is within a given error margin, then a collision occurs. The collision normal is simply \mathbf{n} . The collision point is at \mathbf{r} . The relative velocity at point \mathbf{r} , given that the plane is not moving, is presented in equation 3.

Body to body collisions

Collision detection between bodies is more elaborate than collisions between bodies and planes. Two forms of collision may occur, namely point-to-plane collisions and edge-to-edge collisions; all other forms of collisions can be degenerated to these two types of collisions. A point-to-plane collision occurs when a body vertex is very close to another body. Edge-to-edge collisions occur when an edge of a body is very close to the edge of another body.

Given a body A and a body B three cases need to be considered separately to determine whether the bodies are colliding. Vertices of body A may be colliding with body B , vertices of body B may be colliding with body A , and the edges of bodies A and B may be colliding.

To test for point-to-plane collisions between the vertices of body A and the faces of body B the following algorithm can be used:

Algorithm 3: Compute collision state between body A 's vertices and body B 's planes

```

planecount = number of planes in body B;
for each vertex of body A do
  hits = 0;
  for each face of body B do
    if vertex penetrates plane then
      hits = hits + 1;
    else
      distance = distance of vertex to plane;
    end
  end
  if hits = planecount then return penetration;
  if hits = planecount - 1 and distance < error then return collision;
end
return no collision or penetration;

```

The collision normal is the collision plane's normal, the collision point is at the colliding vertex, the relative velocity can be computed as follows:

$$\begin{aligned}
\dot{\mathbf{r}}_A &= \mathbf{v}_{A,cm} + \boldsymbol{\omega}_A \times (\mathbf{r}_A - \mathbf{r}_{A,cm}) \\
\dot{\mathbf{r}}_B &= \mathbf{v}_{B,cm} + \boldsymbol{\omega}_B \times (\mathbf{r}_B - \mathbf{r}_{B,cm}) \\
v_{rel} &= \mathbf{n} \cdot (\dot{\mathbf{r}}_A - \dot{\mathbf{r}}_B)
\end{aligned}$$

where \mathbf{n} is the collision normal.

Testing for collisions between edges of body A and body B is performed in two parts. Firstly it is determined whether a body's edges are penetrating another body, secondly edge-to-edge collisions are tested for.

To determine edge penetrations the following technique can be used: For each edge $(\mathbf{v}_i, \mathbf{v}_j)$ of body A the position of \mathbf{v}_i and \mathbf{v}_j relative to the planes of body B are determined. If the vertices are lying on opposite sides of a plane, then the intersection point can be computed as:

$$\begin{aligned}
d_i &= (\mathbf{v}_i - \mathbf{p}_k) \cdot \mathbf{n}_k \\
d_j &= (\mathbf{v}_j - \mathbf{p}_k) \cdot \mathbf{n}_k \\
t &= \frac{|d_i|}{|d_i| + |d_j|} \\
\mathbf{x} &= \mathbf{v}_i + t(\mathbf{v}_j - \mathbf{v}_i) .
\end{aligned}$$

The t values are stored and sorted. At least two t values must exist for a potential penetration to occur. Given two or more t values the midpoints between them is computed and tested using the point-to-plane test discussed earlier, if a midpoint is found to be penetrating then the bodies are penetrating.

The above method for detecting edge penetrations is due to Moore and Wilhelms [50]. The following mathematical description to determine collisions between edges can be found at [74].

Given pairs of vectors $(\mathbf{v}_i, \mathbf{v}_j)$ and $(\mathbf{u}_i, \mathbf{u}_j)$ each forming an edge, then let

$$\begin{aligned} \mathbf{v} &= \mathbf{v}_j - \mathbf{v}_i \\ \mathbf{u} &= \mathbf{u}_j - \mathbf{u}_i \\ \mathbf{w} &= \mathbf{v}_i - \mathbf{u}_i \\ a &= \mathbf{u} \cdot \mathbf{u} \\ b &= \mathbf{u} \cdot \mathbf{v} \\ c &= \mathbf{v} \cdot \mathbf{v} \\ d &= \mathbf{u} \cdot \mathbf{w} \\ e &= \mathbf{v} \cdot \mathbf{w} \\ D &= ac - b^2 . \end{aligned}$$

If D is zero then the edges are parallel, to compensate if D is sufficiently small¹, choose

$$s = 0 \\ t = \begin{cases} b/d & \text{if } b > c \\ e/c & \text{otherwise} \end{cases} .$$

Otherwise, if D is not a very small number, then

$$\begin{aligned} s &= (be - cd)/D \\ t &= (ae - bd)/D . \end{aligned}$$

If $s \in [0, 1]$ and $t \in [0, 1]$ then a collision occurs at edge points:

$$\begin{aligned} \mathbf{x}_v &= \mathbf{v}_i + s(\mathbf{v}_j - \mathbf{v}_i) \\ \mathbf{x}_u &= \mathbf{u}_i + t(\mathbf{u}_j - \mathbf{u}_i) \end{aligned}$$

and collision normal

$$\mathbf{n} = \mathbf{w} + s\mathbf{v} - t\mathbf{u} .$$

The relative velocity is computed as demonstrated in point-to-plane collisions above.

A.3.2 Collision response

A collision causes an instantaneous impulse to be applied to the two bodies involved in the collision, instantaneously changing their velocity, angular momentum and angular velocity. The

¹For example: $D < 0.00001$ is suitable

impulse acts along the collision normal, but in opposite directions for the two bodies involved. The impulse magnitude is given by:

$$j = \frac{-(1+e)v_{rel}}{\mathbf{n} \cdot \mathbf{n} \left(\frac{1}{M_A} + \frac{1}{M_B} \right) + [(\mathbf{I}_A^{-1}(\mathbf{r}_{AP} \times \mathbf{n})) \times \mathbf{r}_{AP} + (\mathbf{I}_B^{-1}(\mathbf{r}_{BP} \times \mathbf{n})) \times \mathbf{r}_{BP}] \cdot \mathbf{n}} \quad (13)$$

where \mathbf{n} is the collision normal, v_{rel} is the relative velocity of the points at the collision, M_A is the mass of body A , M_B is the mass of body B , \mathbf{I}_A^{-1} is the inverse world tensor of body A , \mathbf{I}_B^{-1} is the inverse world tensor of body B , \mathbf{r}_{AP} is the vector from the center of mass of body A to the point of collision, \mathbf{r}_{BP} is the vector from the center of mass of body B to the point of collision, and e is the coefficient of restitution — a measure of the energy loss during the collision — $e \in [0, 1]$ such that for $e = 1$ the collision is perfectly elastic and for $e = 0$ the bodies lose all energy during the collision and stick together.

The changes in velocity, angular momentum and angular velocity are given as follows:

$$\begin{aligned} \mathbf{v}_{new} &= \mathbf{v}_{old} + j\mathbf{n} \\ \mathbf{L}_{new} &= \mathbf{L}_{old} + (\mathbf{r}_{col} - \mathbf{r}_{cm}) \times j\mathbf{n} \\ \boldsymbol{\omega}_{new} &= \mathbf{I}_w^{-1} \mathbf{L}_{new} . \end{aligned}$$

A.4 Final remarks

A.4.1 Forces

A number of forces exist that may be modelled in a physics simulator, such as gravity and friction. Whereas gravity, according to Newtonian physics, can be modelled with relative ease, friction is more difficult to simulate. Tribology is the study of the interaction of friction and bodies, Jeff Lander [35] wrote a good tutorial on the subject.

Many simulators make use of a dampening force, both for physical accuracy, and to offer a slight stabilization of the simulator. The dampening force on a body acts against the direction of motion and is proportional to the size of the velocity of the body:

$$\mathbf{f}_{damp} = -k\mathbf{v}$$

where $k \in [0, 1]$ and \mathbf{v} is the velocity of the body.

A.4.2 The implementation

The code for the in-house physics simulator has been made available [25, 26]. The implementation is written in Delphi6 and makes use of the DirectX8 API.

Appendix B

Basic articulated model

This section presents the articulated system body description that is used in the project. A single segment definition is presented and explained, followed by a complete listing of the body.

B.1 A segment and child definition

```
# neck
segment
segmentID      1
segmentType    0
segmentDimension 1.0, 0.3, 1.0
segmentPosition 0.0, -0.4, 0.0
segmentChildren
  child
    childID      0
    jointType    0
    jointPosition 0.0, 0.4, 0.5
    jointAxis    1.0, 0.0, 0.0; 0.0, 1.0, 0.0
    jointConstraint -45, 45; -60, 60
```

B.2 Explanation

The hash symbol, #, is used to indicate the start of a comment, the remainder of the line after the hash symbol is ignored. Comments may not be preceded by statements, though they may be preceded by whitespace, thus:

```
# neck
  # ###some comment###
```

are both legal comments, whereas


```
bassegment 4    # defines base segment
```

is not.

A segment definition always begins with the keyword `segment`. This is followed by the ID declaration for that segment, `segmentID 1`. The segment identifier is used to refer to the segment; this is used, for example, during joint definitions.

Following the ID declaration is a type declaration, `segmentType 0`. The project only supports type 0 segments (rectangular prisms), though other simple types such as spheres or cylinders could be added.

Next the segment's dimensions are declared: `segmentDimension 1.0, 0.3, 1.0` — which specifies the length, height and width respectively. The prism is always aligned along the world x-, y- and z-axis, correspondingly the dimensions 1.0, 0.3 and 1.0 indicate the size of the prism along the x-, y- and z-axis respectively.

The last required specification for a segment is its position, `segmentPosition 0.0, -0.4, 0.0`. This is a relative specification of the segment in world space, as the segment position specified is, in fact, the vector from the center of mass of the segment to the location of the joint that attaches some other segment to this segment. The only exception is the segment position specified for the base segment. In this case the segment position indicates the initial position of the base segment in the world space.

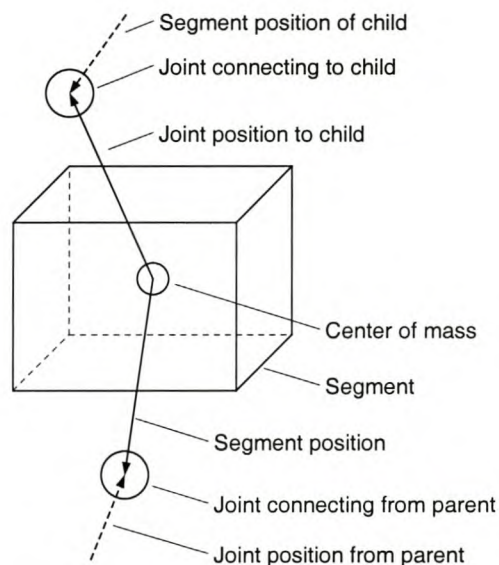


Figure 26: Relative vectors used to describe a segment

Any given segment may also specify one or more child segments, indicated by the keyword `segmentChildren`. Each child definition is subsequently initiated with the keyword `child`.

Each child then specifies an identifier, `childID 0`, which indicates to which target segment the current segment is connected to at this child joint. This is followed by a type definition for the child joint, `jointType 0`. Within this project the joint may be of two types: A ball-and-socket joint (type 0) or a hinge joint (type 1); though other types, such as universal joints, could also be implemented.

After the joint's type has been declared the joint position, `jointPosition 0.0, 0.4, 0.5`, is specified. Similar to the segment position the joint position is a relative reference — it determines the vector from the center of mass of the segment to the joint position.

Next follows the definition of the joint axes — `jointAxis 1.0, 0.0, 0.0; 0.0, 1.0, 0.0`. In the case of a type 0 joint (ball-and-socket) two axes are defined. The second axis represents a principle axis of the joint, around which the segments may swivel. The first axis and an implicit third axis (formed from the cross product of the first and second axes) represent deviation axes, which allow the principle axis to be bent to some extent. In the case of a type 1 joint (hinge) only one axis is defined, which represents the principle axis around which the hinge can swivel.

The degree to which a joint allows swivel and deviation is determined by the last specifier, `jointConstraint -45, 45; -60, 60`. In the case of a ball-and-socket joint two pairs of values are specified. The first pair determines the number of degrees that may be rotated around the first and (implicit) third axes. The second pair determines the number of degrees that may be rotated around the second (or principle) axis. In the case of a hinge joint only one pair is specified, indicating the number of degrees that may be rotated around the hinge axis.

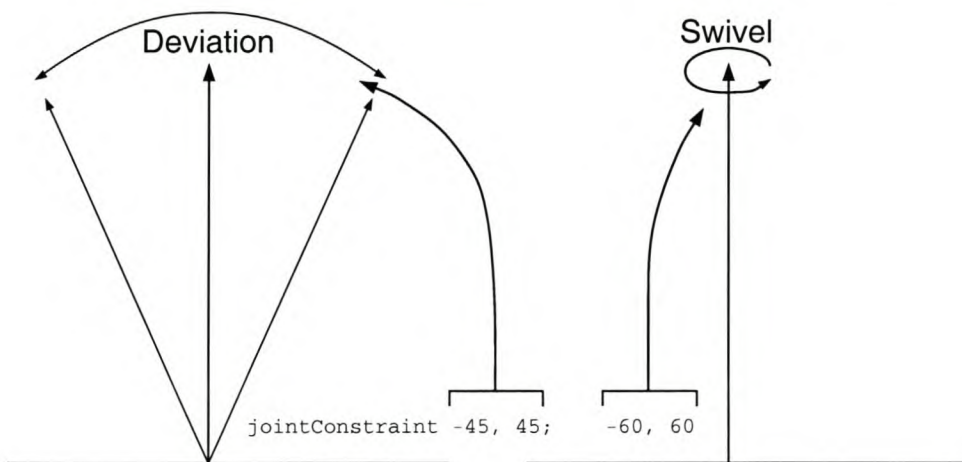


Figure 27: Differences in constraints

B.3 A complete listing of the body definition

The dimensions, layout and constraints of the body were determined by hand:

```
# segmentTypes:
#   0 - cuboid
#   1 - sphere (not implemented)
#   2 - capped cylinder (not implemented)

# jointTypes:
#   0 - ball-and-socket
#   1 - hinge

# joint parameters:
# ball-and-socket: jointPosition, jointAxis (2 vectors)
# hinge: jointPosition, jointAxis (1 vector)

# Note: every ball-and-socket joint exists in conjunction with an AMotor joint
# each AMotor joint is of Euler type, axis 0 and axis 2 get defined. Axis 2
# represents the swivel axis; axis 0 (and the implicit axis 1) allow
# deviation. Axis 0 and 2 must be at right angles to each other.

# EBNF: (note: easier to understand but not fully correct EBNF)
# =====
#
# articulated_system = initialization body
# initialization      = "basesegment" INT
# body                = segment {segment}*
#
# segment = "segment" id type dim pos [children]
# id       = "segmentID" INT
# type     = "segmentType" INT
# dim      = "segmentDimension" VECTOR
# pos      = "segmentPosition" VECTOR
#
# children = "segmentChildren" child {child}*
# child    = "child" cid, jtype, jpos, jaxis, jcon
# cid      = "childID" INT
# jtype    = "jointType" INT
# jpos     = "jointPosition" VECTOR
# jaxis    = "jointAxis" VECTOR [";" VECTOR]
# jcon     = "jointConstraint" INT "," INT [";" INT "," INT]
#
# Note: my loader implicitly expects a newline at the end of each statement
#

basesegment      4

# head
segment
segmentID       0
segmentType     0
segmentDimension 1.0, 1.0, 1.0
```

APPENDIX B. BASIC ARTICULATED MODEL

91

```

segmentPosition  0.0, -1.1, -0.5

# neck
segment
segmentID      1
segmentType    0
segmentDimension 1.0, 0.3, 1.0
segmentPosition 0.0, -0.4, 0.0
segmentChildren
  child
    childID      0
    jointType    0
    jointPosition 0.0, 0.4, 0.5
    jointAxis    1.0, 0.0, 0.0; 0.0, 1.0, 0.0
    jointConstraint -45, 45; -60, 60

# upper chest
segment
segmentID      2
segmentType    0
segmentDimension 2.0, 1.5, 1.2
segmentPosition 0.0, -1.6, -0.1
segmentChildren
  child
    childID      1
    jointType    0
    jointPosition 0.0, 1.6, 0.1
    jointAxis    1.0, 0.0, 0.0; 0.0, 1.0, 0.0
    jointConstraint -15, 15; -10, 10
  child
    childID      12
    jointType    0
    jointPosition 2.1, 1.0, 0.2
    jointAxis    0.0, 1.0, 0.0; 1.0, 0.0, 0.0
    jointConstraint -100, 100; -40, 40
  child
    childID      15
    jointType    0
    jointPosition -2.1, 1.0, 0.2
    jointAxis    0.0, 1.0, 0.0; -1.0, 0.0, 0.0
    jointConstraint -100, 100; -40, 40

# lower chest
segment
segmentID      3
segmentType    0
segmentDimension 1.7, 0.8, 1.1
segmentPosition 0.0, -0.9, -0.1
segmentChildren
  child
    childID      2
    jointType    0
    jointPosition 0.0, 0.9, 0.0
    jointAxis    1.0, 0.0, 0.0; 0.0, 1.0, 0.0

```

```

        jointConstraint  -5, 5;           -5, 5

# waist (base segment)
segment
segmentID      4
segmentType    0
segmentDimension 1.0, 0.6, 0.8
segmentPosition 0.0, 0.0, 0.0
segmentChildren
  child
    childID      3
    jointType     0
    jointPosition 0.0, 0.7, -0.1
    jointAxis     1.0, 0.0, 0.0; 0.0, 1.0, 0.0
    jointConstraint -25, 25;           -15, 15
  child
    childID      5
    jointType     0
    jointPosition 0.0, -0.7, 0.0
    jointAxis     1.0, 0.0, 0.0; 0.0, -1.0, 0.0
    jointConstraint -50, 50;           -20, 20

# hip
segment
segmentID      5
segmentType    0
segmentDimension 2.0, 0.8, 1.0
segmentPosition 0.0, 0.9, -0.1
segmentChildren
  child
    childID      6
    jointType     0
    jointPosition 1.2, -0.9, 0.1
    jointAxis     1.0, 0.0, 0.0; 0.0, -1.0, 0.0
    jointConstraint -35, 35;           -10, 10
  child
    childID      9
    jointType     0
    jointPosition -1.2, -0.9, 0.1
    jointAxis     -1.0, 0.0, 0.0; 0.0, -1.0, 0.0
    jointConstraint -35, 35;           -10, 10

# right upper leg
segment
segmentID      6
segmentType    0
segmentDimension 1.0, 2.5, 1.0
segmentPosition 0.0, 2.6, 0.0
segmentChildren
  child
    childID      7
    jointType     1
    jointPosition 0.1, -2.6, -0.1
    jointAxis     -1.0, 0.0, 0.0

```

```

    jointConstraint    0, 90

# right lower leg
segment
segmentID            7
segmentType          0
segmentDimension     0.7, 2.7, 0.8
segmentPosition      0.0, 2.8, 0.0
segmentChildren
  child
    childID          8
    jointType        0
    jointPosition    0.0, -2.8, 0.0
    jointAxis        1.0, 0.0, 0.0; 0.0, -1.0, 0.0
    jointConstraint  -20, 20; -30, 30

# right foot
segment
segmentID            8
segmentType          0
segmentDimension     0.7, 0.3, 1.5
segmentPosition      -0.15, 0.4, -0.7

# left upper leg
segment
segmentID            9
segmentType          0
segmentDimension     1.0, 2.5, 1.0
segmentPosition      0.0, 2.6, 0.0
segmentChildren
  child
    childID          10
    jointType        1
    jointPosition    -0.1, -2.6, -0.1
    jointAxis        -1.0, 0.0, 0.0
    jointConstraint  0, 90

# left lower leg
segment
segmentID            10
segmentType          0
segmentDimension     0.7, 2.7, 0.8
segmentPosition      0.0, 2.8, 0.0
segmentChildren
  child
    childID          11
    jointType        0
    jointPosition    0.0, -2.8, 0.0
    jointAxis        -1.0, 0.0, 0.0; 0.0, -1.0, 0.0
    jointConstraint  -20, 20; -30, 30

# left foot
segment
segmentID            11

```

APPENDIX B. BASIC ARTICULATED MODEL

94

```

segmentType      0
segmentDimension 0.7, 0.3, 1.5
segmentPosition  0.15, 0.4, -0.7

# right upper arm
segment
segmentID        12
segmentType      0
segmentDimension 2.0, 0.7, 0.7
segmentPosition  -2.1, 0.0, 0.0
segmentChildren
  child
    childID      13
    jointType     1
    jointPosition 2.1, 0.0, 0.0
    jointAxis     0.0, 1.0, 0.0
    jointConstraint 0, 70

# right lower arm
segment
segmentID        13
segmentType      0
segmentDimension 2.0, 0.5, 0.5
segmentPosition  -2.1, 0.0, 0.0
segmentChildren
  child
    childID      14
    jointType     0
    jointPosition 2.1, 0.0, 0.0
    jointAxis     1.0, 0.0, 0.0; 0.0, 1.0, 0.0
    jointConstraint -45, 45; 0, 70

# right hand
segment
segmentID        14
segmentType      0
segmentDimension 0.6, 0.6, 0.6
segmentPosition  -0.7, 0.0, 0.0

# left upper arm
segment
segmentID        15
segmentType      0
segmentDimension 2.0, 0.7, 0.7
segmentPosition  2.1, 0.0, 0.0
segmentChildren
  child
    childID      16
    jointType     1
    jointPosition -2.1, 0.0, 0.0
    jointAxis     0.0, -1.0, 0.0
    jointConstraint 0, 70

# left lower arm

```

APPENDIX B. BASIC ARTICULATED MODEL

95

```
segment
segmentID      16
segmentType    0
segmentDimension 2.0, 0.5, 0.5
segmentPosition 2.1, 0.0, 0.0
segmentChildren
  child
    childID      17
    jointType    0
    jointPosition -2.1, 0.0, 0.0
    jointAxis    -1.0, 0.0, 0.0; 0.0, 1.0, 0.0
    jointConstraint -45, 45; 0, 70

# left hand
segment
segmentID      17
segmentType    0
segmentDimension 0.6, 0.6, 0.6
segmentPosition 0.7, 0.0, 0.0
```


Appendix C

More examples of evolution outside nature

In this Appendix additional examples of evolutionary systems outside of biological evolution are briefly presented. These examples include concepts such as cultures, sport and freedom:

Similar to languages, cultures evolve based on political developments and technological innovation; however, cultural changes may also be triggered by history¹, a prominent example would be Hitler's rise to power of a nationalist Germany in the 1930s. Though socio-political circumstances of the time set the stage for a war, Hitler's power-hungry aspirations that ignited the Second World War were fueled by the desire to recreate the great German empire of the past, the Holy Roman Empire, which lasted nearly a thousand years (936–1806).

Sport is an evolutionary system as time gives rise to new forms of sport and changes the way old forms of sport are played: The rules and regulations of existing sporting disciplines occasionally change. Such changes are usually implemented to make the sport more exciting for the viewer (larger goals in soccer games) or, in contrast, safer for the participants (engine restrictions in Formula One racing). Furthermore, occasionally a new sporting discipline is born such as rugby, for example, which developed from a soccer match gone awry.

The meaning of highly abstract concepts, such as freedom, also change over time. These changes are subject to social and political conditions, as well as philosophical musings. In western civilization freedom, today, emphasizes self-expression and *Selbstverwirklichung*²; however, western society still remembers the days in which freedom was a struggle of liberation from oppression.

¹This is particularly true for eastern cultures that think of time moving cyclically, opposed to the linear understanding of time common to western cultures.

²German, best translated as self-actualisation or self-realisation.

Still, other forms of freedom exist: Friedrich Schiller (1759–1805), one of the foremost dramatists in German literature, stressed all forms of physical and spiritual freedom in his works. Schiller states that, next to materialistic and moral freedom, an additional form of freedom exists: Aesthetic freedom³.

According to Schiller's essays, materialistic freedom can be interpreted as a freedom spawned of a pre-modern frame of mind which demands freedom from oppression. Moral freedom is a freedom born of a rational, modern era which declares that freedom is the right to self-expression, and subsequently chains freedom with laws and regulations. Finally, aesthetic freedom arises from post-modern thinking and does not limit itself through rational restrictions, but relies, instead, on taste and refinement to guide behaviour along, or beyond, rational rules.

³Schiller's thoughts on aesthetic freedom have been collected from his many correspondences, specifically *Über den moralischen Nutzen ästhetischer Sitten* — On the moral use of aesthetic manners — is an important essay derived from the fifth letter to the Prince of Augustenburg (March 3rd, 1793), first published in the third issue of the journal "Die Horen" in 1796. It can be found in Schiller's collected works [66].