Realistic Autonomous Navigation in Dynamic Environments

Alex J. Champandard

October 2002

Master of Science by Research
Institute of Perception Action and Behaviour
Edinburgh University

Abstract

This project aims to develop a navigation system capable of generating humanlike behaviours within complex virtual environments. Fully autonomous animats with sensorial honesty and subjective perception are at the base of the experimentation. The focus on a complete solution emphasizes the capabilities of disparate components and underlines the importance of their interaction during the creation of realistic motion.

A natural interface is designed, providing versatile and powerful control over the movement. Complex desires can be communicated to the system by implicit destinations, allowing route combination and dynamic planning.

At the lowest level, reactive behaviours using evolved perceptrons provide a flexible framework controlled by intuitive high-level parameters, which can be used to craft obstacle avoidance and wall following activities (known as hosts).

Parasites are trained to influence the movement by modifying the information passed to the hosts in the form of obstacle distances. Also using evolutionary neural networks, this parasitic approach can thereby perform behaviour blending smoothly — providing a hook the higher-level behaviours.

The representation of the terrain is assimilated incrementally as the animat wanders around. This topography learning is accelerated and improved by spatial common sense, which factors out trivial movement using the sensory information available. This creates coarse but accurate models of the terrain.

Path planning is performed using a greedy heuristic process to maintain the spanning tree, and reward percolation and accumulation to determine the best path. This exploits the incremental and persistent nature of animat navigation, allowing dynamic planning and route combination.

The system is evaluated as a whole from an external point of view, by several judges who can examine the animats under various trials, showcasing various different conditions and abilities. The animat performs extremely well in specific situations, fooling judges into believing they are human. In other circumstances, motion is rather more reliable than realistic, but is still regarded as an improvement over existing navigation systems.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Alex J. Champandard*)

Acknowledgements

Hopefully the abstract has convinced you to step beyond this page anyway, so I'll be exhaustively grateful to the people that accompanied me along my long perilous journey through science and engineering.

First and foremost, thanks to Gillian Hayes for the focused supervisions and the expert advice during the writeup stages. Apologies for being such a bad influence during nights out, although I hope the fun outweighed the days lost!

Even bigger thanks go to my parents for their support and guidance in my hours of doubt, and to my comfortable uncle David for preventing me from putting McDonald's on my map. The blame should also be placed on Linden Hutchinson and Andy Coates who unselfishly convinced this was the best path for me — and while I'm at it, Patrick and Suresh from Lexicle for making me question my original route in the first place.

I'd like to acknowledge Kurt Miller and Des Browne for keeping me company throughout the yearlong trip, especially during the late-night bursts. Thanks to Kurt for brainstorming the buzzword "pathematics," and to Des for entertaining my game AI fantasies with me.

Big thanks goes to Peter Ottery for the company during the cold winter months and for always pointing out the flaws in my behaviours. Acknowledgements to the rest of the MRobots gang, including Paul Crook for the mutual support during the summer, Jay Bradley for the game AI discussions, Yuval Marlom, Ignasi Cos Aguilera & George Maistros their support, advice and lunch break company. Credit goes to Craig Robertson for the fleeting visits, keeping my trek in place within the grand scheme of things.

I'd like to thank David Lawton especially warmly for putting up with a "crazy Frenchman" sleeping on his floor, Mike Chisholm for the guitar tips and jamming sessions, and Jessica Champandard for completing those great weekend breaks.

Thanks to Tim, Dave and Daniel for the friendship and the many Go games, and to the rest of the "normal" MSc crue for making me feel so lucky and out of place at the same time (no lectures you see!). Cheers to Mat Buckland for his company in the later stages and the exhaustive proof reading, to Rob Saunders for his belief and help, and the #openai developers for the great discussions.

Finally, thanks to the development gurus over on Flipcode for getting me out of dead ends and providing an endless flow of expertise. Thanks to id software for creating such a great AI development platform that is Quake 2, and I'm eternally grateful of your opening of the source code!

Contents

List of Figures

Chapter 1

Introduction

“An ant on the move does more than a dozing ox.” —*Lao Tzu*

1.1 Inspiration

An average field rodent has countless daily problems; it must creep out of its lair early in the evening, fleeing the menacing owl. Most of the night will be spent wandering around, finding interesting locations with food for the young offspring. Then, it must navigate the exhausting journey back to its hole while still avoiding the famished predators.

This is one of many harmless mammals abused by principles of the universe; it’s due to their physical nature that all animals have to deal with such spatial constraints. Movement is the solution, allowing them to handle these obstacles scattered in space. However, beyond just relying on crude motion as a necessity, some animals make it appear almost as an art — effortless yet elegant.

Humans fall into a very similar category; they too are hostages of physics, somewhat limited by the abilities of their bodies. Nevertheless, their movement abilities remain a step ahead of most mammals. A generally superior intellect has allowed them to enhance their locomotion with simple materials *(e.g., pairs of shoes or walking sticks)*. People have also been able to make good use of their environment for further assistance *(e.g., creating roads and posting signs)*. A popular example is the tale of `Hansel` and `Gretel`, who used stones to mark their route into the dark woods, allowing them to safely plot their course back home.

Already, a distinction is appearing between straightforward abilities *(moving body limbs, preventing collision)* and more intricate skills *(remembering landmarks, finding routes back)*. Most types of mammals are capable of simple tasks like avoiding obstacles, explaining why it is often taken for granted. However, complex navigational behaviours can be a distinguishing factor between animals and humans.

In fact, besides the daily quandaries, people have found ways to evaluate and improve their abilities by devising challenges in space. Events are organised regularly to test movement aptitude under different settings; at the simpler level, Olympics test aerobic or anaerobic capabilities for speed and endurance, while more elaborate orienteering events reward navigational skills. Mock versions of the mythological Cretan labyrinth have also even been built to tease and entertain guests in chateaus from the Renaissance. Multi-player settings such as basketball and football present further dynamic difficulties, as well as a necessity for coordination of movement. Other team efforts such as paint balling are similar, though stealth and strategy tend to be rewarded more. Technology is also used to enhance the challenge, as in Formula 1 races where competitors have to steer powerful cars at over 300 km/h.

Even this brief overview reveals the wide variety of skills required in navigation, ranging from power to precision, including reflexes or planning, not forgetting dexterity and synchronisation. This partly explains why equipment is designed to assist and enhance movement in every day life *(e.g., scooters or elevators)*. For example in automobiles, the pilots' personal freedom is restricted, but they can steer the cars from a higher-level point of view, while the underlying mechanics of the car are relied upon for the physical aspects of motion. The performance of the system `(car,driver)` is improved over humans alone. In many such conditions, humans have become small components in moving systems — but most importantly their engineers.

By automating its parts, a whole navigation system can be taken to the next level since the limitations of the human body can be designed around. Some of the simpler motion can be achieved with varying degrees of mechanics. However, the parts requiring the full aptitude of the human brain can be problematic to automate, as shown by the slow progress in the field. Such jobs are commonly taken care of by human operators instead, who are for example responsible for controlling speed in trains or checking for eventual mishaps when passengers board. Removing people from the loop is the next logical step for efficiency in navigation systems, which would make them truly autonomous.

An increasing emphasis is placed on virtual simulations for this reason; the entire spectrum of navigational behaviours can potentially be reproduced. In combining clever supporting design and appropriate assumptions, the problematic parts of human navigation can be modelled and tested with more freedom. Additionally, a fully competent entity can be produced, which presents a different panoply of challenges.

This project intends to develop an entire system capable of reproducing navigational behaviours within such a virtual environment for an autonomous animat. During the research, novel algorithms and solutions will be developed as appropriate to provide realistic behaviours efficiently.

1.2 Foundations

Such a fascinating omnipresent skill in nature and everyday life, as may be expected, has important impacts in science, which aims to understand such phenomena and manifestations. For researchers, this proves both challenging and insightful, explaining why so much work has been done on the subject.

Navigation is prominent in a surprisingly large number of branches in science. Some approaches try to understand the process behind movement; this involves reverse-engineering by observational study. Other methods are rather more practical, attempting to engineer behaviours from scratch, but all attempt to further understanding in navigation.

- Cognition — Navigation is a relatively trivial form of planning, yet it has all the characteristics of higher-level plans (obstacles, dead-ends, traps, dangers), although in a physical rather than abstract form. Thankfully, this provides a practical technique for studying reasoning; by placing creatures in situations that require observable movement, their ability seems amazingly linked to the level of intelligence. While one generally associates mammals with relatively simple motion, it is surprisingly efficient and often leads to pleasant revelations.

- Ethology — As the study of animal behaviour, ethology aims to understand how navigation is possible. Observation at the individual level can help determine why a bee always finds its way back to the flowers. At a collective level, we can explain how and why zebra organise into herds, how flocking birds manage to remain coordinated and coherent, or even how ant colonies can seemingly communicate the location of food to each other.

- Neurobiology — Taking a much lower-level approach to navigation, it is studied by the reaction of neurons in the brains of mammals when they are placed in stimulating scenarios. By extensive testing, scientists can attempt to determine what the animal is taking into account, what parts of the environment are remembered, and how the nervous system triggers motor actions.

  By studying rats, scientists can get insights into how they can find their way through a maze to the cheese, or how the knowledge of rats is used to orchestrate the navigation according to their emotions and drives.

- Computer Science — In a very generic fashion, many problems can be considered as walking through $n$space to find the solution to a problem. These are known as optimal path problems (mostly focusing on shortest options), and can be used in a surprisingly large variety of problems. Whence comes the analogy with navigation, which deals with paths in 3-space.

- Artificial Life — A.L. simulations have attempted to recreate group movement from the ground up (considered as a manifestation of life). Taking into account a limited number of neighbouring companions, simple locomotive rules are defined in each entity to obtain the resulting emergent behaviour. Entire flocks or herds can be crafted and indirectly controlled in this fashion.

- Artificial Intelligence — A.I. has tackled navigation from a very high-level point of view, applied in generic situations in grid worlds. The solutions are devised mostly as search algorithms, which are ideally suited to finding optimal results in a computer-friendly fashion. This can apply to planning single trajectories relatively well, though this neglects many other equally important aspects of animat navigation that are so often ignored in literature.

- Robotics — Over the past two decades, the field of *Robotics* has seen a tremendous amount of focus on navigation. This remains one of its most active areas of research, and arguably the most fascinating. Aside from providing a plethora of technical challenges for computer vision, machine learning, and robot motor control, navigation can transform abstract algorithms into palpable locomotive behaviours; it provides a face for *Artificial Intelligence*.

  Applying the theoretical knowledge gained from computer science and artificial intelligence into real-life is an extremely difficult process. However, the quality of the robots engineered is consistently increasing as more effort, computational power and money is poured into them.

The combination of the fields described above contributes to common navigational knowledge; observation, imitation and modelling by trial-and-error offer a surprisingly broad understanding of navigation. All this experience can be drawn upon to engineer solutions appropriately tailored to the problems at hand.

1.3 Applications

Scientific research is often implicitly driven by the applicability of the technology. In the case of navigation, much can be achieved in by understanding its theory. The potential uses range from the real world to virtual environments; literally any domain where autonomous movement is involved stands to benefit.

1.3.1 Prediction

One of the major benefits of comprehending motion lies in the ability to anticipate it. This allows people to be able to control, shape and influence it how they see fit.

- Hunting — In prehistoric days, Neanderthals successfully hunted mammoth despite them being over 100 times heavier; common techniques involved inciting the animal towards camouflaged traps in the ground or steep cliffs, which they knew would give them the advantage over the beasts.

- Preservation — In modern times, the knowledge can be used to prevent wildlife from crossing dangerous roads, or designing defences that protect species from others (humans included), like in zoos.

- Traffic — The flow of cars and trucks can be analysed on two levels: on a hourly scale for road congestion, or on a real-time basis for physically accurate movement of specific vehicles. This can be used to assist the driver and reduce travel times.

Many other similar domains have also benefited from the understanding of navigation practice.

Type	**Application**	**Example**
virtual navigation	simulated worlds	artificial wild-life
	computer games	synthetic characters
	military training	soldier agents
	animation	virtual people & crowds
robot movement	industry	self-guided vacuum cleaners
	transport	autonomous busses
	toy business	mechanical dogs

Table 1.1: Summary of applications for artificial navigation systems.

1.3.2 Synthesis

Automated movement in the real world has become extremely important for our civilisation, with progress becoming dependent on our ability to travel the globe and to physically move things around fast enough. Good understanding of navigation allows engineers to artificially create systems, synthesising behaviours that can transform inanimate objects into navigating artificial creatures.

Firstly, real-robots are popular applications, for entertainment as well as industrial purposes.

- Toys — Increasingly, toys and similar small gadgets are becoming self powered. Making them autonomous increases the fun they can provide, but relies on their ability to move around. Toys like Sony's robotic pet dog `Aibo`, or fully reactive bugs each have built in navigation skills.
- Industry — Manufacturing industry can benefit from autonomous workers to shunt goods around depots, or self-controlled cleaning bots that keep the factory floor tidy.
- Transport — Mass-transportation in the form of optically guided busses to mostly autonomous trains increase reliability and reduce operating costs. Personalised carriers such as the Segway are also predominantly self-sustained thanks to their balancing system, though they require human pilots.

Understandably, synthesis of navigation is crucial not only for mobile robots but also for virtual agents, which forms the second major category where navigation plays a crucial role.

- Films — Entire herds of stampeding buffalos, like in `The Lion King`, can be generated very quickly by simulating the movement of each mammal. Software is available to automate this task.
- Games — Computer generated forces in `Hidden & Dangerous` for example need convincing navigational abilities to be able to provide a worthy challenge for the human player.
- Military — Virtual soldiers can provide training for troops, mostly from a strategically *(e.g., synchronisation or planning)* and psychological point of view *(e.g., emotional and anticipatory)* point of view.

The main aim of this thesis is to design a system that is capable of fulfilling most of the demands of such applications, summarised in *Table 1.1*. Before venturing further into technical details, a rigorous specification of the problem should therefore be established.

1.4 Definitions

1.4.1 Animat

The project deals with *animats*; the term coined by Wilson (1987) has become an overloaded buzzword even in academia. Etymologically, it appears the word stems from the combination of *"artificial animal"*. This primarily implies that the creature is a physical entity, subject to the constraints of its environment. We intend both real robots and virtual creatures to fall under this category, since they are autonomous entities whether simulated or not.

While people assume a certain level of life-like properties from animats, we prefer to dissociate the term from notions of realism and intelligence. These are simply properties that should be striven towards, although an animat remains a physical entity whether capable of flexible actions or not!

1.4.2 Navigation

Specifically within animat research, this dissertation undertakes navigation: a topic intuitively associated with Spanish explorers of the 16th century. Such initial thoughts need emancipating.

Definition Autonomous navigation is the ability of a system to purposefully steer its course through a physical medium, in an independent, self-controlled fashion.

The only ambiguous term in this definition is certainly one of the most important: *"purposefully"*. Although some philosophers will no doubt debate whether animals (which are also included in this scope) are capable of intention, the word mostly implies a proactive action; falling leaves do not navigate through the air, just as a driverless car rolling down hill does not either. As a lesser connotation, navigation aims to satisfy requirements in space; in practice, it usually involves reaching a target, specified with varying degrees of precision.

By loosely describing navigation as the art of "solving problems in space", the standard definition — consisting of just reaching a target — is somewhat broken away from in a more generic way. Looking at the problem in this new light has many advantages when interacting with the system, and these properties will be exploited by our design (c.f. Chapter 3).

1.5 Challenges

So why is artificial navigation such an interesting problem? Whether applied to real robots or synthetic creatures, there are numerous common issues to take into account.

The environments in which animats operate are extremely diverse, ranging from scouting the surface of Mars to cleaning a warehouse floor, including office environments and radioactive power stations, or even military training environments. Such surroundings host a multitude of hazardous conditions: dynamic obstacles, temporary weather conditions, variations in lighting, and many other unpredictable events. Considering this ubiquitous presence of uncertainty, surrounding phenomena become hard to quantify.

Together with this, animats do not possess of omniscience. Rather, their perception is limited to measly exteroceptive sensors (this is especially problematic for robots). Despite constant improvements in portable sensing devices, or increase in computation power to acquire virtual information, the data available for processing is often both inaccurate and incomplete. This condition is mostly attributed to the subjectivity of the perception.

This implies animats are never totally aware of their state; at best they are able to make only partial observations. This predicament is known as *perceptual aliasing*;[1] the only way around it is to design the underlying AI accordingly — for example using probabilistic reasoning. This inability to recognise their state is a significant problem for animat navigation.

The actuators are equally troublesome (though synthetic creatures have an advantage here). Indeed, due to unforeseen conditions, there may be a disparity between the request and the actual physical action. This causes a divergence between the position in theory and in practice, which can be minimised by the use of proprioceptive sensors to monitor the internal hardware. However, these too are victims of the treacherous noisy conditions; Columbus' idea of *Dead Reckoning*[2] remains utopianism in robotics. Even in virtual environments, there is a surprising precision loss in the rotations required to enforce embodiment.

Given such tough settings, there is little doubt that autonomous navigation is an impressive challenge. Aside from the uncertainty problems introduced, flexibility is ideally required from the solution in order for it to handle novel environments. Together with this, dynamic properties of the surroundings ideally require the animat to continuously update its beliefs, which would otherwise swiftly become out of date. Consequently, a static plan simply won't achieve the goal reliably, regardless of the accuracy of the information at decision

[1] This is due to the one to many mapping from observations to states.

[2] This process attempts to determine the global position by continuously monitoring the relative movement.

time. If possible, the navigation thereby ideally needs to monitor the sensory input stream continuously, and dynamically base its decisions upon that. Moreover, due to the physical consequences of movement such as damage or injury, the animat's response needs to be prompt; the solution is constrained as a real-time system. These utopian solutions become targets as the system is implemented.

1.6 Technology

From the developer's point of view, the task at hand is to design a software solution capable of performing as closely as possible to the ideal system described above, while still dealing with requirements established. Such a model is known as an architecture, which corresponds to a blue-print of the navigation system, showing how the information flows and is taken into account to make the decision of movements. Building an architecture for a specific problem implies anticipating contingencies and noticing regular patterns in the navigational task; architectural short-cuts can then be devised to exploit them. Thus, the more limited the application, the quicker the design and the better the robot will generally perform.

The framework crafted also needs to assist initial stages in the development, reduce maintenance overheads and allow straightforward debugging. Extensibility is also a key factor to integrate into the design, making the system more future-proof. This is achieved mainly by making extensive use of customisable behaviours via scripts and configuration files, which can each be tailored to fit.

Conceptually speaking, the knowledge used for deciding how to navigate is potentially integrated into the system via fixed expert rules, engineered at development time. Alternatively, a learning approach can be set up to learn equivalent behaviours automatically. The latter choice was made, allowing developers to dissociate themselves from the low-level details, allowing them to concentrate on specifying the final outcome of the learning process. In practice, the agent can assimilate procedural and declarative knowledge automatically.

Greater insight into this development process will be provided along with the literature review of Chapter 2.

1.7 Overview

There has been a plethora of work done towards artificial navigation; understandably so, it's a vast topic. Yet still, there are many holes to be filled, gaps to be bridged, dead ends to be explored, and miracle solutions to be discovered. We intend a relatively novel approach to the problem, thereby partly alleviating these knowledge lacunae, but also shedding light on some problematic issues uncovered.

Our work intends to nestle itself amidst the artificial intelligence approaches, while drawing from concepts borrowed from other areas of science. Many research projects tend to be off-balanced towards the preferred approach of the author (showing prejudice against any other approaches along the way), we attempt to keep an open-mind about the many options available for inclusion. Each of the foundations will be drawn upon as appropriate to engineer solutions matching our requirements.

The combination of these three following traits we believe make it unique:

1) Full System — An entire universal autonomous navigational framework is established, rather than developing inconsistent separate components under the delusion that a working system is simply the sum of its parts. This corresponds to Brooks' apprehension towards the design of synthetic creatures as independent components:

"No one talks about representing the full gamut of human intelligence any more. Instead we see a retreat into specialized subproblems [...]. All the work in these subareas is benchmarked against the sorts of tasks humans do within those areas. Amongst the dreamers still in the field of AI [...], there is a feeling that one day all these pieces will all fall into place and we will see 'truly' intelligent systems emerge." — Brooks (1991)

In a similar fashion to Brooks, we build up a system incrementally from the ground up, though this is not done using the method suggested; a custom development process is attempted.[3] This allows us to

[3] One that does not rely on inhibition and subsumption points.

create a working system that does not rely on any other assumptions, experimentally validating the model.

The astute reader may point out a small inconsistency in our policy; the navigation is ironically developed as a separate system. Although instead of looking at it as an isolated module, the intuitiveness of its interface allows it to behave more like an assistant, or a navigational companion who can be asked for favours. This makes the engineering of the animat more like a multi-agent system, where one takes care of the motion.

2) Complex Virtual Environments — Accurate simulated worlds provide a base for the development, experimentation and evaluation of the navigation system. Robots are used to operating in complex environments, but they tend to only perform well in specific situations *(no stairs, sensor friendly walls, no menacing chair legs)*. By tackling the problem from a virtual perspective, experimentation has more freedom; environments are limited only by imagination, time constraints for simulations practically vanish, costs and risks to equipment diminish drastically. Nevertheless, the resulting technology can be applied directly as virtual navigation, and indirectly as artificial navigation in real-life.[4]

3) Realism — The navigation and overall motion-related behaviours are intended to be as human-like as possible — from an external point of view. Accurate underlying mechanisms are simply interesting to study and observe, but do not necessarily provide the most realistic motion.

This requires a **top-down** approach to design, which differs significantly from the bottom up approaches (some based on ethology, neurobiology, or artificial life). Practically, this implies we specify the kinds of behaviours required first, understanding them and conceptually reverse engineering them. Only then can they be synthesized, in an incremental fashion. So in essence, the design is top-down and the development is bottom-up.

Another significant part of this realism, we believe is due to the autonomy of the system. Much effort was invested in actively enforcing the autonomy of the animats. Few virtual navigations are capable of coping with this restriction, let alone enforcing it.

Striking the balance between these requirements in a navigation system remains an extremely interesting and challenging task.

1.8 Motivation

Aside from the applications, there are three main reasons for developing an artificial navigation system within the context of virtual environments.

1.8.1 Genuine Settings

Initial attempts at robot navigation altered the environment in a sensor-friendly fashion (by painting white lines on the floor, or drawing small visual landmarks on the walls). The design would then be fine-tuned to take into account the possible assumptions incurred, extracting and exploiting the data implanted into the surroundings. Aside from being inappropriate for most environments, the process was expensive and time consuming. Similar trends can be spotted in current computer game agents, whose implementation takes advantage of the underlying terrain representation — information not accessible visually.

Today's approaches in robotics are more flexible; using more reliable sensors, they no longer rely on indirect human intervention. In synthetic creatures, competent expert systems are increasingly used to devise fully autonomous control systems. Though generally, one would agree that a certain degree of competence and realism has been lost in the process.

[4] Again, in the words of Brooks *"Simulators are doomed to succeed."*

1.8.2 Capable System

A surprising proportion of papers looking into high-level intelligence for animats *(e.g., planning, multi-agent systems)* seem to rely on a miracle autonomous navigation system. Yet it is nowhere to be found; the most competent systems in virtual navigation are either extremely simple planners, non-autonomous, or low-level behaviours with no practical ways for controlling them.

The challenge is thereby to provide a system elaborate enough to handle the needs of such higher-level components, while still respecting the constraints imposed by the environment as well as the simulation itself. Once the navigation system has been devised, very little additional work is required to build a fully functioning animat. Indeed, it can be relied upon like a very smart horse that understands material concepts!

1.8.3 Simple Usage

In practice, implementing navigation seems a trivial task when done in a straightforward fashion, but more elaborate behaviours require time and effort to match. The system should attempt to minimise the amount of time taken to obtain the desired results, mainly by abstracting tasks as much as possible.

However, despite animats being able to learn the basic concepts, navigation systems still rely on being designed by researchers — more or less. This is usually done in a modular fashion, in order to accommodate various different algorithms efficiently, thereby also allowing incremental development.

1.9 Aims and Assumptions

1.9.1 Primary

The project intends to develop a navigation system for an autonomous animat. While fundamentally this seems like an engineering task, research comes into the equation to push current ideas or technology further and develop novel solutions suited to the problem at hand. Additionally, with regards to the settings, questions of achievability will need to be answered.

Design an interface

Starting from a high level, the services provided by the navigation system must be specified. This involves drafting an intuitive yet powerful interface. Ambiguity and inconsistency both need to be avoided as much as possible, and research will have to determine how this can best be done.

Develop a complete navigation system

The underlying functionality of the system will then need to be implemented to provide the behaviours as specified. These will need to work reliably within the complex environments the animat is expected to survive (e.g., indoor and outdoor terrains, crowded and deserted environments).

Since there has been a modest amount of work on entire systems within our context, the project will encounter many research issues that will need to be tackled. Notably, weaving together independent components together, and handling human-like navigation in autonomous conditions will require further investigation.

Produce realistic behaviours

This is a relatively complex problem, having taken nature a few billion years to fine-tune. The ability of a creature to display such interesting movement behaviours is generally a sign of competence, too easily mistaken for intelligence. Although conversely, a lack of navigation ability generally leads to the dismissal and humiliation of the animat in question.

This project aims to develop realistic navigation that does not rely on rigging environments, cheating or objective information. Rather, a fully embodied, honest creature is defined; authentic animats are at the base of the experiment. Existing algorithms and solutions will no doubt have to be extended and improved to handle the problem.

1.9.2 Secondary

Flexible architecture

Designing an intelligent agent is not a trivial task. This should be assisted as much as possible by the methodology, tools, and support software. A well-established framework will simplify development, speed up debugging and encourage extensions and improvements.

Real-time solution

Since the solution is expected to perform efficiently on lower-end consumer hardware, in non-exclusive fashion, its implementation will need to be as lean as possible. There should be minimal overhead in both computational power and memory usage.

1.9.3 Assumptions

The extremely low-level details of locomotion, such as moving individual limbs and dealing with accurately simulated characters, are abstracted out. This stems from the fact that all our development platforms do this by animation, and this is not a requirement. Furthermore, this capability is often conceptually dissociated from the essence of motion since the final result is most important, and not the order in which the limbs were moved.

From a perceptual point of view, objects are represented as symbols, and do not need extracting from the raw sensory data. As such, the animats do not have any form of visual perception per se, simply an interface which fakes this higher-level of processing that humans perform somewhat automatically.

1.10 Summary

Since the thesis focuses on a complete system and tackle issues that arise in the process, a certain amount of detail will have to be glossed over in each of the sections. Notably, the depth of the background reviews of specific modules and their evaluation is neglected in favour of completeness.

The rest of the dissertation is divided as follows:

- Related Work — The next chapter contains an overview of existing solutions, including insights into robot based, military, academic and computer game applications. An objective analysis of these systems looks into their advantages and deficiencies, allowing the identification of key areas for potential improvements.
- Design — Afterwards, in *Chapter 3*, the design of the proposed solution will be discussed, allowing us to define the skeleton of the system. This specifies the modules utilised as well as their interaction.
- Framework — The *Chapter 4* describes the common techniques used throughout this project, and documents the relevant implementation details of the software used.
- Modules — Subsequent chapters *(5, 6, 7, 8)* will discuss specific components of the navigation system in more detail; each provides information about requirements, specification, design, motivation, and implementation for the respective modules.
- Evaluation — The system as a whole is evaluated on human terms in *Chapter 9*. A discussion of the results and corresponding development process follows, with each component conceptually analysed where applicable.
- Conclusion — Finally, *Chapter 10* comments on potential applications of the technology. We reflect on the thesis of this Masters, hopefully providing pertinent insightful comments. The road ahead is considered as planned improvements for the future are discussed.

This should provide a thorough overview, although sadly not exhaustive, of the system created during this yearlong project.

Chapter 2

Related Work

"He who loves practice without theory is like the sailor who boards ship without a rudder and compass and never knows where he may cast."
—Leonardo da Vinci

This chapter looks at existing solutions capable of performing navigation, from a variety of backgrounds. Many fields of applied science have roots in navigation, each of which are relevant for this dissertation. Admittedly, some concepts may need adapting, but any agent related publication may prove insightful. Therefore, the following pages attempt to provide a representative cross-section of previous efforts, ranging from *robotics* to the *computer games* industry, including *military development* and other *academic research*. Each will be briefly described and evaluated against some general criteria relevant to the problem at hand.

Designing a fully-fledged autonomous navigation system is a surprisingly large task, as the size of this dissertation and many other project reports testify. To that end, designers often find ways of splitting it into smaller manageable chunks — and understandably so; modularity is one of the hottest topics of this past decade, and applies equally well to *Artificial Intelligence*.[1] In part, this explains why a significant proportion of navigational research has focused on separate components. Such specific algorithms relevant to the architecture are reviewed in the appropriate chapters; meanwhile this literature review takes a broader look at complete packages, concentrating on how they are woven together as a whole.

2.1 Robotics

Navigation was one of the earliest problems tackled with robots, and has arguably been one of the most popular ever since. Over the decades of research in the area, trends have naturally emerged. These different styles will be first summarised, as they can provide a myriad of pertinent insights even before the review commences. Specific implementations will then be analysed in more detail.

2.1.1 Overview

There are sufficient fingers in one hand to count the number of distinctive navigational architecture types. Historically, there are in fact three (reactive, deliberative, hybrid). However, there has been much inter-breeding over the decades that boundaries are no longer distinctly marked out. Some subcategories have also become increasingly popular and broader than others, but each still has its own advantages and pitfalls. Selecting the most appropriate blend is effectively a matter of omniscient experience, which can only really be done successfully with hindsight — if at all. Even then, there are compromises to be made.

[1] Despite Brooks' apprehensions of the design of individual "intelligent" parts.

Reactive Architectures

Strictly speaking, fully reactive navigation has no sense of state whatsoever, and uses a direct mapping from sensors to actions. This implies that there is no possible ambiguity; one sensory input corresponds to one action vector. This can be done by explicitly storing and matching pairs (`sensation/action`), or with a more implicit approach like neural networks. However, in practice, a negligible number of persistent variables is still tolerated for modern pseudo-reactive models (with a simple sense of state). Such schemes cope with continuous actions relatively well.

Behaviour Based Models

Behaviour based architectures are composed of parallel concurrent behaviours (Arkin, 1998). These can be purely reactive (as originally intended, a subset of the previous category) but also based on internal representations (in subsequent implementations) — whatever helps the designer perform mappings from condition to action. However, since the system is distributed, there is no good place to put a central world model; the exchange of information between behaviours is generally done implicitly via the environment.[2] Increasingly, however, internal communication is used to facilitate this task. Typically, a priority-based model of arbitration is applied to select the dominant behaviour. Such models are constructed incrementally by extensive testing and debugging, which implies a bottom-up design that emergently creates a whole.

Deliberative Architectures

Deliberative navigation typically generates a sequence of actions, based on a centralised internal representation of the world. When the aim is to reach a specific goal, the current solution often requires to be refined over time to avoid dynamic problems; this corresponds to the conventional `sense-plan-act` paradigm. Internal representations need to be tailored in order to facilitate the deliberative process *(e.g., cells, grids or nodes)*, along with discrete actions. This allows a search algorithm to be performed, which — thanks to reasonable anticipatory capabilities — proves capable of exploring consequences of actions in the future in order to find the best outcome.

Hybrid Systems

Hybrid architectures tend to separate systems into conceptually different parts, each with different tasks, communicating together. These layers can be considered as independent black boxes, which simplifies development. This category is also known as heterogeneous, since the internal computation methodology can be extremely varied — as opposed to homogenous deliberative or reactive systems.

In some cases, the components may be arranged in a layered fashion. Depending on which way one looks at the problem, this layering can be either horizontal or vertical. *Horizontal* arrangements are typically based on the flow of information from sensors to effectors (passing once or twice via each layer, increasing or decreasing level of abstraction), while *vertical* solutions tend to have layers of increasing behavioural complexity; for example lower levels of the hierarchy can involve straightforward obstacle avoidance, while higher-level modules deal with path-planning.

2.1.2 Solutions

Over the years, numerous award winning systems and architectural masterpieces have been designed. Here is a representative selection of systems, featuring some of the most capable.

Rhino & Minerva

`Minerva` is an interactive tour-guide robot in a museum, designed in follow-up to `Rhino` (Burgard et al., 1998; Thrun et al., 1999). There are numerous similarities between the two models, but the latter has witnessed

[2] This philosophy culminates in the theory that robots should tie a string around their finger to remember what its doing.

significant improvements. Notably, there are an astounding 20 different asynchronous modules involved, including interaction modules *(human interaction, emotional state, web-interface)*, navigation modules *(mapping, localisation, collision avoidance, path planning)* and hardware interfaces *(sensors and effectors)*. Three concepts are at the foundation of this architecture:

- Learning — The map of environment is entirely built-up online, which allows unknown environments to be assimilated.
- Probabilistic Reasoning — Position is modelled as a probability distribution, updated with Markov localisation. The occupancy grid used to model the terrain is updated by applying Bayes' rule, which requires sensor models.
- Any-time Algorithms — All components in the real-time system are built around soft deadlines, which generally injects more flexibility into the design.

Interestingly, the collision prevention module takes into account dynamics, as opposed to just kinematics. This implies a certain amount of prediction in this typically reactive behaviour. Soft constraints force a move away from obstacles rather than seeking targets, and hard constraints are set-up to stop movement before impact. Dynamic obstacles such as people are filtered out before the probabilistic reasoning. Finally, the path planning takes into account danger of getting lost (avoiding large halls, hugging walls, and preventing ambiguous identical turns).

Xavier

`Xavier` is an office robot, based on a layered system (Simmons et al., 1997). The task-planning layer is the highest-level, asynchronously prioritising goals assigned by humans via a web interface, suspending and activating tasks. The path-planning layer computes the quickest path to a destination, using an A* algorithm that takes into account blockage probabilities *(doors, people)* and the risk of misinterpreting junctions. A navigation layer takes care of "virtual" low-level processing; this includes state estimation and execution of tasks using a POMDP (a state machine of possible cardinal movement directions). Finally, the lower level deals with physical sensory data and motor execution of the virtual orders.

Notes of interest include the following points:

1) The decomposition of the system is chosen by task function rather than by architectural constraint; reactive and planning components are scattered throughout the hierarchy.
2) Lower levels are not dependent on higher levels, which assures consistency and reliability.
3) The design emphasizes internal representation, improving the overall efficiency of the implementation.
4) No signs of environment learning can be observed.

Thanks to this technology, `Xavier` robot has serviced thousands of requests and travelled hundreds of kilometres.

AuRA

`AuRA` uses a hybrid approach (Arkin and Balch, 1997), split into a deliberative hierarchical planner *(for mission planning, spatial reasoning, plan sequencing)*, and a reactive controller using schema theory *(behaviour fields super-positioned)*. These provide an improvement over potential fields, which have some problematic issues. Various techniques are used to resolve the problem of local minima: deliberation, using noise as "reactive grease", momentum, the `avoid-past` behaviour, and other learning and adaptation techniques. Gain values also allow the scaling of each field. There is no need for arbitration since all the behaviours are combined in an order independent fashion.

The solution has numerous advantages, identified by its creators: modularity (in design and experimentation thanks to iterative improvements), flexibility (by a variety of learning mechanisms, and self-optimisation), generalisation (it seems to perform well in numerous test environments), and "hybridity" (both deliberative and reactive policies combined).

SSS

SSS is based on three levels; a symbolic layer performs landmark based navigation (using junctions, appearance and disappearance of walls) enabling the behaviours required in the lower-levels, a subsumption layer combines various reactive behaviours, and a servo-control layer deals with angular velocity controlling rotation ands linear velocity controlling translation (Connell, 1992).

There are a few clever ideas used in this system which deserve some attention:

1) The geometric maps are very coarse, ignoring small dynamic obstacles.

2) It is able to follow path segments reliably, thanks to behaviour based motion.

3) Localisation is based on rectangular corridors, and their layout is used as landmarks.

4) The robot tries to stick to the average heading over short-term past, providing a small sense of persistence.

5) Wall-following identified as being a useful behaviour, allowing the robot to get around simple obstacles reliably.

Despite its old age, SSS remains a landmark in the field, which many other researchers have used as guidance.

ATLANTIS

This is **A** **T**hree **L**ayer **A**rchitecture for **N**avigating **T**hrough **I**ntricate **S**ituations (Gat, 1992). The structure described is generic, and specific instances need to be designed on a per-problem basis, which does admittedly require a reasonable amount of work. Modules of the framework include:

- Controller — Coordinates reactive sensorimotor activities. To allow complex transfer functions, a specialised programming language was devised, intrinsically supporting state-machines and inter-module communication.

- Sequencer — Arranges sequences of activities and deliberative processes. When designed to deal with cognisant failure (aware of its own mistakes), problems can be detected and planned around.

- Deliberator — performs the planning computations, initiated and supervised by the sequencer. These take the form of a LISP program whose output is passed merely as suggestion to the rest of the system.

This framework has been applied to both indoor and outdoor navigation, in object collecting robots as well as Mars rovers.

Saphira

Saphira's architecture emphasizes the 3 C's: coordination between the low-level effectors and high-level goals using an abstraction hierarchy, communication with inter-agent cooperation and NLP, and coherence of knowledge thanks to a strong internal representation which aids coordination (Konolige et al., 1997).

The organisation is partly vertical and partly horizontal; first comes a layered perceptual system *(dealing with raw depth, surfaces, objects, then people)*, and secondly an action hierarchy *(reactive, purposeful, then task schemas)*. On top sits a `PRS` (`Procedural Reasoning System`), which determines weightings of specific behaviours based on goal-oriented objectives. The control problem is split into basic-behaviours *(obstacle avoidance, wall following)*, specified in fuzzy logic (using basic operators `AND OR NOT`). This means each of them specify a fuzzy 'distribution' of possible movement called a desirability function, which is then combined using a defuzzification process taking the weighted average. One should particularly note the following:

1) Basic behaviours should not conflict, since the weighted average defuzzification process fails. More complex combination technique have been developed to counter this.

2) Coherence in behaviours is enhanced by using internal artefacts; harmless artificial geometry is inserted to guide the robot through an area, which would normally trigger a different behaviour (e.g. following a wall and reaching a small temporary opening). This is advantageous for decoupling the motion control from the noisy sensorial input.

3) Localisation is done by finding linear features in the sensor data, and locating the robot relatively to the sensed walls.

4) Grounding keeps track of artefacts by converting features to hypothesis; this applies to walls and corridors.

AMOR

This is a hybrid architecture designed for office delivery tasks, in a natural dynamic environment (Hans Blaasvaer and Christensen, 1994). The system is distributed, comprised of components running in parallel. Logically, it is divided into three categories:

- Sensing — Includes generic obstacle detection, and more specific procedures such as door finding.
- Planning — This is composed of local reactive actions, as well as higher-level task management.
- Action — Allow execution of effectors as well and control of the data acquisition.

At the base, reactive behaviours assure reliable navigation, while on top it is guided by a higher-level strategic executive. However, one should note that the robot is given *a priori* maps of the office *(doors, walls, connectivity information).* Other object properties, however, are assimilated and dealt with online.

DAMN

As a **D**istributed **A**rchitecture for **M**obile **N**avigation, this approach advocates the development of independent behaviours (Rosenblatt, 1995a). These activities can be reactive as well as deliberative; their implementation is wrapped inside a black box. Their desired actions are communicated to a centralised arbiter under the form of votes. The arbiter is responsible for combining them and generating appropriate commands. This is done taking into account priorities and objectives, which provides weightings for each of the behaviours.

In practice, there are 5 activities that are used: avoid obstacles, follow road, seek goal, maintain heading and avoid tip-over. These behaviours are handcrafted based on domain specific knowledge. The arbitration is split into three components, constraints (e.g., speed limits), actuation (explicit positive/negative votes for specific actions), and effect (implicit request for a resulting state, regardless of the action). There are some major benefits:

1) There is no need for a hierarchical structure; the multiple levels of goals and constraints can be simultaneously fulfilled by the arbiter and the distributed nature of the architecture.

2) Subsystems are easy to integrate, and the development process is simplified by incremental design (also suited to evolutionary strategies).

This approach has successfully been applied to several autonomously controlled vehicles, mainly outdoor ones.

2.1.3 Synopsis

In brief, there is no such thing as the ideal architecture. Each design reviewed here has some issues, though they generally do not arise in the domains they were intended for. Often pushing these solutions to their limits reveals problems with the design.

Flexibility

Some navigational frameworks are very generic, and require a fair amount of implementation, testing and debugging to get fully functioning, while others involve little additional work to get up and running as they have been designed with direct applications in mind. This distinguishes the creation of navigational solutions with the tools required in the process.

Policy

Generating the behaviours is also a controversial issue, since either extreme is possible; both deliberative and reactive approaches have been successfully demonstrated (though again, “success” is typically a context specific metric). However, it does seem that recent literature has settled on hybrid approaches, allowing the problem to be split modularly. An unspoken consensus has been reached to design the planning to make as few assumptions as possible, and let the execution deal with problematic dynamic conditions.

Modularity

With the increasing complexity of navigation systems, engineers increasingly favour a modular approach — whereby components can be implemented independently. The process of dividing the system is still a matter of debate. Decomposition can be:

- <u>Behavioural</u> — In some cases, distinct activities can be observed (like following walls or preventing collision). These can usually be designed separately, but some form of arbitration is required in order to decide which behaviour gets control.

- <u>Functional</u> — The system can also be made of components which are required to perform different operations and computations. This is generally done based on previous research in sub-areas of navigation (e.g., planning and localisation).

- <u>Representational</u> — The information sensed can be analysed, and interpreted at different conceptual levels (distance data, cell based, entities). Different operations can be undertaken based on these different representations; these are also usually linked to a similar effectory decomposition.

Often, as you may expect, designs show traits of multiple of these categories.

Arbitration

Systems often have multiple components, whether behaviours or computational processes. It is possible to let each of them control the animat, ignoring each other. However, this often has malevolent side effects. Either extensive design is required to create emergent coordination, or alternatively some form of command fusion like a voting system. This corresponds respectively to distributed and centralised models.

Design

The engineering of a navigation system is very similar to the creation of a computer program. Starting from the specification, a top-down approach can be taken to draft the interface and develop the matching implementation. With a bottom up approach, working components are assembled together, in order to match as closely as possible the requirements.

Abilities

Given the necessary capabilities of the system, there are different ways of blessing it with navigational skills. Firstly, an expert system can be devised; the developer generates a set of fixed rules, which serve the animat throughout its life. Second, a learning system can be setup to learn the behaviours required — either offline as a pre-processing, or online.

Representation

Can the behaviours be obtained reactively, or with a clever arbitration of simple activities? If so, then a representation of the environment is not necessarily required. Otherwise, when planning or other deliberative processes are essential, the animat must store an internal model of the world. This can be done in an implicit fashion, or with an explicitly detailed version.

Knowledge

For those animats needing a representation to operate at a higher-level, the issue of how it is obtained arises. This can be given to it apriori, when appropriate. The acquisition of the knowledge can also be assisted by sensorial cues placed by designers. Finally, the entire environment can be learnt automatically without any external help.

Sensing & Cognition

The sampling of the data, and the internal processing can deal with different types of information. This can be continuous (using floating point numbers), discrete (using booleans or integers), or even fuzzy (with fuzzy variables). This affects design as much as the development process.

2.2 Academia

This section looks into projects from the rest of academia — outside of robotics research. Many non robot-based projects focusing on navigation often do so coincidentally; problems in navigation are often somewhat convenient to develop and debug, and provide a good demonstration of generic algorithms. There are few navigation systems developed for the sake of it!

Soar Quake-Bot

The `Soar Quake-Bot` (Laird and Duchi, 2000; Laird, 2000) is an interesting concoction since it is the first academic implementation to tackle a computer game navigation problem. This aspect of the bot is brushed over in every paper, but Dr. John Laird (2001) was kind enough to take some of his time to describe the system. The bot can scout around the terrain when not disturbed and create a simplistic model of its environment, which it can use later as part of the deliberative process. The bot is fully autonomous, but has significant limitations. Firstly, the bot can only handle specific layout, where rectangular rooms are connected by hallways. Secondly, the connectivity information is stored in a fashion which does not scale up elegantly; it is of order $O(n^2)$. Finally, the bot goes through a terrain learning phase, wherein it moves in a very deliberate fashion in order to identify and understand its surroundings. However, since it has managed to remain fully autonomous, this agent remains a reference for simulated navigation — despite the limitations.

Quake A.I. Agent

In his masters dissertation, Jonathan Ritchie (1998) devised a fully fledged navigation system for a game agent. His approach unfortunately was heavily dependent on humans for preparation; they were required to move around the level dropping nodes in the corner of rooms. The bot can thereby locate itself, and plan routes through the rooms as determined by its higher-level strategy. Though this approach did spawn incredibly efficient representations, the time required to map a small terrain was too significant to be practical, both for users and game developers.[3] The dynamic path-planning, on the other hand, was extremely practical and elegant. The digraph created from the preprocessing phase is essentially distorted around nodes that are of value to the agent; distances of adjoining paths are artificially set to their minimal value. Then, when Dijkstra's standard shortest-path algorithm is used, it is locally influenced by the counterfeit gain of travelling by the beneficial item. The length of the edges limits the extent to which path can be distorted;

[3] In his thesis, Jonathan points out the problems with the time consuming process of laying the room nodes, and advocates its improvement as future work.

distorting short edges will have little or no effect on the global path length, and not influence the outcome of the planner. It generally produces good results, but it can be difficult to anticipate how the navigation will behave. Additionally, this excludes any chances of using a distance heuristic for the search.

Agent Framework

James Seymour (2001) developed a Flexible and Extensible Framework for Modelling and Simulating Physical Agents. From a navigational point of view, it consists of an amalgamation of past research, allowing the control of agents in 2D worlds. The map can be learnt dynamically as a regular grid, based on sensory information. Deliberative algorithms can be mapped onto reactive guiding strategies, combined with other activities using layered subsumption. This allows, for example, flocking behaviours or D* algorithms to be implemented and integrated.

2.3 Military

Tactical Movement Planning for Individual Combatants

Reece et al. (2000) present a solution that is very reminiscent of the academic approaches described above. The terrain is discretised into small cells, which can be assigned varied cost based on cover and direction. Indoor and outdoor scenarios can be handled interchangeably as the graphs inside link with the grid outside. A standard A* algorithm is used to find the optimal path over this search space. A post process is applied to the resulting path to smooth it, and determine the appropriate posture.

There are some interesting points to be made about this approach.

1) Only potential fields are considered as reactive behaviours, and they are discarded since it was difficult to avoid obstacles while remaining close to them.

2) The standard Euclidian heuristic used in A* becomes ineffective when intricate cost variations in the cells can be observed.

3) The hard work done by the search to obtain an optimal path is negated by a post-process aiming to obtain human-like behaviours.

Though the approach works extremely well for the application it was designed for, such problems may prove problematic in other scenarios.

2.4 Games Industry

Along with the increasing focus of computer games on artificial intelligence, navigation systems have also improved; they are probably even at the inception of the trend.

Eraser Bot

The `Eraser Bot` was the first good simulated opponent available for Quake 2 (Feltrin, 1998). This was mostly due to its navigation abilities, since the code was actually develop to test the ideas of the author, Ryan 'Ridah' Feltrin. However, conceptually speaking, the bot's movement cannot be classed as autonomous, since its representation of the world are based on routes travelled by the human players. So essentially, human perception is used to establish a set of routes, which is saved to disk for future use; these files are compressed yet still weigh from 10k to 30k, depending on the size of the level. The creation of such route files is not an especially difficult task, but it is something the user must pay attention to. Indeed, the player is advised[4] to move around a lot, and return to the start of routes as soon as possible, presumably so the internal representation can be more efficient. Getting this right is a task difficult enough to justify the online distribution of route files prepared by experts. Despite all this, a revamped version of this navigation code was used in the commercial game `Kingpin`.

[4] Recommendation taken from the `Readme.txt` file provided with the bot.

Unreal Tournament Bot

Originally, the `Reaper Bot` was designed by Steven Polge (1996) as add-on for the revolutionary *First Person Shooter* (FPS) `Quake`. Its major feature was the ability to learn the map dynamically. This was done by placing nodes at specific locations in the world, where items existed. Then, during play, these items were connected together when the bot was found capable of moving from one to another. In practice though, this is done by a piece of omniscient code which makes the human adversary drop bread-crumbs; the author calls this *'dynamic mapping'*. The exploitation of the internal maps still seems to cause some problems, notably as the bot was coded to reach out and grabs items that are physically out of reach, breaking the basic physical rules of the game — (Anonymous, 1997).

This technology was developed and expanded as part of Epic's `Unreal`, another ground breaking *FPS*. However, many were disappointed with the Artificial Intelligence in general, despite it being the first game to ship with built-in bots. The following generation of game AI within Unreal Tournament (Epic, 2000), on the other hand, is said to be the best around. The custom editor provided with the graphics engine allows the user to place the waypoints manually. This allows the human designer to assist the navigation — otherwise a potentially troublesome process.

Quake 3 Bot

The `Quake 3 Arena Bot` (van Waveren, 2001) can be considered as a polished amalgamation of the author's previous experimental efforts: `Omicron` and `Gladiator`. The navigation itself is very robust, since it is based on the intrinsic BSP[5] representation of the game graphics. Essentially, the world that the player can collide with is composed of basic 3D primitives: prisms, pyramids, and select other irregular polyhedra. By expanding these volumes, the author (J.M.P 'Mr Elusive' van Waveren) was able to determine the current location of the bot; this forms the *Area Awareness System* which the bot is based upon. The connectivity between these areas is then established via knowledge-based preprocessing,[6] which effectively simulates the player's potential movement. A graph with a two level hierarchy can then be built with the room connectivity information, which is used for the path-planning. This is accomplished by a breadth first routing algorithm, whose result is stored in a cache.

PathEngine

PathEngine is a middleware library (Young, 2002), aiming to handle shortest path queries for in-game agents. The solution is based on a precise knowledge of the ground mesh (areas which can be walked upon by the agent). These can be exported as triangles from the same 3D editing tool used to create the polygonal world itself. The library processes this mesh to determine accessibility for each agent. A conceptual graph of the connected areas can thereby be established. This is used to perform standard shortest path computations, and returning the result requested by the game AI.

2.5 Analysis

Anyone with experience in navigation systems cannot help but feeling a sense of amazement for well-designed architectures, almost as for impressive edifice from the Renaissance. However, this is often of very little consequence in real-life — just like magnificent buildings can be very awkward to live in! This section contains a list of practical criteria used to examine the potential of the solutions in real-world applications, and specifically the problem we are concerned with.

2.5.1 Criteria

The following paragraphs provide a description of the deciding factors used for judging the solutions. They were also partly chosen to match our goals and intentions. Therefore, some will also be part of our formal requirements listed in the *Design Chapter*. For now, it suffices for them to allow satisfactory classification

[5] `Quake 3` uses a proprietary Binary Space Partitioning file format to store spatial coherence between the polygon brushes.

[6] Precomputation usually takes a couple minutes of dedicated processing time to complete on average maps.

of the navigational architectures already reviewed. That said, despite the criteria being effectively atomic in nature, they may still conflict with each other. From this arises the difficulty in creating a system which successfully satisfies each of the constraints.

Autonomous

Like most terms in science, autonomy is extremely overloaded. The worst instance of this involves considering remotely assisted entities as autonomous, or even those fully controlled by humans (like in `Robot Wars`). Animats belonging to groups may also be classed as autonomous themselves, despite their dependency on others in that group. However, the most widely accepted definition in the field — and the one that makes the most sense theoretically — is of a truly independent animat, capable of controlling itself under all circumstances. In practice, this can be achieved by imitation and self-learning, but other human designed techniques are equally plausible.

A consequence of navigational autonomy is that no objective, or third party data may be passed to the entity. Only information that has been perceived by the animat may be used for its movement. This also implies that no *a priori* knowledge or pre-processing of the terrain is allowed. Solutions will be rated on how well they can cope without such “illegal” information.

Dynamic

In realistic simulations, the environment can witness terrain transformation, moving creatures, temporary objects and changing conditions *(e.g., doors opening and closing)* among others. These all contribute to making the world a complex dynamic problem, which must be dealt with.

While typical environments do remain mostly static, the dynamic parts are often key features *(like doors, platforms and crowds)* — despite being limited in space and time. The navigational architectures described will be judged on how well they are capable of dealing with such scenarios.

Flexible

Navigation architectures can be designed in a very abstract fashion, as well as for specific problems. In fact, the problem itself can be tailored to suit the solution, by using recognisable marks on the walls or lines on the floor. In natural environments, many different scenarios may be encountered, including many that are not specifically designed for the animat!

The flexibility of a solution lies in its ability to cope with such a wide range of conditions, indoor as well as outdoor for example. Variable terrain layouts ranging from dense indoor obstacle courses to wide open plains will be considered when evaluating the applicability of each scheme.

Human-like

Mammals in general — and humans specifically — have a very distinct navigational style; the motion is extremely elegant and smooth. It remains effective overall, while in some situations errors can be committed. These are both high-level *(e.g., taking the wrong turn)* or low-level *(e.g., falling off a cliff or stumbling over a banana skin)*. This is not an error due to the lack of adaptability, as often witnessed with AI.

While some systems have been specifically designed to handle some of these cases, others have not explicitly. This criterion will consider how well the solutions can be adapted to exhibit such realistic movement.

Real-time

In theory, the navigation behaviour could be computed in different fashions: pre-processed, devised incrementally, temporarily cached, or fully on the fly. Whichever approach is chosen, a timely response is required by the animat, since the simulation itself happens continuously.

Current systems all have some kind of upper limit on the processing that can be done. If a prompt response is required in every case, these restrictions will need to be taken into account. The systems will be examined on how well they would perform in a very demanding context.

2.5.2 Observations

From the background reviews, this section tries to extrapolate trends with respect to the criteria established. Both positive and negative evaluations will be described, as they can be equally influential for the unfolding of the project. Attempts will be made to keep the concepts abstract, but as relevant as possible for the design; problem specific considerations will be given in the following module chapters.

Autonomous

In computer games, agent autonomy is very limited. The programmers handle the navigation by using global terrain data, and designers guide the paths by placing waypoints. However, the player is not generally expected to actively help the animat in any way. Autonomy is not something that is seen as important for such an application.

Academic work seems to be slightly more adept at dealing with autonomy. For the solutions reviewed, this is achieved by creating navigational behaviours that do not rely on any form of objective data. Either the capabilities of the systems are kept simple, or assumptions are made about the environment (without which the system would fail). Such independence of control is something that researchers strive towards.

Robots typically only rely on their sensors to perform the navigation (as opposed to global positioning systems and global maps for example). This is a restriction that researchers have to put up with more than anything. Entire architectures have been designed around interpreting the sensorial data, in order to allow sufficiently complex behaviours.

Dynamic

In-game animats generally have reasonable capabilities for dealing with dynamic conditions. For starters, locked doors are generally taken into account by the path planning. Closed doors can be open with a simple behaviour, which can be taken into account by increasing the length of the path concerned. Platforms and elevators are dealt with similarly. Queries are mostly dealt with dynamically, as the targets unexpectedly change based on the current state (although these are kept to a minimum, and even designed around). However, other players and dynamic obstacles are not always taken into account; violent assaults can generally resolve the problem! Changing terrains are not handled either, unless the pre-processing was made aware of that.

Other academic implementations seem on a par with these commercial implementations from a technological point of view, although unafraid of fully dynamic searches like D*. With regards to human-like behaviours, they generally remain less polished than professional implementations.

Robots are a reference for handling dynamic surrounding conditions. All of the architectures reviewed are either pure reactive behaviours (which can cope with the problem intrinsically), or have reliable reactive policies at their base. This allows them to deal with people, doors. The systems with map learning also tend to use probabilistic reasoning, which allows them to update their representation over time; changes in the terrain are taken into account. Path plans are also done dynamically, due to the high-probability of failure.

Flexible

In game simulations, the flexibility is limited as much as possible, but gives satisfactory results. To squeeze as many processor cycles out of the navigation code as possible, it is not only restricted to the types of terrain found in the games, but also to specific "levels". The designer often has to place waypoints manually, which makes supporting different terrains fairly tedious and time consuming. When waypoints are not placed, an extensive pre-processing phase is required before any kind of movement is possible.

Some other research done in academia is a slight improvement. While the type of terrains is restricted to match the exploration behaviours, specific layouts can be learnt automatically online. This decreases the hassle entailed by handling novel environments.

Architectures designed for real robots are striving towards flexibility. That said, problems tackled are still often limited to office tasks, or outdoor navigation. The reactive robots tend to handle any configuration transparently without realising it. Those that need cognitive maps to perform deliberative reasoning usually learn them online. This makes them reasonably flexible, though the representation is generally tailored to fit the problem (e.g., corridor intersections as landmarks and visual cues).

Human-like

Animats in computer games are rated by their ability to resemble humans. It's the AI programmer's job to make sure they fool the human players. Recent efforts are admittedly good; path-smoothing techniques can generate organic trajectories while paths can be realistically chosen to minimise risk. However, while overall the illusion is convincing within their usual context, it can usually be improved upon; not to mention that behaviours can potentially go horribly wrong in unforeseen situations (Champandard, 2002b).

On the other hand, robots struggle for even reliable motion, so human behaviours seem far out of reach. This is definitely something researchers strive towards, but the underlying robotic hardware often doesn't lend itself to the task.

Real-time

Programmers of game agents often see the worse part of the computation budget. This forces them to neglect otherwise interesting features, in order to squeeze the most processor cycles out of the logic. The result is an extremely lean implementation, with everything pre-processed where possible.

Academic work often has additional features focusing more on the research aspect. This causes them to remains a step behind with regards to efficiency and real-time capacity, though still reasonable.

Finally, robots are intrinsically capable of real-time performance, due to their design requirements. That said, they typically have dedicated hardware to exploit, which is usually fully required to analyse the sensory data.

Feature	Robotics	Game AI	Academic
autonomous	generally	<u>**never**</u>	possible
dynamic	intrinsic	<u>**minimum**</u>	overhead
flexible	reasonable	<u>**limited**</u>	<u>**simplified**</u>
human-like	<u>**no**</u>	usually	<u>**no**</u>
real-time	<u>**exclusive**</u>	shared	passable

Table 2.1: Conditions fulfilled by existing systems from different domains.

2.5.3 Summary

To sum, up the pre-conditions establish are satisfied individually by some approaches, but never as a group. The synopsis looks at the best options of the three major trends, robotic designs, in-game navigation system, and academic architectures.

Table 2.1 demonstrates this; cells with underlined text show that unsatisfactory results are achieved for this feature. This is somewhat an oversimplification, but reveals the areas that this dissertation wishes to address.

The motivation for creating a custom system is to satisfy these overall requirements within the context of virtual environments. Existing systems cannot be directly applied, as they have not been considered with each of the premises in mind.

Chapter 3

Design

"Navigation is about wayfinding, you can't treat it as separate because many other things run parallel with it." —*Clement Mok*

In the task of engineering a complete solution for our navigation problem, the initial step is to design the architecture based on a conceptual study. This essentially involves determining what behaviours are necessary as well as organising them in order to satisfy the external expectations.

This chapter will start by formalising a set of requirements. Then, building on the observations and lessons extracted from the literature review, our generic approach in tackling the problem will be explained and justified. An interface with the rest of the system will be drafted, thereby laying down a foundation for the implementation. Finally, the underlying architecture is to be organised, with the role of each individual task described.

3.1 Requirements Revisited

In *Section 1.9* of the introduction, a brief list of project objectives was given somewhat informally, but from a practical perspective. These can now be further refined to take into account observations from *Section 2.5.1* of the literature review. However, these criteria used to judge existing work were rather more analytical. Since they are nonetheless important concepts for our architecture, these latter points need to be incorporated[1] into a new set of requirements that extends and combines those listed in *Section 2.5.1*, thereby improving the development process. Thus, later stages of the design and implementation would correspond to fewer requirements.

Environments

For better or worse, simulated environments in games are becoming more realistic — but more complex. This includes continuous 3D worlds of potentially infinite size. Increasingly intricate scenarios and storylines are forcing game creatures into unexpected situations, in which they are naturally required to move around unhampered. As such, no assumptions should be made about the animat's "*natural habitat*," as they may exist in a wide variety of settings: indoor and outdoor, deserted and highly populated, complex and very simple. The ability of the navigation to take into account these changing surroundings will thereby increase the likelihood of successfully reaching its goal.

[1] There is a (somewhat ambiguous) mapping of the objectives onto the requirements.

Already today, numerous interacting objects can instigate problematic dynamic conditions even in mostly static worlds. Ideally, these require a dynamic architecture, which would scale well for the future as worlds themselves become more dynamic. Admittedly, many such situations can currently be foreseen and designed around with relative ease. However, as they become more frequent and unexpected, only a flexible navigation system will spare designers the aggravation of hard-coding these necessary behaviours.

In practice, this corresponds to the model of the flow of information, and how it is taken into account by the architecture. Also, this affects what assumptions can be made from the sensory input. The components will have to be chosen and designed to deal with these situations autonomously.

Simulation

Conceptually, this virtual world is simulated in a fully continuous fashion, and all interactions happen in real-time. That said, for efficiency reasons the agents are rather given discrete opportunities to reason, though this happens at a high-enough frequency for it not to be an issue. However, this does not guarantee that events happen in a deterministic fashion.

Only a real-time system will be capable of coping with these restrictions at interactive rates. Efficiency is also crucial since the system is expected to perform non-exclusively on the hardware, sharing resources among agents as well as with other components. To maximise the quality of the final product, the computational overhead of the module should also be minimal, or at least scalable with respect to the quality of motion provided.

Practically speaking, this influences the design of the computational processes used by each module *(e.g., heuristic, deliberative search, reactive)*, as well as an overall efficiency of the implementation.

Embodied Animats

Each creature is fully genuine, modelled as an embedded entity. To that end, the information it receives is physically limited, as well as the scope of its possible actions. As such, no pre-processing techniques common to most complex navigation systems will be possible since that bulk information is simply not initially available.

The design will have to take such limitations into account by being fully autonomous, which reduces the amount of information it has to deal with.

From an engineering point of view, this translates into the design of the interface with the environment, and how the body's constraints are enforced. This also affects what assumptions can be made during the design, as the system will either have to cope with ambiguous information, or somehow reason to extract meaning from the data.

Behaviours

Increasingly high-expectations are placed in the quality of the behaviours observed. Our navigation module should be capable of providing realistic motion ranging from complex patterns involving jump sequences to hesitation-induced pauses — as well as errors like reversing into walls or getting lost by assuming a different starting location.

For the system requirements, this implies a design of components which interact together and emergently create human-like behaviours. They also need to be flexible enough, so appropriate behaviours can be crafted and inserted into the system when necessary.

In practical terms, this has a great impact on the parameters chosen for low-level motion. Also, it reveals the importance of the conceptual reasoning in the higher layer to emulate the human cognition process.

Higher-level AI

As the navigation component will constitute a small part of a complex system, it must provide an intuitive handle for specifying the desired result. As sophistication increases, one can expect these higher-levels to require complex navigation, with intricate time-dependent responses. Therefore, the range of behaviours provided also needs to be sufficient to fulfil these needs, yet never inconsistent.

Only a dynamic system will be capable of adapting to such changing orders. This can potentially be done in a realistic fashion by incrementally incorporating the new requests into current motion, or radically computing an entirely new plan.

All together, this fully determines the interface with the client software specifying the rest of the animat. The overall expected functioning of the system is also thereby formalised.

Development

The process of design and implementation often go hand in hand, in an incremental fashion. As such, the architecture of the system should facilitate both these tasks, reducing development times. The system should also be easy to test and debug, and customisable enough to allow simple adjusting and extending at will.

By definition, this implies building a very flexible system, allowing behaviours and modules to be included — almost at the click of a mouse. Such modularity allows incremental development, as well as facilitating debugging and improvements.

For the implementation, this implies a very object oriented style of programming, which would allow the components to be relatively easily organised and extended. The level of competence required to modify the system should be proportional to the complexity of the operation. Experts are capable of changing the entire skeleton of the architecture, connoisseurs can update specific tasks and behaviours, while beginners have the option of configuring them.

3.2 Approach

The task placed in front of us is to design and implement an architecture that manages to satisfy all of these requirements, without neglecting others. This section underlines the general concepts at the base of our model allowing us to satisfy these requirements. The creation of specific modules will, wherever possible, attempt to follow these guidelines.

3.2.1 Embodiment

In robotics, embeddedness can be an undesirable and somewhat annoying property; so much so that an increasing amount of recent research is trying to enhance robot perception abilities by overloading them with sensors (both internal and third-party). In fact, the field of pervasive/ubiquitous computing is trying to break these constraints — to the satisfaction of many researchers.

However, for the simulation of realistic virtual animats, embodiment appears fundamental. The physical constraints imposed on their animal counterparts define their relationship with the environment, which some ethologists believe is a key component in the emergent behaviour observed. In navigation particularly, we believe it has numerous intrinsic advantages that influence the realism of the motion. Therefore, beyond considering it as a requirement, we have gone through great lengths to actively enforce this embeddedness.

Not only is the perception restricted to sensory range, but it is made fully subjective — that is, relative to the current state of the animat. Additionally, sensed data is generalised and fuzzyfied to some extent, which increases the realism of the simulation.

3.2.2 Learning

Many believe learning is an essential component of intelligence. Rather, we present it as a human characteristic, whose modelling will increase the authenticity and realism of the animat. Additionally, this simplifies the manual handling of somewhat complex scenarios, at the cost of the overhead needed to develop the learning algorithm.

In our system, the learning is split into two sub-classes — as generally believed by psychologists. The declarative learning gathers facts about the animat and its environment, while the procedural learning attempts to produce useful behaviours.

Our behaviour learning takes place offline, which simulates the navigational abilities that mammals are blessed with over time (thanks to evolution and early life development). Factual learning takes place online, to prevent being restricted by environment specific knowledge. This also allows interesting effects like memory lapses to be simulated, as expected from humans.

3.2.3 Heuristics

Early AI work focused heavily on symbolic representation and search algorithms. While this approach usually manages to solve what it sets out to do, it remains a very inflexible solution ill suited to the simulation of mammal-like creatures. Modern biologically inspired approaches are applicable in some cases, though the search paradigm remains engraved in AI developers' brains.

In the development of our navigation, where instantaneous reactive behaviour cannot be used, we devise heuristic solutions as much as possible. This allows them to order to provide "quality of service" results, where the approximations improve over time. Basing our solutions on psychological observations allows even more realism to be introduced into the mix.

3.2.4 Stochasticity

Along with the search paradigm, developers seem to have kept the attitude that nothing but the optimal result should be returned. Though this may have benefits for industry optimisation problems, the simulation of realistic animats often requires more than flawlessness. By this we imply that many interesting and desirable behaviours do not arise from perfection — contradicting many sci-fi stereotypes.

Instead of randomising perfect results, which would clash with the use of heuristics, we build this stochasticity into the solutions themselves. When done in a reasonable fashion, efficiency is not sacrificed at the cost of a more captivating behaviour.

3.3 Interfaces

From the requirements and the informal guidelines, a robust interface can be designed; it in itself establishes the skeleton of the architecture. The design of the interface is crucial since it can make the difference between a useless black box and an extremely effective tool.

Since the navigation system is to be developed as a separate component, the interface with the rest of the software will also provide a fundamental definition of the functionality required, as well as formalising the assumptions made from other modules. These external interactions can be narrowed down to two types: environmental and with the higher-level AI component, both of which we will discuss in turn.

3.3.1 Environment Interaction

This formalises the information available to the animat from the world it's in, as well as the actions it can perform (input/output). It has the following properties:

- Sensorial Honesty — The information given to the animat is restricted to that within sensory range. The physical laws of the virtual environment are enforced actively, which implies the system is more likely of applying to real-world scenarios, given adequate pre-processing to support the assumptions. For example, the animat cannot see objects through walls.

- Subjective Perception — The agent can access the environmental information via sensorial queries; they can only return values relative to the current state of the animat, in the true spirit of embodiment. For example, objects in space appear relatively to the current orientation. In practice, this forces the agent to interpret this data to maintain a coherent internal state (if any).

- Inaccurate Information — Together with subjectivity and honesty, the agent also has to deal with inaccuracies. Once again, these are imposed upon it in a human-like fashion. For example, distance estimation of obstacles becomes more error-prone as it increases, and determining the exact identity of other players is not always possible when they are facing in the opposite direction. Customisable scripts can be written to formulate these errors parametrically.

- Noisy Actions — Finally, the virtual body of the animat is simulated with sufficient accuracy, imposing restrictions on actions undertaken (in a somewhat "physical" fashion). Therefore, the actions executed by the agent are not guaranteed to exactly match the requests.

Not only do such constraints provide a close link between simulations and real-life, but they also provide an extremely flexible development test-bed; solutions can be evaluated for a much wider range of settings *(noise, complexity, or timescale)*, and assumptions are much easier to insert into the system (allowing the development to focus on a particular component). We strongly believe that *"Software simulations are doomed to succeed,"* in the words of Rodney Brooks.

Aside these developmental challenges, we rank this approach as a key factor in the creation of human-like behaviours, which implies a greater level of realism in virtual environments. Indeed, this scheme has numerous intrinsic advantages over traditional models, as will be emphasised all along this dissertation.

3.3.2 Higher-level AI

The second interface that needs to be specified resides between the navigation system and the user's custom AI reasoning component that lets high-level orders filter down. Such an interaction requires both simplicity (in order not to overwhelm the programmer) and versatility (allowing the raw power of the underlying mechanism to shine through).

Instead of using explicitly specified targets, the system relies on distal reward values. This creates the concept of implicit destinations, which can be combined to guide the animat. The navigation systems must maximise its reward, in fashion similar to reinforcement learning. A knowledgeable reader may notice similarities with potential fields; the independent rewards indeed superimpose to define a navigation policy for the entire terrain. However, we associate this with arbitrarily complex layouts rather than mostly reactive navigation, which presents familiar problems but on a much larger scale (e.g., local minima).

Terminology

The simplicity and intuitiveness of the interface often relies on some underlying concepts, in the form of fundamental navigational behaviours.

- Seeking — This behaviour is defined by heading towards a fixed target; this is called `following` for a moving point, but the concepts remain the same. While these are usually associated with shortsighted reactive behaviours, we expand the definition to cope with complex terrains and more deliberative approaches, permitting distant occluded points to be specified.
- Fleeing — Conversely, this behaviour attempts to move away from static points (or `evading` dynamic ones). Once again, arbitrary obstacle configurations should be taken into account, not only uncluttered scenarios.
- Avoiding — This type of motion is abstracted from obstacle avoidance, but applied to any point in space; it must be simply circumnavigated. This differs from the previous bullet point, which is a more active intention rather than a passive action.

All other types of motion can be more or less trivially mapped onto these primitive ones, such as `patrolling`, `exploring`, and `finding`. The definition of the interface is based around the ability to elegantly specify which behaviours should be performed around which points in space, and how they should be combined together.

Definition

Though these behaviours are extremely intuitive and could be assimilated with a negligible amount of experimentation, a more formal definition may prove valuable to developers. Moreover, it will allow robust evaluation of the underlying algorithms, based on such precise metrics.

Both `seeking` and `fleeing` can be abstracted to the same definition. The value of a point in space x decreases exponentially (respectively, increases) with respect to the distance from the target point t.

$$V(x) = k * e^{-|x-t|} \tag{3.1}$$

Here, k is the factor scaling the behaviour; negative values indicate `fleeing`, whereas positive cases correspond to `seeking`. The `avoiding` behaviour may be easier to define with a simple tail-off coefficient, but generally can be mapped onto fleeing behaviours. The combination of different rewards is done by simple addition.

Rationale

To date, most deliberative navigation policies can be expressed as 'shortest-path to fixed destination' queries, in an attempt to keep things simple and efficient.[2] No doubt this representation was also influenced by the abundance of related papers within the context of *network flow problems*, *graph theory*, or *combinatorial optimisation*. Though applicable, this approach has serious limitations (short of calling them flaws); it relies on the ability of a decision-making module to map the navigational desires — however complex — onto single destination points. Since artificially intelligent agents are becoming more sophisticated, communicating such intentions to a navigation system only by specifying one target point is generally ambiguous and incomplete. We call this phenomenon behavioural aliasing, which cannot be neglected.

The most can be made of the limitation by carefully designing the destination selection function; prioritisation of the drives and exponential distance scaling of the selection probability prove extremely useful in practice. However, this tedious task remains non-trivial, not to mention the computational overhead incurred in the higher layers.

Additionally, we argue that the higher-levels are ill-suited to making decisions about the points in space that should be selected! In robotics, some systems separate the task-planning with the path-planning, in order to get around the problem with the interface. There is much redundancy involved, as the information used is often duplicated (travel times are used to plan the order of goals before paths to each are computed).

This is possible, as the skill of many artificial players in modern computer games demonstrates, or as the reliability of the robots running office errands testifies. However, we argue that the planning within the navigation system itself is better placed to perform decisions about such space-related problems, and can often do so more efficiently.

Motivation

As expectations become higher and requirements increase, cracks start to appear in the old model. Whispers of dynamic environments, multiple path combination and weighted destinations can already be heard in the game A.I. community. We believe a more generic definition that encapsulates the original would remedy these problems. Such a rigid specification would also:

- Allow testing and comparisons of different implementations on equal terms,
- Provide an interface draft in the development of a navigational system,
- Bridge the gap between behavioural studies and the maths-based theory.

This paper intends to define intuitive parameters that can implicitly control the results obtained by the path-planning algorithm.

3.3.3 Interface

The interface designed during this project has many links to the ideas described above. Some accessors allow the user to tweak the cost and rewards of particular locations in space freely. This is done either relatively to the current location using 3-vectors, or globally with regards to the world model using landmarks.

[2] Local reactive models do not rely on such an approach, but they have not been scaled up to the entire terrain as many problems tend to arise in the process.

```
void SetCost( const Vec3f& point, const float c );
void SetCost( const Landmark* l, const float c );

void SetReward( const Vec3f& point, const float r );
void SetReward( const Landmark* l, const float r );
```

While there are other main methods of control, these are the most useful. They tell the navigation system to find its way to a specific point, taking into account the current set of rewards and costs.

```
void Move( const Vec3f& destination );
void Move( const Landmark* landmark );

void Patrol( const std::vector<Landmark*>& l, bool ordered = false );
```

3.4 Architecture

Designing an architecture is a matter of determining what functionality the system must provide, finding a correspondence with modules that can be implemented, and organising them together in a coherent fashion.

3.4.1 Principles

Synthesising complex behaviours necessarily relies on achieving reliable primitive motion. In a similar fashion to horseback riding, the rider would have great difficulty choosing the way if the horse could not follow path segments reliably. We craft such straightforward navigational entities — called "hosts" — using a reactive mapping from the sensors to the effectors. This allows them to perform tasks such as obstacle avoidance, among others.

In such circumstances, one notices that elaborate movement is essentially a subset of reliable motion. Indeed, seeking distant targets requires avoiding obstacles along the way in a particular fashion. Instead of believing that this is just a matter of arbitrating appropriate reactive behaviours, we rather rely upon higher-level algorithms to influence the outcome. The reaction of the host can be manipulated by modifying the sensory data it requires. Such an approach can be understood as inserting (or removing) internal artefacts to the world model; these are harmless bits of geometry that push the animat away (respectively, a lack of geometry that draws it in).

We call the entities responsible for adapting this sensory information "parasites", as they indirectly abuse the capabilities of the hosts by bamboozling them. They are typically responsible for less trivial reactive behaviours like seeking and fleeing. Since poor performance of the hosts would also affect the parasite negatively, the latter are responsible for not overloading the former. This approach allows the behaviours of the hosts and the parasites to be combined in a smooth and natural fashion.

These reactive influences can be applied in weighted fashion to obtain compound blended behaviours. This provides a hook for higher-level behaviours to influence the movement. Notably, one of our algorithms is for path planning, which suggests the desired heading to the "seek" parasite.

The choice of the hosts and the parasites can be made fully based on the requests of the client AI. The entire system is driven from state to state by upper interface calls. On the other hand, tasks like learning of the terrain or sampling the surroundings can be done completely transparently. Such data structures are chosen to assist the behaviours only, and match their level of cognition.

3.4.2 Layout

The lower layer of the system consists of reactive behaviours, which handle the motion control. This component can be instantiated as any activity mapping sensory data onto primitive actions. The outcome of this process is a set of fully functional navigational entities (the hosts). They get full control of the motion effector, and remain liable for the survival of the animat with regards to navigation.

More elaborate behaviours form the higher level of the navigation system. Virtually any control algorithm can be implemented, as long as they can be expressed as a desired heading. An arbitrator is responsible for selecting these behaviours depending on the current state.

In order to provide a link between the upper and lower layers, the set of parasites is introduced to influence the motion. Their task is to intercept information passed to the hosts, and falsify it as necessary in order to indirectly control the animat. This parasitic approach to arbitration provides a hook for arbitrary higher-level components to affect the movement when appropriate.

Just like any other modern architecture, ours cannot be simply classed as layered vertically or horizontally, implying it falls quite nicely into the popular hybrid category. That said, many interesting parallels can be made with the classical architectures.

- Horizontal Layering — The bottom layer has a flow of information from the inputs to the outputs. However, unlike most other systems, the representation does not become more abstract with each layer; it remains the same right until the decision is made.
- Vertical Layering — There is a certain sense of vertical layering since both the representation and the levels of cognition vary from practical data to more abstract concepts. However, the link between these layers is only implicit as the upper layer can only suggests it moves to be made.
- Behaviour Based — The arbitrator can select a combination of hosts and parasites to generate any reactive behaviour. This can be done based on the state and persistent variables allowing activities to interact in a typically behaviour based fashion.
- Voting Systems — The arbitrator can also combine parasites and their corresponding reactive behaviours. These are blended together using weighted influences on the host. Mishaps can be prevented by monitoring the fitness feedback of the host.

3.4.3 Process

The main procedure of the navigation system can be understood as follows:

1) Select Host — The fundamental navigational entity must be chosen first. It will perform a reactive behaviour whose output will be directly sent to the motion effector.
2) Sample Surroundings — Using its sensors, the animat must gather information to update its internal representation of the local surroundings. This is a dynamic data-structure storing distances to obstacles all around.
3) Request Data — The host is now asked to request information from the cache of the surroundings, in order to base its motor decisions upon.
4) Prepare Parasites — Using the requests of the higher-level AI, specific parasites can be chosen in order to obtain the desired moment. Appropriate parameters need to be chosen for these parasites, which usually involves running another algorithm to compute the desired direction.
5) Apply Parasitical Influences — The parasites can now be applied by bamboozling the information requested by the host. In this fashion, the motion can be influenced in the appropriate directions.
6) Engage Host — Finally, the host itself gets to process the information it requested and decided what the final action should be.

Tasks that are performed transparently or simply run on a regular basis.

3.4.4 Parallels

Many similarities with other systems can be observed. While some were purely coincidental, they need to be acknowledged to confirm the position of their original authors.

- Xavier — Simmons et al. (1997) emphasises the independence of the lower layer, which increases reliability in the situations. Our lower layer is also self-standing, providing satisfactory motion even when no other components are active.

- SSS — Like the work of Connell (1992), our geometric maps are extremely coarse. Reliable reactive behaviours are relied upon to exploit them.

- DAMN — In the project of (Rosenblatt, 1995a), concurrent control algorithms can be combined with a voting system. This is reminiscent of our parasitic approach which can blend reactive decisions; multiple parasites applied to the host can influence the motion.

- Saphira — The work of Konolige et al. (1997) uses internal artefacts (virtual geometry) to avoid switching behaviours unnecessarily; this improves coherence. Our systems uses internal artefacts to blend behaviours and provide a hook for higher-level influences.

- Minksy — The representational promiscuity applied here involves choosing a model of the terrain that suits each layer best.

- Using incremental design ala Brooks, our system is built from the ground up.

- Seymour (2001), among many others, maps deliberative path-planning algorithms onto suggested headings. This reactive policy can be followed like a carrot on a stick.

3.5 Components

This section briefly describes each of the components used within the system from a conceptual point of view. Those causing issues in the implementation, or with particularly interesting algorithms will be described later in the dissertation.

3.5.1 Surroundings

This is a simple persistent data-structure that stores a discrete representation of the local environment. It is updated on the fly by querying the visual sensors, which pass back the distance of nearby obstructions. When queried by the motion controller module, a simple look-up is performed to return the estimate of the obstacle's distance. A consequence of this module is local spatial awareness; by continuously caching information about the environment, this can be subsequently queried in any direction, independently of the orientation.

However, since our system is tested in the context of navigation rather than for game playing, the ability of this module is not pushed to its limits. Indeed, the animat can mostly look in the direction of travel.

3.5.2 Motion Controller

This is conceptually the lowest layer of the system, working on a mostly reactive basis. The activities handled by this module generally map sensor readings onto movement actions. Specifically, this involves taking in the distance of obstacles, and outputting a desired direction and speed.

This component is the only one to be connected to the output layer: the motion effector. All other layers thereby require and depend on it. However, it does not explicitly perform the arbitration! The only inputs it is given are those that it requires for basic functioning.

The major innovations of our approach involve a modular feed forward neural network for controlling motion, combined with incremental evolution and high-level fitness parameters to generate the desired behaviour. This is described further in *Chapter 5*.

3.5.3 Reactive Influences

In order to combine more elaborate reactive behaviours with the primitive motion provided by the hosts, we advocate a parasitic approach somewhat different to existing work. It is based on a more "laissez-faire" attitude, involving a machine learning approach to bamboozling environment models. Inserting or removing obstructions can influence the host into appropriate directions.

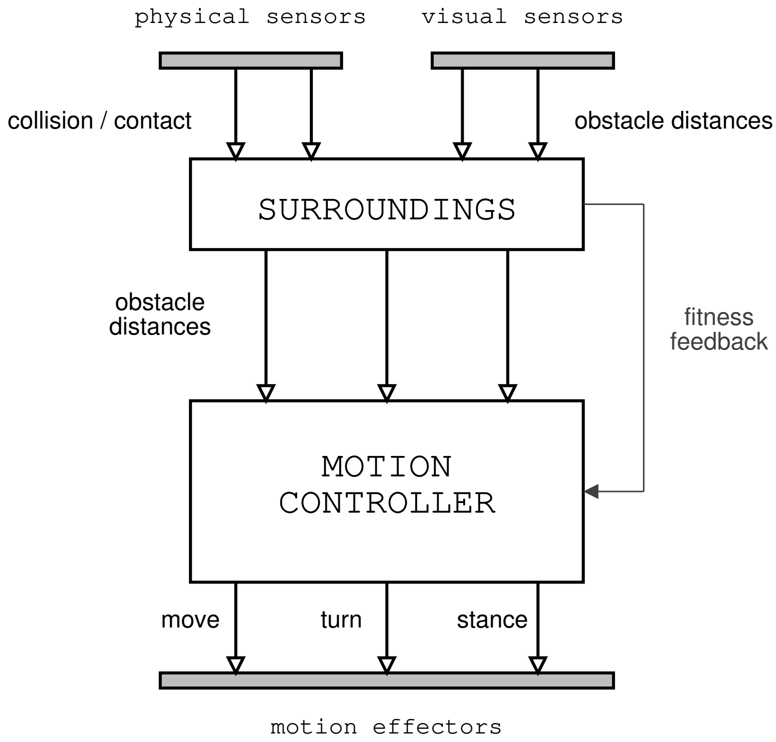


Figure 3.1: Simple host capable of reactive navigation.

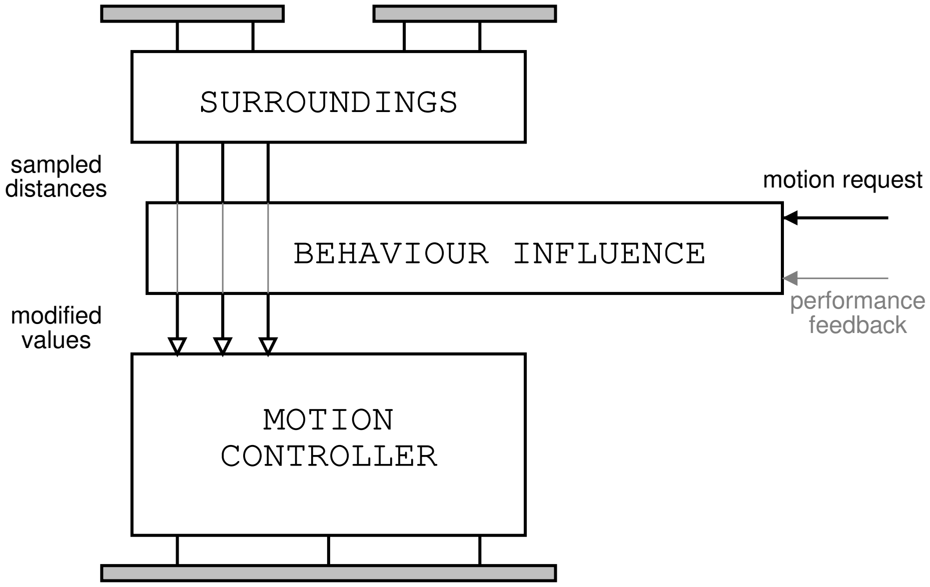


Figure 3.2: Parasites that bamboozle the flow of information to the host, thereby influencing the behaviour.

In practice, we use another neural network (a feed-forward perceptron) to intercept and rig data-flow from the representation of the surroundings to the motion controller. This will be discussed in more depth in *Chapter 6*.

3.5.4 Topography Learning

The creation of a cognitive representation of the terrain is done automatically, without the navigational agent having to consciously worry about it. Naturally, selecting the exploration behaviour as appropriate can assist this process, thereby speeding up the learning (e.g., actively seeking areas of un-mapped terrain).

Our approach is based on the incremental learning of a graph, inspired by the Hansel & Gretel fairytale. A pebble is dropped when no other pebbles are accessible, and it is linked together with the previously closest pebble in the cognitive map. The model is enhanced by adding features common in SOFM, such as node displacement and edge decay. This explains some of the similarities with growing neural gas (GNG), though the numerous assumptions about the navigational nature of the problem allow us to highly optimise the model. Indeed, the concept of spatial common sense (SCS) is introduced to generate simpler models, of higher quality, quicker than usual. Additionally, human-like features such as cognitive constraints, preferences and habituation further contribute to our incremental self-organising topography. *Chapter 7* is fully dedicated to describing the algorithm and data-structure behind this concept.

3.5.5 Path-Planning

This module relies on the learnt topography to deliberatively compute dynamic paths to destination nodes. This is done in an implicit fashion since the path is passed back continuously, one step at a time. Such a scheme allows multiple weighted destinations to be taken into account, in a dynamic fashion.

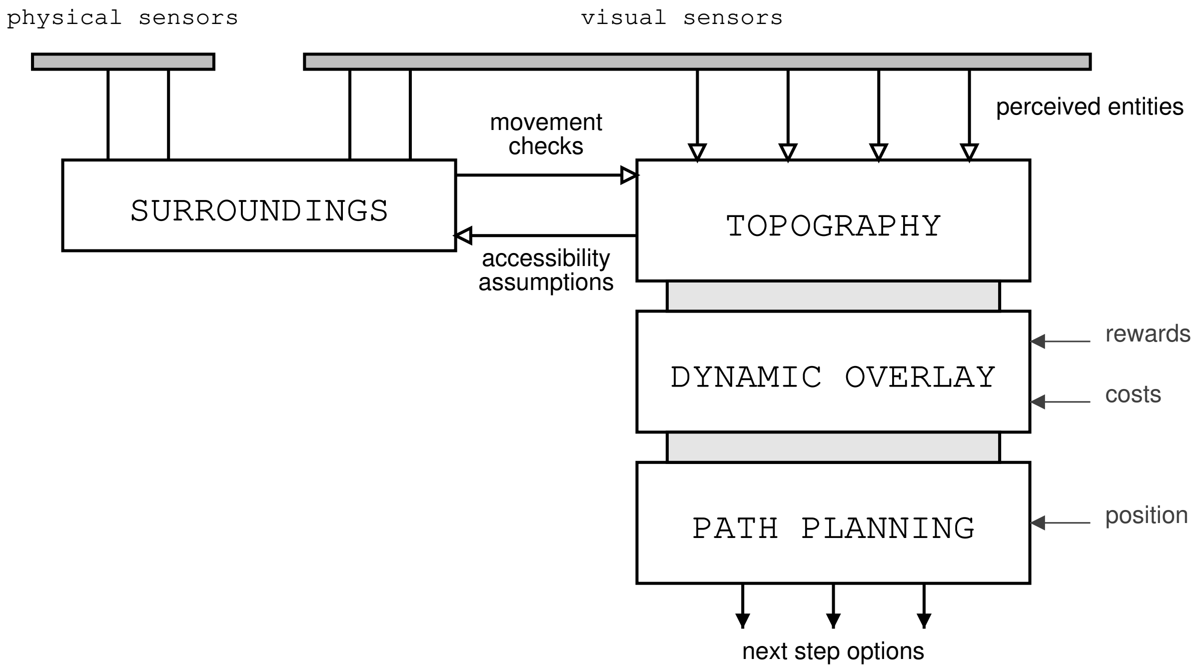


Figure 3.3: Higher-level cognitive layer that handles the planning of paths.

Properties of the terrain that the higher-level AI has decided upon (such as rewards and costs for specific areas) are stored alongside with the topography, but in a separate data structure called the `Overlay`, shown in *Figure 3.3*. This is used by the path planning to take into account the dynamic aspects of the problem.

The underlying process itself is a heuristic. A simple routine is applied repeatedly to improve the quality of approximation in the results. The greediness of the procedure provides very good initial estimates. A stochastically chosen seed allows interesting behaviours to arise, while guaranteeing that the optimal path will be found if enough computation power is dedicated to the task.

3.5.6 Arbitrator

The arbitrator is responsible for selecting the appropriate hosts, and fine-tuning their weight coefficient. It must also monitor their performance, in order to prevent them from succumbing to the parasites or inappropriate conditions.

The is responsible for selecting which of the higher-level behaviours are appropriate to fulfil requests (multiple are possible), select the appropriate parasites and give them targets (multiple are possible). These can be optionally blended together, in which case the arbitrator must compute their weights.

A finite state machine proves sufficient to handle all these requirements, as each situation can be preempted and handled with relatively flexible commands.

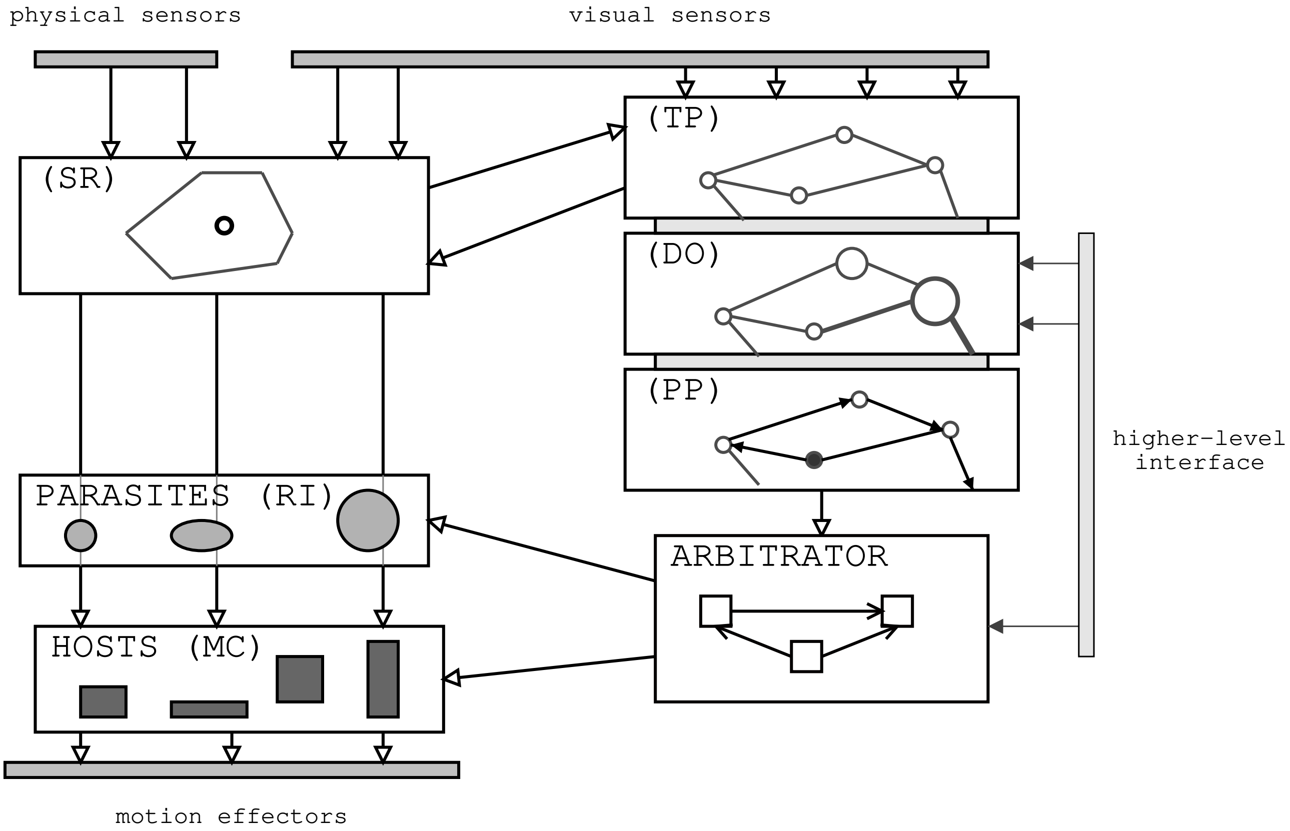


Figure 3.4: Simplified diagram of the entire system.

Chapter 4

Framework

"Science is organized knowledge." —Herbert Spencer

The task of implementing and developing the architecture can be assisted and also sped up by a common set of tools and methodologies, which can also improve the final result. This chapter aims to regroup and describe the common AI techniques and ideas used in our experimentation. Then the core of the framework will be explained. A description of the environment will be provided, along with a description of its interaction with animats.

Readers with theoretical and practical experience of all these fields, methodologies and software explained below may wish to move on to the next chapter tackling the first component of the navigation system itself.

4.1 Background

This first section intends to describe techniques of *Artificial Intelligence* that were used within this project, and vindicate our approach to them.

4.1.1 Evolutionary Algorithms

Overview

The field of *Evolutionary Algorithms* is inspired by Darwin's theory of evolution (Holland, 1992; Mitchell, 1998; Goldberg, 1989). Various different approaches exist, ranging from *Evolution Strategies* (ES) to *Genetic Algorithms* (GA). While initially isolated research groups, they are merging slowly; ideas are being borrowed and implanted across old boundaries. Such techniques can be used for artificial life simulations, where survival of the fittest is a fundamental issue, but also implicitly in optimisation problems.

Motivation

The main benefit for using GA in this context lies in unsupervised learning. They allow us to train other components based on objective fitness functions. This allows designers to tweak the behaviours using high-level parameters, assuming these have been incorporated into fitness scripts. Supervised processes on the other hand are often too restrictive with regards to possible solutions, whereas self-adapting solutions often have too little control.

Due to their generality, GA also allow various parameters to be optimised, no matter what the format or representation of the problem.

Description

The rest of our learning schemes rely predominantly on *GA*. It is essential to understand their generic template before proceeding with specific implementation details.

1) Initialisation — The population needs to be created first; this is a group of genotypes that represent possible solutions to the problem. Using random values for initialisation allows very good coverage of the initial search space. Each of these solutions is evaluated on the problem, and their fitness is noted.

2) Selection — One or two parent individuals need to be chosen from this population (in asexual or sexual breeding respectively). This usually involves processing parts of the population, and applying a selection scheme to pick out fit individuals. Commonly used approaches include roulette wheel selection, tournament selection, and best of group selection. Such elitism allows the quality of the population to improve over time.

3) Crossover — When two parents are selected, their genetic code is merged together to create an offspring. In essence, it's a matter of determining whether an allele should be set from the mother or the father, gene by gene. Standard genetic operators such as unary crossover (the mother genes are copied until one random point in the genotype, then the father's), binary crossover (the father's alleles are only copied between two random points), uniform crossover (each gene has a x probability of being copied from either parent). Since these operators are usually symmetric, it is customary to generate two offspring, dubbed son and daughter; they have genetically opposite genetic code.

4) Mutation — Once the crossover operator has been applied, a bit more variation can be given to the offspring by mutation. Select genes are chosen, usually at a low probability, and have their value mutated. Depending on the type of gene, this can happen in various ways; bits are flipped for binary genes, random values can be added to integer genes, and floating-point genes are subject to Gaussian distributions. This brings more genetic diversity to the population, preventing stagnation and improving the coverage of the search space.

5) Evaluation — Once the offspring have been created, they need to be tested against the problem. This is known as the evaluation procedure, whose goal it is to determine the fitness of a solution at a particular task (as performed in the initialisation stage). Once this fitness has been determined, it may be normalised; window and rank based techniques are common. This maintains pressure for survival on the best solutions, and increases the chances of poor solutions.

6) Replacement — Given the normalised fitness, the candidate offspring can optionally be reinserted into the population. Conceptually, the process can be either steady state (where new populations are created each generation), or incremental (where individuals are inserted into a persistent population). Both these cases can be handled implicitly by a replacement function. In practice, it can replace the parents, the worst individuals, or based on other randomised metrics.

Typically, each stage of the evolutionary process takes parameters; how these parameters are initially chosen and change during the evolution naturally has a huge impact on the number of epochs needed for convergence.

Capabilities

While problem-specific solutions often perform better, *Evolutionary Algorithms* still are a reliable form of optimisation. In fact, they are often praised for the following benefits:

- Generality — Evolutionary solutions generally rely on a very abstract form of representation, known as a genotype. This allows the underlying genetic operators to also be very flexible, allowing virtually any problem to be optimised.

- Efficiency — By combining the benefits of both random search and local optimisation, algorithms can normally avoid local minima effectively. Near-optimal solutions can thereby be reached when good default parameters are chosen.

Issues

Despite what many people think, *GA* are no miracle solution. Numerous simple mistakes or assumptions will lead to sub-optimality, and other properties make them less desirable as software solutions.

- Representation — The ability to convert genotypes to phenotypes is relied upon by the genetic algorithm. It is often a non-trivial process, as many different approaches to modelling a genotype may exist in the first place. Likewise the conversion to a phenotype includes many possible techniques (explicit, parametric, growing approach). Nevertheless, this remains a crucial part of the evolutionary process, as it can directly impact the quality of the final solution.
- Coherence — When crossover and mutation operators are used on specific genomes, they may cause an inconsistency in the genetic code. This proves problematic when it needs mapping onto a phenome, as the conversion procedure will need to check for consistency, truncating and completing genetic code as it sees fit. Alternatively the conversion routines must work with inconsistent genomes. Though possible, this is a very tedious process.
- Genetic Drift — Since good genetic code is more likely to propagate throughout the population, the crossover operator can cause a specific allele to take over the gene pool. This has the negative effect of preventing diversity, and causing premature convergence.
- Stability — Since much of the underlying process of the genetic algorithm can be seen as a biologically driven random search, this causes issues with online learning. Performance stability cannot be guaranteed, nor can its improvement over time; one moment it may evaluate the fittest individual yet, while the next trial could see the second worse. This is something that should be taken into account.
- Fitness Function — The definition of the fitness function is surprisingly important. Ideally, it needs to be defined smoothly over the search space. GA tend to struggle with fractures in fitness space.

Precautions can be taken to prevent all these problems but the designer should be aware of them in order to understand them as they happen.

Approach

The implementation used throughout the project is based on a toroidal *GA*, similar to that described by Gruau (1994). The implementation itself is based on Champandard (2001). It has the following properties:

- Incremental — Individuals are reinserted directly into the existing population. Each stage does not have a new set of individuals; there is no concept of "generation".
- Localisation — The population is stored in a grid world, wrapping round in both dimensions in a doughnut-like fashion. Selection of individuals to mate is limited to the local neighbourhood of the offspring's location. This actively enforces localisation to prevent genetic drift; fitness islands, where the genetic code is similar, quickly appear in the population. Since these islands can only directly affect local individuals, the spreading of dominant alleles is thereby limited, but possible when appropriate.

Since we do not believe the problems tackled with GA are exceptionally complex in our case, the choice of a particular GA over another has little motivation or practical benefit. Many properties of the chosen GA are interestingly very close to the *Artificial Life* paradigm (our initial research did indeed lean towards such multi-agent evolutionary systems).

However, we include the following enhancements; some deal with issues to improve performance, while others facilitate the implementation:

- Candidate Selection Scheme — A stochastic walk from a randomly selected grid coordinate is used to find the worse individual in a local neighbourhood. Its location is chosen for the next candidate offspring. Such a greedy candidate selection policy tends to increase global fitness rapidly.

- Elitist Replacement Policy — When the candidate has been evaluated, it is only re-inserted into the population if its fitness is higher (or randomly with low probability). This also tends to improve the quality of the solutions in the early stages.

- Phenotype Specific Operations — Due to the tremendous inconvenience involved in converting a generic genotype to a phenotype, we apply our genetic operators directly to the data-structure of the phenotype. Without any major drawbacks, we can also include problem specific operators that improve the quality of the evolution by simplifying the search space.

4.1.2 Neural Networks

The field of neural networks is part of machine learning, inspired by a neuro-biological model of the human brain. They are based on artificial models of real neurons, which are more or less accurate; we are still somewhat uncertain about how neurons fully function.

Topology Types

With over five decades of research in the field, there are understandably many different types of ANN. Each has its own preferred domain of application, and selecting the most appropriate type already requires thorough consideration.

- Perceptron — This is the famous feed-forward layout, where input values propagate directly through the network towards the output connections (Rosenblatt, 1962). While initial models only consisted of a single layer, improvements in the early 1980s allowed multiple layers to be trained (D. E. Rumelhart, 1986).

- Hopfield Network — Named after its inventor (Hopfield, 1982), this network has a fully recurrent topology, such that all neurons are interconnected. The dynamics of such networks are reasonably complex, but understood. Training algorithms have only just been devised.

- Gasnet — This recent model is based on the discovery that gas can behave as a neurotransmitter (Husbands, 1998). The complexity of the neural model introduces time dependent results and plasticity, which can be of great benefit for robotic controllers.

- There are also many other neural models based on self-organisation.

Given any topology, each neuron can be simulated in turn by processing the inputs to generate an output. The neural network can thereby perform computations, which can be interpreted and exploited as appropriate.

Learning

The purpose of learning is to replace an initially random set of weights with a near optimal one. There are numerous ways of doing this, as you may expect.

- Back-Propagation Training — This is one of the original forms of training (Rumelhart et al., 1986), based on supervised learning. Before each training sample, the output error is percolated back through the layers, gradient descent is performed on the weights. Many enhancements have been made to the initial algorithm over the years (Riedmiller and Braun, 1993), aiming to improve performance and reliability.

- Evolutionary Algorithms — A relatively recent form of *NN* optimisation involves *GA*. They can be used to evolve all parameters, including weights and layer sizes. Both supervised an unsupervised learning can be done in this fashion. Typically, performance doesn't compare to back-propagation algorithms, though they can have many advantages in specific situations.

- Hybrid Learning — Most training and evolution algorithms mentioned have advantages in specific situations. Custom solutions have been devised that exploit these benefits, performing well in a wide variety of situations (Castillo et al., 2000).

The selection of a training algorithm is somewhat tied to the problem at hand, and can be simpler than the choice of a topology.

Capabilities

While neural networks are very flexible solutions, they are generally used for two purposes:

- Regression — This technique is used to smoothly control robotic limbs, based on information given by a collection of sensors.

- Pattern Recognition — In a high-level fashion, they can be used to sort input vectors into classes. This has numerous uses in industry, such as conveyor belt monitoring and face recognition.

- Function Approximation — Neural networks can be used to approximate complex functions, modelling them in a much simpler and more efficient fashion.

- Prediction — This is used to anticipate time series, by a certain amount of steps. The stock market is being monitored using this technique.

In these applications listed, interesting properties of neural networks are exploited. In general, these are the primary incentive for using such solutions.

- Generalisation — Due to the way the knowledge is internalised, a well trained Neural Network can be expected to perform well in situations that it has not encountered before. By taking especial care during the design of our model, generalisation can be almost guaranteed.

- Noise Tolerance — One of the many advantages of the neural network approach remains its ability to deal with noise effortlessly. This is often attributed to the internal neuro-fuzzy representation.

- Continuous Values — Despite originally being intended to work on boolean quantities, many practical applications have shown the potential of using continuous floating point values, for robotics controllers especially. This is an intrinsic advantage since deciding how to convert continuous values data to discrete ones can be an error prone process (but nonetheless crucial).

Issues

Like other seemingly miracle solutions in AI, Neural Networks reveal their own set of pitfalls when one looks beneath the surface.

- Local Minima — When learning the weights by gradient descent, some algorithm can settle in a sub-optimal configuration. This can be due to the back-propagation algorithm itself, or the way the inputs are modelled which directly shapes fitness space; it may not be possible to solve the problem with the given inputs.

- Expert Design — Often, devising a good model is an intricate process, with under- and over- fitting notably causing problems. While initially, design relied on knowledgeable specialists, more and more solutions allow reliable tuning of structural parameters.

- Modelling I/O — The modelling of the interface is equally important, and tends to be neglected. This procedure cannot yet be handled automatically, and causes an extra dependency on the expert. Such preparations take up a significant proportion of the development time.

Approach

To resolve these issues (at least partly), we take the following precautions — further described by Champandard (2002a). These relate mostly to treating the problem as a (neural network) design phase, whereby each decision

- Environment Modelling — The data on input is often in an inconvenient format for machine learning algorithms generally, or neural networks more specifically. A simple pre-processing phase can significantly enhance the quality of the training. This can include techniques such as normalisation, intentional addition of noise, or transformation of the values via a function, generating compound features as well as removing symmetry.
- Structural Choices — We opt for a feed-forward perceptron for its speed and simplicity. By using the previous output as an extra input we can also get sufficient feedback to model interesting behaviours. Due to bad scalability issues, we tend to limit our networks to small sizes, both in neuron and layer counts.
- Evolutionary Algorithms — Using the genetic algorithms described in the previous section, unsupervised learning is used to train the networks.

Motivation

There are numerous reasons for choosing an evolutionary artificial neural network based approach, most of which can be split into two categories. Firstly, it may be appropriate to justify the need for a machine-learning approach to the problem.

- Simplicity — Models based on explicit mathematical equations are already simple from a *"cut and paste"* perspective. Little effort is required to get the initial solution working. However, adjusting the parameters of the model is a tedious matter, which requires much more knowledge of the underlying process. A machine learning approach allows the learning algorithm to adapt to new parameters intrinsically. This black-box approach allows a good compromise between high-quality performance and simplicity of the tuning.
- Customisation — Since the machine learning algorithm is usually based on high-level parameters, defined in human-like terms, it provides a much simpler way of obtaining the desired emergent behaviour. Additional fitness components can be set-up by the programmer, forcing the agent to learn to cope with these additional constraints while solving the collision avoidance problem at hand.

Secondly, the Neural Networks have many desirable properties that influence decisions to include them:

- Generalisation — In creating our navigation system, there will not be an infinitely long training procedure! This implies the need for such a high-level form of learning, allowing a finite number of well-chosen scenarios to do the trick.
- Noise Handling — In many aspects of the system, the continuous values passed around will be subject to imprecision (from internal communication to sensory data).
- Efficiency — When tackling a problem like navigation, fortunate enough to be easily optimisable, one can expect a reasonable size of the final *NN*. This not only implies fast learning, but also that its simulation will be extremely efficient (especially when implemented correctly). For animat simulations this is crucial since the frequency of the responses greatly affects the outcome.

The theory reveals neural networks as a good candidate solution for some navigation problems, but only experimentation will confirm this.

4.2 Environment

The world used to develop the navigation system is virtual 3D. It proves challenging due to its not realistic nature.

Figure 4.1: Features of a complex game environment. From left to right: ladders, stairs, bridges, doors, elevators and ledges.

4.2.1 Overview

While some believe computer games are ideal for developing human-level AI, they seem to forget that such games are generally straightforward (like deathmatch first-person shooters focused upon). Due to the high-pace of the game, a very reactive opponent — admittedly with convincing behaviours — would often perform well enough not to be noticed.[1] The fact that teenagers can pick this up in a few minutes, or that game developers can muster up their AI in a few months before shipping is a testimony to this fact.

Instead, our interest lies in the environments offered by modern game engines. Providing an excellent platform for development, this commercial quality software is all too often neglected by researchers. Our work uses the `Quake 2` engine, recently made open source (id Software, 2001). The custom AI itself can be inserted into the game DLL, which handles all the game logic, without recompilation of the main executable.

4.2.2 Description

These polished 3D simulators, along with their interesting indoor design provide an ideal test-bed for a navigation system. Without going into too much detail, such environments can be summarised as follows:

- <u>Terrain</u> — This composes the major part of the environment, composed of indoor buildings and outdoor landscapes. Conceptually speaking, this can be further split into two parts:
  - *Structure*: The important part of the terrain that the animat can collide with. This includes floors, walls, fences, big trees, buildings...
  - *Detail*: The second part of the terrain that is purely cosmetic; they have very little effect on the movement, but may be used for other purposes (i.e. as landmarks). This includes plants, pavements, grass...

  This distinction will be made obvious by the interface to the environment. Structural queries will reveal the rough shape of the terrain, while detail will be accessible as visible entities.

- <u>Items</u> — The terrain is populated with objects; some cannot be picked up by the animat, while others have a particular benefit. For example, there are health-packs, armour jackets, ammunition, and power-ups...

Additionally, there's an assortment of contraptions scattered around the terrain that can each affect the player's movement. These challenges for the navigation system are:

[1]In fact, the other bots evaluated somewhat rely on this; the spectator modes are crippled enough not to allow objective observation of the bot.

- Jump-pads — Small devices placed on the floor that can project people through the air towards another point. The pad must be walked upon for it to be triggered, and the fall can be controlled actively.

- Teleporters — Either implanted into walls or as standalone devices, teleporters carry the player to a known location (deterministically). These also need to be walked into for them to activate.

- Platforms — Rectangular elevators that can carry the player along them. Some lifts just move up and down, while others have complex paths (dubbed trains) that take skill to board and descend.

- Ladders — As bits of walls that the player can grab hold of, ladders have a more generic definition than per usual. They could be standard ladders, fences, or big trees. Sadly, ladders are not handled as entities; a special kind of structural query reveals the nature of the wall.

4.2.3 Development

The use of a robust game engine accelerated the development greatly. A combination of passive observation and interaction with the bots also simplified the debugging.

1) Let the animats learn in accelerated console mode, without the graphics.

2) Check on the learning by connecting a spectator client to the server.

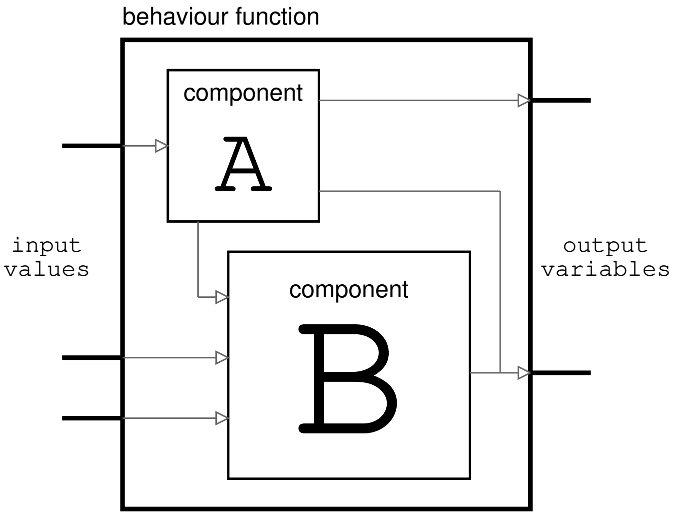
3) Freeze the animats when a satisfactory level has been reached.

4) Connect a player client to the server, and interact with the animats.

Such a typical session is the base of our development procedure.

4.3 Framework

This section underlines the core of our implementation, as well as the tools used to develop the architecture.



to

Figure 4.2: Flow chart of a flexible behaviour; script function can handle the components as appropriate.

4.3.1 Skeleton

The skeleton of the architecture is defined in C++. Using object-oriented programming, the underlying behaviour of each module must be specified in a configurable fashion. The interaction between these different modules must also be hardcoded in C++.

This task is pretty straightforward once the architecture itself has been designed; it's simply a matter of implementing the control algorithm and hardwiring inter-module communications. It provides an extremely efficient base, which can be entirely customised during the next step of the development.

4.3.2 Flexible Behaviours

These are abstract classes that can be customised to perform any task, at virtually any level of the architecture. However, they are typically assigned to learn specific behaviours — whence comes their name.

A behaviour is essentially a function in the dynamic scripting environment. Python was recruited for this task since it provides all the power offered by a modern programming language, and virtually any standard control technique can be implemented. In order to simplify the process, it can optionally rely on "flexible components." these are generic objects that provide more sophisticated AI techniques — like evolutionary neural networks. Components can be instantiated during the creation of the behaviour, and used freely within the behaviour function. The scripting language provides the cement which connects the components.

The underlying learning process of a component is based on a NN, whose properties are set inside an hand-editable `XML` file. The genetic algorithm can also be tweaked using this same file. The fitness function itself is stored inside a `Python` script. This is not only used to perform pre-processing, but also post-processing and the actual evaluation. The control variables it returns are then directly used to perform the task at hand. The scripts written are able to take into account high-level parameters set inside the configuration file, allowing non-programmers to simply tweak these values.

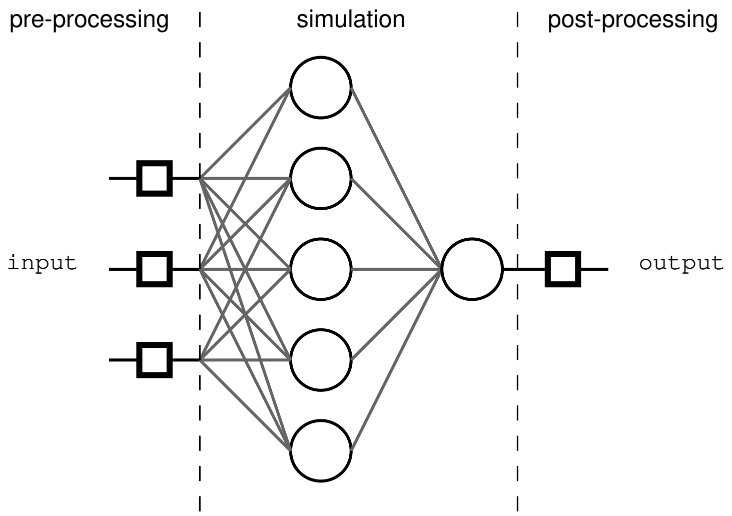


Figure 4.3: Diagram of a typical flexible component, a neural network wrapped inside a custom script

Once connected to the rest of the skeleton, the "flexible behaviours" can become modules themselves. There are two phases involved to insert flexible behaviours into the navigation system; one of them corresponds to its implementation within the skeleton, while the other instantiates the generic behaviours by providing a configuration and a control script. The behaviours are designed on a per-individual basis, but multiple such instantiations are possible to accomplish varying navigational behaviours as part of the same module in the system.

Inside each python script, variables have naming conventions. The prefix `"r_"` indicates this variable is returned by the script to the system, `"g_"` indicates a global variable, while `"s_"` is a state variable that is linked with the current estimated state of the animat.

4.3.3 Animat Architecture

All the contact with the environment is handled by the open-source `Fear` library — designed and implemented alongside this research. `FEAR` is a **F**lexible **E**mbodied **A**nimat a**R**chitecture, which defines generic sensory and effectory interfaces between an abstract agent and any virtual environment. The interested reader should read the design documentation on the official web-site (Fear, 2002). Though detailing the internal workings of this framework are not entirely relevant for this dissertation, some key concepts will have an important impact on the navigation system.

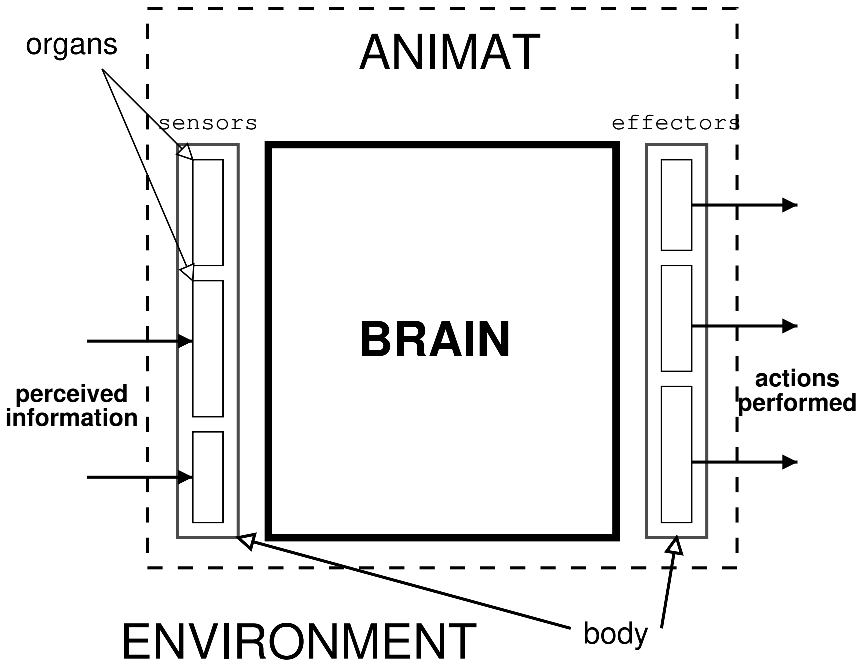
In practice, we need to look at the two extensions that are used by our navigation system. The first two are considered sensors: `physical` and `visual`, while the third `motion` effector is optional — since the system can simply return suggestions and not execute anything.

- Physical Sensor — These provide basic information about the environment in immediate contact with the animat. This includes the type of medium *(air,ground,water)*, the presence of obstacles with collision feedback, and odometric queries to track movement.

- Visual Sensor — The vision interface allows items and other humans to be perceived in a symbolic fashion. Only the type of each entity is returned, and they do not broadcast their own unique ID — as commonly assumed. Additionally, the structure of the environment can also be checked visually; this is done by explicit requests rather than passing all the raw distance data back. This allows a more efficient, scalable management of the information, but also implies that the visual sensor can abstract out the complex processing (such as checking for visible obstacles along a path). This is something humans can rely on thanks to millions of years of evolution behind them.

- Motion Effector — Movement is handled via this class specification. The agent can chose to `move` in any direction (speed restrictions are applied when moving backwards), and independently `turn` its body for looking in different directions. Additionally, less trivial actions such as `jump` and `crouch` are provided.

For more details, the complete code listing for the specification of these interfaces is given in *Appendix B*.



to

Figure 4.4: An embodied animat using its sensory and effectory organs to interact within its environment (reproduced with permission from the FEAR team).

Chapter 5

Hosts: Motion Controllers

"Obstacles are like wild animals. They are cowards but they will bluff you if they can. If they see you are afraid of them... they are liable to spring upon you; but if you look them squarely in the eye, they will slink out of sight." —*Orison Swett Marden*

The motion control (MC) is a low-level module providing a flexible framework to define mostly reactive activities. By putting this scaffolding to use, specific applicable behaviours mapping sensory input streams onto action sequences can be crafted with minimal overhead (e.g., for collision prevention or wall hugging). This allows navigational entities to be produced within the context of our environment; we call these "hosts", as they are primitive creatures capable of straightforward movement.

First, a generic description of requirements of motion controllers is provided, as expected from a somewhat idealised solution. Then, select existing approaches will be analysed. Based on that, a specification for this module will be defined and the technology framework will be established. Finally, the specific parameters used for our obstacle avoidance and wall following behaviours will be given and explained.

5.1 Requirements

This section tries to outline implementation independent requirements from similar control components. However, these requirements are not always explicit aims for motion controllers — though they can nonetheless be evaluated with regards to these criteria. Fortunately, they are also the exact features our implementation is striving for.

- Real-time Response — Since the MC is responsible for carrying out movement, the responsiveness of the system depends on the speed of its execution. This has consequences on the sensitivity to higher-level commands, as well as the ability to handle dynamic conditions. A timely response is thereby required, in a fashion reminiscent of *Real-Time Systems*. Specifically, one with a soft deadline where the module's output remains important as time goes by, but the benefit of a late action decreases with respect to time.

- Human-Like Motion — By handling the lowest-level of motion, this module is an important (but not exclusive) aspect of realistic overall behaviour. While this is not always an immediate requirement, related properties such as smoothness or laziness are coveted, as they often translate into efficiency.

- Reliability — When the rest of the system is built on top of a single component, it needs to guarantee that nothing will go wrong. This implies providing satisfactory movement in all situations, not just the general cases. In the worse case, the module should be capable of *"cognisant failure"*, whereby it can identify problematic situations, allowing further action to be taken elsewhere in the system.

- Noise Tolerance — As discussed in previous chapters, noise can and should be expected from the sensor values. Together with this, many higher-level modules in the system may interfere with the results, thereby potentially causing some unexpected changes. Designing the MC around this fact will prevent a majority of problems related to this.

Despite appearing trivial, this task of motion control nevertheless remains the most essential component of the entire system. Other behaviours, which could be classed as higher-level, will rely entirely on this layer to actually perform the motion.

5.2 Existing Approaches

This section looks into options for implementing the MC, based on existing work. Solutions will be evaluated with respect to the requirements listed above, as well as criteria given in *Chapter 3* considering the design policy.

5.2.1 Grid Based

Initial investigations in reactive behaviours and virtual obstacle avoidance always reveal a plethora of grid-based solutions (c.f. the review of *Chapter 2*). These either use simple reactive rules (both expert designed and learnt) to determine the next step with a minimal amount of reasoning, or more deliberative approaches based on local search.

While these are appropriate for simple worlds, complex environments are ill-suited to being discretised; our requirements do not limit themselves to such regular environments. Embodiment does not lend itself well to the paradigm either; the animat's sensors would require exhaustive probabilistic processing to generate a usable grid representation. Also, the cardinal actions implied are also inappropriate by their coarseness (especially unrealistic).

While such searches may be plausible at a higher-level of the architecture, they do not provide the realism, reliability and efficiency required for low-level control.

5.2.2 Flocking Behaviour

Craig Reynolds' work (Reynolds, 1987, 1999, 2000) is generally associated with artificial life and collective intelligence, but his creatures are defined with primitive low-level actions — emergently creating realistic group behaviours. Obstacle avoidance happens to be one of these activities, achieved by using obstacle distance sensors, which return the exact position of obstacles at a given angle. Some of his creatures have fixed distance sensors (of layout angles $\{-45°, 0°, +45°\}$ and limited distance ranges), while others use a random scan within their field of view. It should be noted that, due to the simulated nature of these sensors, they have an extremely precise angular setting as well as the distances returned.

Mathematical equations defined by the author take into account these measurements, updating the speed and turning rates. The reactive nature of this model implies a very efficient implementation, capable of real-time response. This explicit approach seems to work surprisingly well in practice, with the creatures only colliding with obstacles (in dense environments) every 500 steps of the simulation.

5.2.3 Fuzzy Logic

Some projects use fuzzy logic for obstacle avoidance behaviours; a keen reader should refer to an in-depth review by Saffiotti (1997) for more information. Essentially, fuzzy variables are defined to represent the presence of obstacles on the left, right and front. The actions are also made fuzzy; turn/brake. An expert system of fuzzy logic equations (fuzzy program) can use these variables to define what action should be taken. Evolved fuzzy classifier systems are also increasingly popular, since they allow the learning of the behaviours.

In practice, this provides very smooth movement for obstacle avoidance. Efficiency is maintained as long as the fuzzy programs stay simple and the classifiers remain small. Fuzzy logic is ideally suited to reliably handing noise too. More interesting behaviours can be defined additionally to this using very similar concepts, such as seeking and fleeing — but these will be reviewed in the next chapter, as appropriate.

5.2.4 Feature Maps

Recent work in imitation learning has tackled wall following behaviours using a self-organising feature map (Marom and Hayes, 2001). The sensory sonar data is used as the input to the SOFM. A first approach associates the output with motor schema by imitation, a correspondence learnt during a separate phase (the results can then be exploited later). Another method uses a reinforcement process to learn the mapping to continuous values in action space. This is learnt online, and the action selection progressively varies from exploration to exploitation.

Due to the way the motor schemas are exploited in the first case (without interpolation), the motion learnt is not especially smooth, though it does prove generally quite reliable thanks to the extensive imitation phase. In the second case, low frequency of action selection produces smooth results, but a sufficiently long training process would also theoretically prove adequate (this was not the aim of the project).

5.2.5 Summary

Obstacle avoidance is a simple task, and the motion control should be kept as simple as possible. Self-organising feature maps are especially suited to problems requiring tight control. Obstacle avoidance does not exploit such sophisticated adaptation in input/output space; the overhead of SOFM is not required. Grid based approaches do not fit in with the requirements of complex environments. The flocking behaviours have the disadvantage of being based on explicit equations, which need to be understood and tweaked to craft desired behaviours. Fuzzy logic based approaches have a similar problem, as well as requiring a dedicated fuzzy interpreter.

Experience from these implementations can be drawn upon to define both an interface for the MC and its operating process.

5.3 Specification

This section focuses on the definition of the interface. While this is a crucial aspect of its integration into the system, it also allows us to sketch a rough outline of the expected underlying functionality.

Although each agent is modelled as a 3D entity with a position/velocity tuple to match, in most media movement generally remains constrained to a 2D plane due to gravity. The local environment is represented relatively to this state, so inputs and actions are also therefore relative.

5.3.1 Input

Due to the simulated nature of the project, the information needs to be actively acquired, unlike real robots for which the sensory data simply needs sampling and processing. Since the acquisition of this information can be relatively costly, the dependency of the MC should be kept to a minimum. This sensory data can be indiscriminately fetched from the environment directly (the world acts as input), or indirectly using a simple cache of the obstacle data (a persistent internal representation). Conceptually speaking, this does not affect the motion control itself.

Our model samples the surroundings in a very coarse fashion, using the concept of a virtual whisker. These return the distance of an obstacle within range, at a specific angle on a horizontal plane around the animat. This approach intuitively seems to present a high-risk of perceptual shortcomings. Specifically, obstacles between whiskers may be invisible, potentially causing the animat to ignore them.[1] However this potential hazard, due to the persistence of the models, the continuity of the sampling, and the noise artificially inserted to change the angles of the whiskers can be prevented with minimal effort.

[1] This idiosyncrasy happens in real robots too, notably when their sonar sensors fail to detect chair legs.

This limited sampling also has many advantages; such discrete data is a significant simplification of the complete environment. Although, this generalisation is usually quite accurate since the local surroundings contain the most relevant information (due to the reactive nature of the task). This implies that the function mapping sensory input onto motion is a much simpler process — in terms of both search space size and intricacy. The levels of generalisation obtainable by the solutions have a beneficial impact on learning times.

While initially whiskers only allowed the detection of solid objects, this didn't allow very human-like behaviours; animats would fall off ledges and drop into slime pits quite happily. Consequently, they were extended to also detect liquids and gaps (handled in the same fashion as a wall, remaining horizontal). This is not necessarily biologically accurate, but generates more realistic movement.

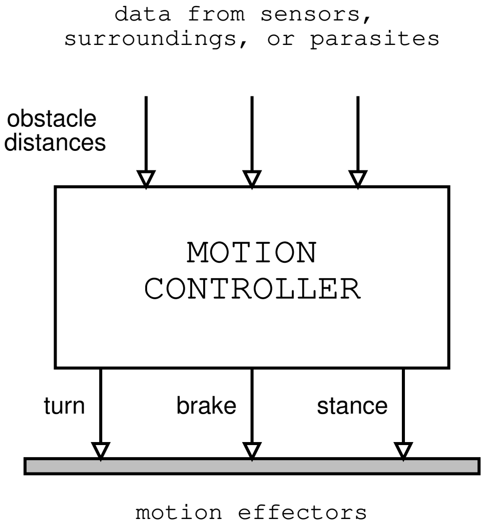


Figure 5.1: The motion controller's interface with the rest of the system.

The knowledgeable reader will notice the similarities of this input scheme with the highly acclaimed work of *Craig Reynolds*, though the project never actively set out to model it. In essence, parallels can be drawn with concept of the whisker as well as the angular noise intrinsic to these sensors. However, our model provides a more flexible motion controller as it can detect various other obstructions than solid obstacles (ledges as well as walls). Additionally, the flexible behaviours provide the ability to evolve the layout of the whiskers, which potentially removes the dependency on human design.

5.3.2 Output

The outputs of the motion control are obviously actions that engender movement! Specifically, these can be continuous cardinal controls `(forward,backward,left,right)`, all of which are performed relatively to the current position. However, this can be simplified further into more intuitive parameters that are simpler to learn. In practice, the two commands `turn` and `brake` are sufficient, and prove to be more human-like when the direction of travel is distinguished from the orientation.

In fact, this dissociation was one of the key concepts for obtaining realism, compared to other human players. It produces very interesting strafing behaviours, complex manoeuvres such as fleeing while looking back towards the predator, and provides a platform for rather more stupid mistakes like running into walls by negligence.

Additional parameters for the movement involve specific stances such as `jumping` and `ducking`, respectively to get over low obstacles and underneath constrained ceilings. Each of the commands are taken into account by the effectors and applied to the physical state of the animat using the physical rules of the world. The next frame, it can then query its internal sensors to check what actually happened, and update its state accordingly.

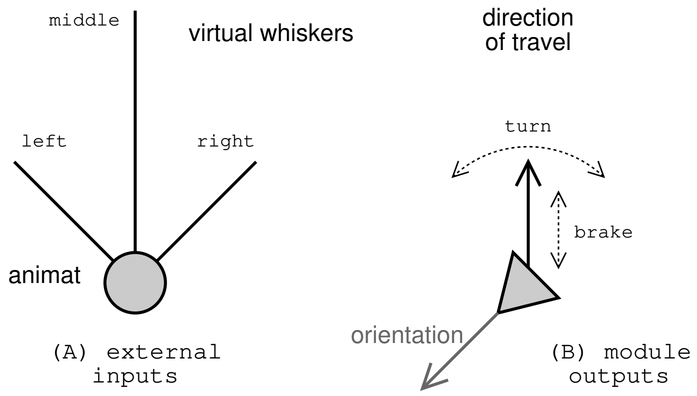


Figure 5.2: Graphical interpretation of the module's I/O interface.

5.4 Technology

Now the skeleton of the module has been defined, the underlying tools that will craft the behaviours need to be described. Specifically, the solution presented is based on Evolutionary Artificial Neural Networks (EANN), an approach which is becoming increasingly popular for robotics controllers. Aside from the general benefits listed in the *Background Chapter*, these also tackle the following issues:

1) Real-time Capacity — Due to the efficiency of simulating simple neural networks, these can easily achieve even rigid real-time deadlines. This is important for MC to obtain overall responsiveness.

2) Smooth Control — The continuous outputs, the neuro-fuzzy internal representation, and the use of relative turns usually causes the output movement to be smooth. This provides surprisingly human-like behaviours.

Generally speaking, the task of avoiding small obstacles can be easily tackled without using complex deliberative algorithms — as shown by most approaches in the brief review of *Section 5.2*. By working on a purely reactive basis in taking immediate sensor data as input, achieving the soft-deadlines becomes a more likely outcome.

Flexible behaviours are used to handle this, as described in *Section 4.3.2*. Navigational "hosts" can be created using any method, though we use the EANN approach. Despite them having no internal restrictions, there are common issues to be taken into account during their development.

- Fitness Feedback — When flexible components are used, their performance needs to be evaluated to assist the learning. In this case, this can be done on a purely reactive basis; collision with walls are punished and large obstacle distances are rewarded.
- Realism — The key aspects of the reward function involve realism components. These are parameters that quantify human-like characteristics *(e.g., laziness, persistence, smoothness)*. In practice, they prove important to tune the desired behaviour.
- Symmetry — Many configurations expose some kind of balance; for motion control the animat turning left is similar to its turning right. Instead of expressing this as a requirement in the fitness function (which constricts the solution and complicates the problem), it can be actively enforced by flipping the input distances and negating the output yaw. This proves to generate better quality animats.
- Incremental Learning — The evolution often needs to take place in stages to be optimal, or even correct altogether. Though the framework supports parallel evolution, components can be learnt individually then frozen.

The ability to take into account these factors will increase the overall quality of the movement, as well as speed up the development and learning process.

5.5 Model Design

This section explains how our technology was used in practice to create specific hosts within the module's skeleton. These provide fundamental behaviours which will be used regularly by the system.

5.5.1 Obstacle Avoidance

Since this problem is relatively simple, two small neural networks are used. The behaviour is composed of two primary components: `steer` and `brake`, each controlling one output variable. They are both evolutionary NN learnt preferably beforehand, but within the environment itself.

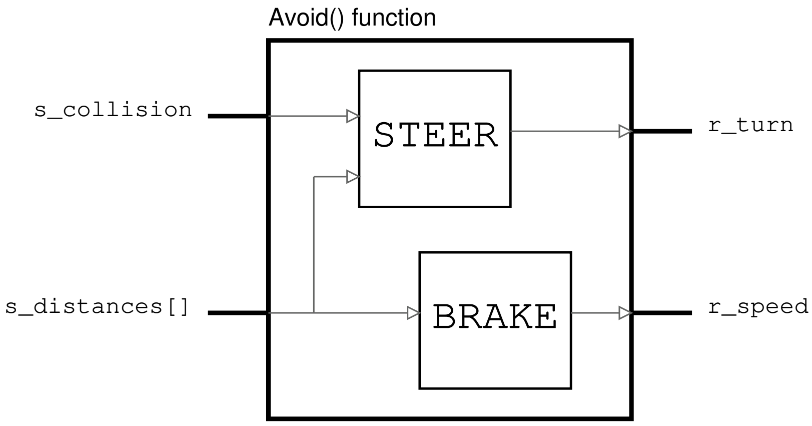


Figure 5.3: Diagram of the obstacle avoidance function.

Steer

The steering NN component conceptually deals with three pieces of information: the normalised sensory data (which may constitute more than one actual input), one recurrent connection and a collision indicator. The value of the collision indicator increases linearly with the collision time, to give the network a sense of urgency. The clamped output corresponds directly to the turn rate required.

Brake

The braking NN component is given similar inputs: one feedback connection, and the distance sensors. The output corresponds to the amount of slow-down required. The model attempts to minimise the use of the brake as overall performance can decrease, and human players rarely slow down anyway.

Procedure

These are evolved incrementally, partly to obtain a high-quality solution, but also to prevent the solution exploiting loopholes in the fitness functions.[2] First the steering is evolved with speed locked on full. It is rewarded for avoiding obstacles efficiently and realistically. Only then is the braking component included, rewarding for preventing collision when the occasion arises.

Parameters

The properties of the obstacle avoidance can be configured by XML in a straightforward fashion. The full listing is given in *Appendix A.1*, and should be consulted for practical insights into the engineering of the behaviours.

[2] A good way of avoiding collision is not to move forward at all!

The obstacle avoidance behaviour is a function in the scripting environment, defined from high-level parameters. The values are used as weights in the fitness function, indicating how much reward or punishment is given for collisions, as well as oscillation, laziness, urgency, and persistence of the movement.

Post processing can also be configured, and indicates that the input is symmetric, or if the output values must be clamped.

The external whiskers of the motion controller are specified in the `"layout"` section. Each sensor can be described with specific angular and distance settings, or left to be optimised by evolution. This is done by setting either attribute to `"evolve"`. Surprisingly, a wide range of solutions evolve successfully (ranging from 7 scattered sensors, to one single whisker practically straight), but no distinct trends arise. In practice, forcing three whiskers on the left, right and middle allow good behaviours to evolve. Their length is limited to 5 steps only to reduce the overhead required for detecting the obstacles.

The neural network used is a feed forward perceptron with simply 5 neurons in the hidden layer. Evolutionary parameters are also provided, which allow the network to learn the behaviour according to the fitness function.

Script

This `Python` procedure takes into account the high-level parameters and incorporates them into the processing and the evaluation of the fitness. It can be found alongside the XML file in *Appendix A.1*.

Symmetry removal is performed to simplify the search space, and to guarantee that the animat is just as capable of turning either way. This is done by making sure that the right obstacle is further away than the left, flipping and negating values appropriately. This halves the search space by mapping one side onto another.

The main part of the behaviour function builds up the arrays acting as inputs for the neural networks, and calls the simulation. Once the result has been retrieved, it is post-processed. This consists of simple clamping, and the second part of the symmetry removal.

This next part evaluates the fitness of the behaviour. All fitness components are scaled according to parameters in the XML file, which are denoted with the `"x_"` prefix. The penalty increases linearly with each colliding time-step. Variations from the previous output are punished proportionally, as well as large changes in orientation — especially when no obstacles are present. Oscillatory movement also deserves a constant penalty.

Results

The resulting animats perform very reliable obstacle avoidance. When not disturbed by obstacles in front of them, they keep moving straight. When obstacles arise, they swerve away relatively rapidly. Externally, this looks like they bounce off the walls.

This is due to the relatively short length of the whiskers. With slightly broader sensing horizon, they are capable of anticipating obstacles more. Also, punishment can be increased for such sharp turns, but this proves to generate unreliable behaviours in tight corners where the animat occasionally gets stuck.

5.5.2 Wall Following

Parameters

Once again, a node in the XML document defines the wall following behaviour in a simple fashion. *Appendix A.2* lists the file, and the interested reader should consult it for further information.

The wall following behaviour is another function in the scripting environment. The high-level parameters use specify the desired outcome. In practice, these reward the presence and proximity of a wall, but punish large turns and collisions.

The wall following behaviour uses two whiskers only. One relatively long whisker on the side checks for the wall, while one smaller frontal one checks for potential collisions while moving forward.

The neural network used is a feed forward perceptron with 12 neurons in the hidden layer. This empirically proved to generate satisfactory behaviours.

Script

A function defined in `Python`, listed in *Appendix A.1*, is used to apply these parameters and synthesize a behaviour.

The task is somewhat simpler than for obstacle avoidance. The sensory values are given immediately to the neural network, and clamped as a post-process. Flipping values for following walls to the left or the right is determined by the arbitrator.

The remaining part of the function performs fitness evaluation, simply checking the obstacles and extracting the high-level properties from them. These are weighted, and added to the total fitness which is reported back.

Results

Again, the behaviour is surprisingly robust, except in one situation. When the wall curves away from the bot especially quickly (like a narrow wall where 180° turn is required), the bot can overshoot and end up spinning upon himself repeatedly if the behaviour is not disengaged.

This can be remedied by adding a proactive breaking component, and training the bot on such layouts more often; they are very rare in practice, so an average training session does not get the opportunity to fail there.

Chapter 6

Parasites: Reactive Behaviours

"Consider the difference between riding a horse and driving an automobile. A horse will not run into a telephone pole at high speed. If you fall asleep in the saddle, a horse will continue to follow the path on its own. If you have two horses but only one rider, you simply have one horse follow the other." —Jonathan Connell

In the previous chapter, fully functional entities capable of purely reactive motion were crafted, using a mapping of sensory information directly onto the effectors. These fundamental behaviours were defined in terms of obstacles only, which made them generally reliable — in a way you'd expect animals not to repeatedly collide with walls.

However, the resulting movement still lacks the sophistication of human navigation, which is understandably more than just reliable obstacle avoidance. The first step to extending this lower layer will be to blend more elaborate behaviours to the movement, while still remaining in the domain of reactive motion. This includes activities such as seeking and fleeing, providing a handle for incorporating arbitrarily complex higher-level tasks into the architecture.

In essence, the problem resides in combining behaviours of varying reactive degrees: avoid with seek/flee for example. This can be done by selecting them *(arbitration, overriding, voting)* or blending them *(interpolation, defuzzification)*, although the process needs to maximise the overall performance.

This chapter will first look at a generic set of criteria, which can be used to rate the suitability of a solution. Then existing approaches that could be used to solve this dilemma are considered, and we discuss their respective pitfalls. Thereafter, our "parasitic" approach will be described along with the biological parallels. Finally, specific instances like evading and pursuing will be demonstrated and explained.

6.1 Requirements

This section has many parallels with its counterpart in the motion control chapter *(Section 5.1)*, in both structure and content. It outlines generic requirements from solutions capable of integrating reactive behaviours together. These are criteria for their evaluation as well as goals for our implementation.

- Real-time — Such reactive behaviours need to be as alert as possible. A significant part of the system relies on them, and the dynamics of the environment often impose a responsive setup. If the overhead of combining activities together is too high, it completely negates the effort placed in crafting efficient behaviours in the first place.

- Realistic — Mixing activities is something humans do relatively well, and not always in a visibly obvious fashion. This is another important part for a system explicitly striving for natural motion, but also implicitly for solutions desiring smoothness (i.e., for safety issues).

- Reliability — Splitting navigation into sub-behaviours makes sense from a development and debugging point of view, though problems may arise when assembling these activities back together again. Incorrect triggering of the behaviours can lead to inappropriate or even inconsistent results. This process is, therefore, a key issue for reliability.

- Optimality — There are many different ways of integrating behaviours together, each of which can provide different results. These should match as closely as possible the wishes of the user.

- Simplicity — Designing and tweaking the interactions between the behaviours, as well as the selection process can be a time consuming process. The method should allow the engineer to spend the least time possible on the procedure.

Unlike the MC, this problem is not so trivial. However, it is equally important for the outcome of the system, as an increasing number of problems must be dealt with.

6.2 Existing Approaches

No matter which paradigm is used, the important concept is to retain a fully working motion-controlling layer. Naturally, there are observations and regularities to exploit; the trick is to find a way of doing this reliably, without neglecting either of the potentially diverging requirements.

6.2.1 Subsumption

As one of the most famous techniques for behaviour arbitration, subsumption finds its roots in navigation (Brooks, 1991). Fundamentally, described alongside a methodology advocating incremental design, it uses suppression and inhibition points to integrate new behaviours into the existing system. This is done by overriding the information flow at key points in the system. The result can be represented as a complex diagram, with behavioural components handling distributed control between the sensors and the actuators.

Over the years, this has been interpreted as a vertically layered approach. While the analogy is made in the original paper, where these layers are built in and linked together as the designer sees fit, nowadays the output of previous layers is entirely subsumed by the layer just above in the hierarchy. Such a disparity in practice makes it difficult to analyse the technique fairly.

Despite the attractiveness and popularity of the generic design methodology associated with Brooks' architecture, we believe that subsumption fundamentally has many flaws for human-like navigation — some more critical than others.

- Complexity — Adding an extra "layer" on the top of the system is not a trivial task, the challenge it poses increases as the number of underlying layers grows. Not to mention the fact that defining suppression and inhibition points is not a trivial task in the first place, requiring foresight when building lower layers. This requires time-consuming expert design, implying this is not a trivial procedure that could be easily automated.

- Unrealistic — Though rigid selection of existing behaviours may be satisfactory in many cases, the simulation of human-like navigation requires a little more refinement than just inhibiting connections and superseding others. In fact, the verb "subsume" in itself implies that existing behaviours are replaced, albeit enhanced. The blending and combination of behaviours is not something that a subsumption network could handle, which rules out a certain degree of smoothness and intricacy in the actions.

- Redundancy — While redundancy is a trick mother nature uses to avoid the side effects of noise, this is not always a desirable property of robotic systems (time costs, unreliability, computational overheads). However, the approach employed by recent subsumers promotes independency of the layers rather than complex intercommunication. Since activities often have intricate interactions, layers spend a lot of time determining if it's ok to subsume lower ones.

Despite subsumption not being suited to the problem at hand, it remains a very generic, flexible, conceptually simple methodology that can apply to a broad range of situations. It does not seem surprising that a custom tailored solution could outshine it.

6.2.2 Expert Systems

In a fashion similar to the grid based obstacle avoidance, further reactive behaviours can be incorporated into rule-based systems. These need to be manually engineered appropriately to maximize the expected reward. For example with fleeing while avoiding obstacles, steps in a direction further away from the target are given priority when obstructions are not present.

Similar approaches based on rather more deliberative approaches are also popular. Global searches like A* are performed to seek particular targets, while backtracking takes care of avoiding obstacles and dead ends.

Fuzzy logic has also been used to create expert behaviours with fuzzy programs. The engineer encodes the behaviour arbitration as rules, which are interpreted and defuzzified. This provides surprisingly smooth actions, even with simple programs.

6.2.3 Voting System

Like in a democracy, voting systems arbitrate behaviours by attempting to reach a compromise. Candidates are essentially actions, and the most popular is executed. This proves very successful when no conflicts are witnessed. More elaborate statistical algorithms for combining the votes are necessary, or introducing the concept of a veto.

6.2.4 Learning Systems

Using the user's specification of the desired outcome (in the form of a fitness function), learning approaches can be used. Reinforcement learning algorithms can find an optimal policy mapping states onto actions — given enough training time to explore the search space.

Classifier systems are also capable of learning, using genetic algorithms and combined classifiers. These are breed according to positive feedback from the environment. This allows combinations of behaviours, since there are few restrictions on the possibilities of a classifier.

Both approaches attempts to synthesise practical behaviours based on abstract rewards.

6.2.5 Summary

In brief, some solutions presented above are too computationally expensive to be feasible in a simple agent (A* or SOFM), while others have a certain complexity in design and do not provide a simple way of crafting the behaviours in complex worlds (expert systems and fuzzy equations). Additionally, select approaches have issues with modelling human-like behaviours (subsumption and classifiers), lacking the tools and flexibility to create a wide range of realistic motion.

6.3 Specification

In this section, the parasitic approach is described, revealing its simplicity and biological inspirations.

6.3.1 Foundations

Often, humans take for granted their ability to avoid obstacles, as this is performed somewhat subconsciously. This provides a natural base implicitly influenced by higher-level plans. To model the fact that the lower level is in control, and that further reactive behaviours are not necessary, the following functional observations are made:

1) There is no need for arbitration when there is only one output.

2) Integrating new reactive behaviours is trivial if all the other module interfaces remain unchanged.

Specifically, the following properties of navigation allow us to establish a reliable architecture.

- In a cluttered environment, fleeing and seeking always involves avoiding obstacles successfully, but in a particular fashion. These behaviours can be classed as a subset of the more primitive motions defined in the lower-layer.
- One can falsify the input values from the distance sensors to obtain the desired motion. This process can be easily applied more than once in a weighted fashion to blend chosen behaviours.
- When this bamboozling is done in a reasonable and controlled fashion, the impact on the performance of the lower-layer is minute, but sufficient.

These observations what can be taken into account to specify the interface.

6.3.2 Input / Output

Practically, this is achieved by intercepting the obstacle distances from the surroundings. These are continuous values that indicate the proximity of walls. While these are almost accurate estimates on the input, they become rigged on the output in order to influence the movement of the host in charge of motion control.

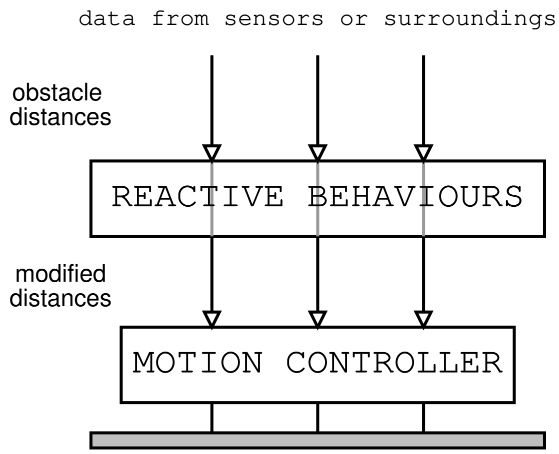


Figure 6.1: The interface of the reactive behaviours with the rest of the system.

Certainly, there are some strong similarities with a horizontally layered model; layers pass information to each other in direction of the output. However, there are some key differences. While traditional models tend to use different levels of abstract representations away from the input, the parasitic approach keeps a constant representation throughout the base level right until the output. This crucially implies that there is no dependency established — even less than in standard horizontal models: any layer can be removed without any significant structural issues (though naturally, the expected behaviour may change). Another consequence is the ability to design the layers completely independently from any other, without having to worry about previous representations. Simple working systems can thereby be achieved with much less effort.

6.3.3 Biological Parallels

The research described here was not the foundation of our model. However, many interesting parallels can be drawn, and due to the novelty of the approach the discussion is welcome.

In the experiment (TALWAR et al., 2002), three electrodes are inserted into a rats' brain, two of which simulate whisker twitches on the left and right sides. The human "pilot" can trigger these remotely to attempt to control the mammal. In order to increase the likelihood of obtaining the desired movement, the third probe stimulates the reward centre of the brain. Eventually, the rat learns to associate a virtual flicker on either whisker with movement on the corresponding side. While this is not especially complex conceptually, the advantage of the model is that rat's locomotive capabilities are not discarded (rats are great climbers). The result is a fully functional navigational entity of much higher quality than we could expect from robots, even within decades of more research.

This model is extremely interesting for neuroscience as well as biology, but the links for Artificial Intelligence, especially navigation, are also of great benefit:

- Robustness — The fundamental navigational behaviour of the rat is not overridden by the electrodes. The millions of years of evolution allowing such smooth and reliable motion is thereby not discarded, subsumed, or otherwise sacrificed in favour of human overrides.
- Simplicity — From the pilot's point of view, there is very skill required in controlling the rat. A simple simulated twitch of the whisker on either side sets the rat in motion along a complex obstacle course.

Since the rat's obstacle course can be seen as a particularly tough test environment, the applicability seems quite generic. An artificial version of this model would have mainly benefits for robotics and animat creation, though the theory may assist neuroscience also. The challenge lies in translating the experiments from roaming rats to computer controlled virtual creatures, as this is a more elegant approach suited to creating human-like navigation.

6.4 Technology

As a reminder, the motion-controller requests obstacle distance information from the surroundings in order to perform its reactive mapping onto motion. Typically, there are three values passed — corresponding to left, front and right whiskers. This layer intercepts the calls, and modifies their value. This process is done by a neural network mapping learnt online by artificial evolution, in a fashion very similar to the previous chapter.

6.4.1 Formulation

The task at hand is defined mathematically as follows; given a set of samples $s_i = (d_i, a_i)$ from the surroundings (respectively distance and angle values), determine the set of values d'_i such that they maximize the long-term fitness function f.

The problem is modelled as an additive function g, since this simplifies its learning greatly:

$$d'_i = d_i + g(s_i) \tag{6.1}$$

We also make the function g modular, in that it is applicable to individual samples rather than the collection of data as a whole. Early models were successfully tested using this latter brute force approach, but this was discarded since there are many benefits associated with processing individual samples:

- Simplicity — Since the function is to be learnt, any reduction of search space size is welcome. Modularity is a well-known technique among the machine learning community for achieving such a task. The additive function, as defined in *Equation 6.1*, achieves this by abstracting the computation applied to a single sample.
- Flexibility — Any brute force solution that expects a fixed number of samples will fail when the request counts do not match. If the motion controller was extended to adaptively sample the surroundings in an incremental fashion, such an approach would fail. When individually processing samples, however, these limitations can be overcome.

The theory must now be put into practice using evolutionary neural network.

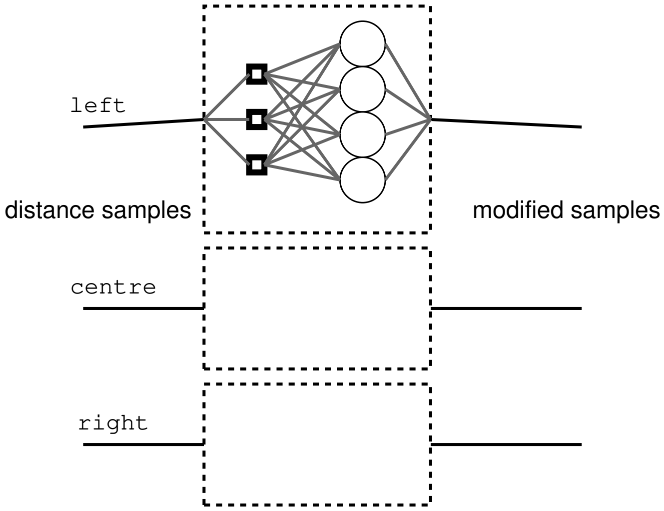


Figure 6.2: The modular neural network which processes each distance sample to influence the movement in the appropriate direction.

6.4.2 Procedure

Once again, flexible behaviours are used to tackle this task; a single feed forward perceptron is used to learn the additive function. To the reader extremely familiar with NN techniques, the parallels with the modular neural network approach by Gruau (1995). The main difference is that we manually design the solution, whereas Gruau's approach would allow this to be evolved.

While there are no particular restrictions on parasites, common themes arise during their design.

- Since the parasite is responsible for keeping the host healthy, it must monitor its fitness feedback. The parasite can learn to take this into account when applying its influence.
- While reactive behaviours can be combined in a very rigid fashion too, the parasitic approach allows blending behaviours and should be taken advantage of.
- Selecting features for the parasite to base its decision on is not a trivial task. It must permit the first two points above, as well as allow the reactive behaviour to be accomplished. Especial care must be taken in this process, especially when attempting to model the modular function.
- Finally, the realism also needs to be taken into account. This must be done by providing high-level parameters that can be tweaked in a simple fashion.

6.5 Model Design

Fleeing and seeking are abstracted into one script function, but their parameters are tweaked individually as appropriate.

Procedure

Seeking and fleeing behaviours are only combined with the obstacle avoidance behaviour. Once this host has been evolved successfully, the parasites are learnt. It is trained on average for 60 seconds, attempting to turn the host to a specific orientation. This rotates variably over time, and mimics seeking of a point in space.

Features

The major issue with designing such parasites is deciding what inputs to give the neural network. As mentioned above, they need to allow the host to successfully turn the parasite, as well as blending behaviours when the appropriate situation arises.

Final selected features for a single sample `(distance,angle)` include:

1) The raw distance sampled by the whisker.

2) The absolute value of the relative orientation.

3) The raw fitness of the host for the past timestep.

4) The clamped angle relative to the desired orientation, normalised with respect to all other relative angles.

The output, as explained, is simply added to the distance to create the modified distance, which is passed on to the host.

Parameters

The reactive behaviour is another node in the navigation document object model (DOM), and can be found in full detail in *Appendix A.3*.

First, parameters for the fitness function need to be specified in high-level form. In essence, divergence from the target is punished. Laziness can also be rewarded to prevent unnecessary modifications to the distance samples from the whiskers, more so under dangerous conditions (when the walls are close).

The neural network used is evolved with this fitness function. Five inputs correspond to compound features used to base the decision on, and sixteen hidden nodes are required in this case.

Script

The function in the `Python` script performs the computation, and defines how the parameters are used.

The pre-processing phase normalises the angles of the whiskers relative to the desired orientation, and finds the minimum, maximum and average values. This is to transform the data into a convenient representation for the neural network.

A loop construct deals with each whisker individually. A sequence of features is built up, containing the most relevant criteria that are used to decide how to modulate the output. The neural network processes these features, and returns the value to be added to the distance sample. The fitness components are then computed, punishing large changes in dangerous situations, and to a lesser extent in normal conditions. Persistence is also rewarded to prevent jerky movement.

A final post-processing stage punishes divergence from the desired orientation, reports the fitness to the neural network, and last of all returns the modified distance values.

Results

The seeking behaviour proves generally reliable, though finding the right balance in the weighing parameters can be crucial. If the values are too strong on either side, the animat can seek the heading too much and run into walls, or not enough and it looks like it's wandering around nonchalantly.

Once again, in some scenarios unseen during the evaluation, performance drops and the animat can even get stuck in a corner. In some cases, it manages to break free, but in others the arbitrator must intervene and disable the parasite. The chances of this occurring are greatly diminished when the terrain previously learnt is used to drive the seeking; indeed, rarely is a move straight into a wall even requested!

6.6 Analysis

When compared to the biological model used on the rat described above, the system does prove a success; although there are some small differences that should be pointed out.

- Continuous Action — With the rat, the sensations in the brain are performed in an extremely coarse fashion, presumably for the comfort of the rat; only two probes are implanted and the stimulations are temporally quite rare. This differs with our approach in that the host is constantly being deceived. Additionally, the information is modified in a smoother fashion — rather than using binary on/off approach. One could consider our model as a more general superset of the rat model.
- Mirrored Signals — While the rat receives signals on the left side when it is required to turn left, our approach on the other hand tends to modify the perception on the opposite side. Conceptually, it's a bit like placing fake walls in the mammal's mind in order for it to treat them as normal.
- Learning Method — It is generally assumed among cognitive scientists that the method of learning closest to the mammalian approach is temporal difference learning. We do not utilise this approach, instead making use of the genetic algorithm. While the outcome of the learning can be considered similar enough, the learning process differs quite significantly.
- Neural Influence — In the rat model, the artificial stimuli mimic sensory inputs. In our model, they are placed between the cognitive model of the world and the motor control, and influence the flow of information between these two components. A more biologically accurate model could be obtained by placing the parasite right after the sensory connections.

These differences were partly a consequence of the fact that the research did not actively set out to model the neuroscientific experiment. Also, some were necessary changes to cope with the simplicity of the model, allowing it to generate interesting behaviours none-the-less. Finally, some changes can be attributed to the sake of experimentation, proving in the end to be a valuable addition to the model.

Chapter 7

Topography Learning

"How often I found where I should be going only by setting out for somewhere else." —*R. Buckminster Fuller*

To support any form of higher-level intelligence such as planning, the navigation system requires an internal model of the terrain. This representation is learnt online in a human-like fashion to support dynamic conditions and assimilation of new maps.

7.1 Overview

The challenge at hand is to map space, in a more or less precise fashion. The degree of precision is usually a factor of the quality requirements, the rest of the architecture, and the restrictions imposed by the hardware. These already represent important issues, before even reaching the technological side of things.

While our mathematical understanding of Euclidian 3-space allows us to understand each point as a unique vector, often the only way to recognise a location is by its surroundings. Practically, this can be characterised by all the sensory data available at a particular location. Even with perfect sensors, there are ambiguities in the world, due to the many to one mapping from position to sensory data.[1] These ambiguities can be partly resolved by using the concept of "position" and updating it as time goes by. This second point presents two complementary options for representing places.

A third issue involves the obstacles; should these be stored explicitly in the representation, or are they implicit in the mapping of open space? Both options naturally have their own advantages. When the obstacles are ignored, the representation is compact and to the point. On the other hand, if the obstacles are stored, a more precise model of the world is created, which comes in useful for exploration and terrain analysis.

Finally, the learning of the terrain in autonomous systems is often coupled with the localisation algorithms, aiming to position the animat in space. In our case, complex localisation was neglected, mainly due to time constraints. We rely mostly on odometry and local correction of the position using perceived objects. Global localisation is not performed, which tremendously simplifies the development of a terrain-learning algorithm.

[1]A predicament known as perceptual aliasing, briefly mentioned in the introduction.

7.2 Requirements

This section underlines generic criteria for judging terrain-learning algorithms, which become requirements for our own implementation.

- Pre-processing Independent — As animats do not usually get the opportunity to analyse the terrain before they have been on location, they should be able to learn from being in the new environment, but also potentially from a more or less precise map when possible.

- Dynamic — Since small changes may occur in the environment during the simulation, the agent should be able to cope with them as they are noticed.

- Incremental — Since the animat is expected to build up its knowledge of the world from scratch, and be capable of dealing with dynamic changes, the learning of the topography should be done in an incremental fashion.

- Generic — The system should make few assumptions about the nature of the environment; for example, both indoor and outdoor settings should be supported. The properties of the animat should also be flexible, so that varied sized creatures with different capabilities would still be able to learn the topography.

- Efficient — Since the process of learning the layout of the terrain is a small aspect of the navigation system, it should have a minimal overhead, as any performance overhead will affect the rest of the system.

- Optimal — The module should try to learn strictly the information required to perform the task required from it. When striking this balance between practical benefit and memory requirement, the efficiency reaches a maximum.

- Noise Tolerant — Since from the low-level sensorial point of view there are no guarantees that the data is accurate, the terrain learning module will not be able to rely on that fact.

These contrasting requirements must be taken into account in the best suited fashion to solve the problem at hand.

7.3 Existing Approaches

7.3.1 Biomimetic

Some approaches are inspired by study of real animals, based on how they are capable of locating themselves in space. Conceptually, each of the approaches explained below are based on the perception of the animats; their knowledge of a place is based on what their senses told them about its surroundings. For further information, and more detail about each of these solutions, the keen reader should look into the work of Trullier and Meyer (1997).

Place Recognition

Such models associate the input from the sensors with a conceptual location (O'Keefe and Nadel, 1978; Burgess et al., 1994). In practice this can be done with standard pattern recognition technology, which provides *"Landmark Triggered Response"*. This can be learnt over time, but there is little consistency within the model; all known landmarks have no connection between each other.

This is a very reactive model that can be found in animals such as rats. Simple feed-forward perceptrons have been shown sufficient to model this.

Topological Representation

When the world model is aware of connectivity between the landmarks, more sophisticated cognitive processes can be employed, (e.g. in planning and localisation). This allows a deliberative process since the states are connected, as opposed to a simple reactive process implied by the previous model (Schmajuk and Thieme, 1992; Trullier and Meyer, 1998).

However, since each connection in the graph has few known properties, they cannot be exploited by the animat reliably. Indeed, if the animat doesn't know which actions lead to taking which path, then moving around in the topology becomes extremely arbitrary.

Metric Maps

For these models, angular and distance information is also stored with the connections. This allows the animat to compute expected travel times, and know which direction it needs to head towards in order to reach a neighbouring node (Wan et al., 1994; Worden, 1992).

Geometric Maps

Finally, taking into account global consistency of space, geometric maps assign a position to the known places. This is a fairly accurate model of real world, though it tends to be projected into 2D due to gravity; the third dimension is often not required. Such representations allow a more elaborate form of anticipation and knowledge based processing of the terrain. These appear to be used by dogs, monkeys and men.

7.3.2 Strong Representations

Some terrain models are based on more explicit data structures, using stronger mathematical basis (grids, triangles). These representations are not based on perception, rather the mapping of 3-space in a convenient fashion.

Grid Worlds

In many cases, it is possible to exploit properties of the environment and represent them as regular lattices. Square components are most frequent, as they offer simple storage and adjacency handling, but hexagonal grids can also be found.

Often, both open space and obstacles are represented as Boolean conditions, but probability distributions are becoming increasingly popular in robotics.

Cellular Representations

Other cellular representations are not necessarily based on straight grids. The ability to handle irregular discretisations is often useful as complex worlds rarely conform to such linear patterns.

In practice, much simpler constraints are imposed for placing each node in the grid. The only restriction is one of minimal distance to any other node in the grid.

Polygonal Meshes

A world can also be decomposed into surfaces that can be walked upon by the animat. Often in virtual worlds, these can be defined in terms of the geometry used for graphics or physics, created in the same process. These triangles laid on floor constitute a graph also, where nodes are triangles and links are determined by adjacency.

7.3.3 Synopsis

The strong representations often carry unnecessary detail and complexity, and as such become a strain on the memory as well as a burden for the path planning. The simpler biometric approaches tend to rely too heavily on sensory pattern recognition, and do not provide any sense of space. It is often obvious that humans have a sense of space, so the data-structure should really be based around a Euclidian reference axes, much like geometric maps.

7.4 Foundations

The sophistication of the low-level behaviours allows much less information to be stored in the representation of the terrain; despite the lack of precision, the reactive behaviours will be able to exploit the knowledge reliably. Objects and items in the world should be incorporated into the representation of the terrain too, in order for them to take part in routing strategies. While localisation is an important issue, most of it can be done independently from the topography; odometry and visual position correction is satisfactory for short distances. Global relocation is generally extremely predictable in nature (notably during teleportation or respawn after death), and can be neglected during the design of the terrain learning. Additional auxiliary data-structures and algorithms can be devised as necessary.[2] Essentially, all this is achieved by a coarse graph mapping empty areas and learnt incrementally. This covers Euclidian space rather than being built upon a perceptual basis.

7.4.1 Overview

The main concept behind the solution is inspired by Hansel and Gretel's story tale. Virtual pebbles are dropped in memory at regular intervals, when the previous one is no longer accessible. The model is improved by storing nodes in a graph rather than in a sequence, which allows arbitrary terrains to be assimilated without relying on a specific exploration policy.

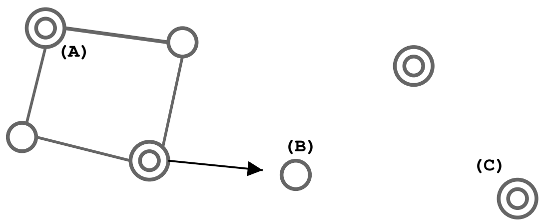


Figure 7.1: Learning paths by movement: (A) Existing graph learnt previously (B) New node dropped and linked due to restricted access back (C) Nodes representing objects that have been seen but not connected yet.

The graph learnt includes nodes which essentially mark out objects in the terrain. These are included so the path-planning can include them in the routes when necessary. Additionally, they tend to provide a great skeleton for the graph, as objects seem to be often placed down in key areas like intersections, corners or even dead ends. Such nodes are placed automatically when new objects become visible to the animat.

Since in some locations there are no items around (or they cannot be perceived), the terrain learning algorithm does not rely on their presence. This second type of node is purely virtual (handled completely transparently from others), and leaves no mark in the physical world itself. Normal nodes can be placed in arbitrary positions, in order to provide links between object nodes or to mark out empty rooms. Such link nodes are required when the original path is too complex, or the animat cannot move directly between the two end points.

The nodes dropped and connected together as the animat navigates around in the world. If it moves from one node to another, then there is a possible path in space that can be used, and a virtual link is created in the graph. If it moves into an area with no nodes, a new one is dropped and a link is created too! Doing this repeatedly allows the layout of the terrain to be learnt incrementally.

[2]For example, simple global localisation has been performed by detecting single visible objects that match unique nodes in the topography.

7.4.2 Assumptions

Exploration

Since the representation of the terrain is spatially coherent, the coordinates of existing points in space could be used to guide the exploration process. While these benefits are undeniable, we assume a simple reactive navigation strategy from the animat. When small amounts of noise in the sensors are used, repetitive patterns do not arise and the animat tends to wander into different places. Also, regular changes of "hosts" provide good results, since each are best suited to different situations; wall following can find doors and exit rooms easily, while obstacle avoidance scans areas very efficiently. This tends to emergently explore entire levels. Human examples combined with a desire to reach particular objects can both improve the process too. However, the algorithm itself doesn't rely on any particular exploration policy.

Areas

The algorithm assumes that the current position is known, either by odometrically tracking it or by using localisation. If the current position is uncertain (for example after teleporting or respawning), the algorithm will start to learn the terrain as a separate area. This will be adjusted with more information and experience of the terrain, and merged with the original area when a link is discovered.

Media

The environment can be conceptually divided into different materials *(air, water and ground)*. These affect how the animats interact with parts of the world; cue the concept of "medium". Defined as a surrounding or enveloping substance, this includes the media mentioned. However, we extend this definition with virtual media that simplify the handling of certain situations. Such concepts include ladders, teleporters, and platforms — each of which directly affect the movement just like natural media.

The algorithm assumes that the animat can determine the current medium, using the physical sensors. This would allow it to assign a medium type to each node, and modify the topography learning algorithm based on local changes.

Spatial Common Sense

A strong sense of *spatial common sense* is relied upon to produce high-quality maps, and this is a key aspect of our solution. This drastically reduces the times required to learn maps, as it does not require exhaustive scouring of the terrain (trivial movement is abstracted out). The quality is also increased, as the information stored is more concise and relevant.

Fundamentally, the SCS is an anticipatory state transition function; it determines if local movement can be performed or not. It attempts to emulate the strong ability of humans to visually extract short paths from their environment (e.g., *"Yes, I can walk to that table from here."*).

Given to points in space, the SCS function needs to determine if it's possible to get from one to another. The function itself can take into account parameters such as distance, height, and the visible structure of the landscape geometry to make the guess.

7.4.3 Procedure

The basic method for updating the topography is defined as follows:

1) Determine the current position and medium, based on proprioceptive sensors, odometric queries and local position correction.

2) Check if an update of the topography is needed based on the current and previous state (position, media, etc.).

3) Locate the closest currently accessible link to the topography.

4) Update topography based on the current location and the previous one.

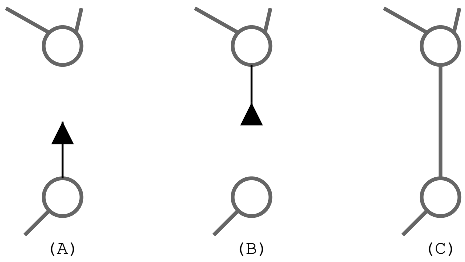


Figure 7.2: Learning paths by movement: (A) Closest node known, agent moving away (B) Different nearest node found (C) New path created.

5) Actually set the current state to the new parameters determined by the sensors.

The update of the topography is a compound procedure which happens as such:

1) Check medium for problematic conditions, and if necessary update the links to the topography.

2) Conditionally link the current edge to the previous (edge/vertex).

3) Optionally connect the current vertex with the previous (edge/vertex).

4) If required, create a new vertex [and optionally link to previous (edge/vertex)].

These operations depend on the type of link between the animat's position to the topography, which can either be edges or vertices.

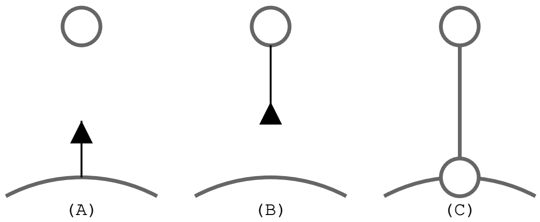


Figure 7.3: Learning paths by movement: (A) Closest link is a path (B) New closest link found (C) Path split at link point, and new path created.

7.5 Technology

The solution presented here is a very flexible framework which can be tailored to match particular scenarios. However, there are some common requirements that need to be taken into account.

7.5.1 Nearest Neighbour Search

This k-NN search is used to find the nodes of the topography closest to the current position. From a functional point of view, it can be treated as a black box, since the underlying implementation should not affect the results in any way. While a binary point tree was initially used to perform this search, subsequent implementations used a sweep scan on two sorted arrays containing the coordinates of the points. This proved to offer better cache consistency in the implementation.

The search can also return not only nodes, but also points on edges. When the animat is closer to an edge than a point, the closest projected point is found and returned instead. This allows coarse graphs to be refined without losing precision, as the edge can be split to place a new node to be processed transparently.

Data Structure

Like most terrain representations, a directed graph-like structure is used as a model. This consists of nodes and edges. Nodes can represent items, objects which affect movement, or just empty space. Edges indicate a possible route between two nodes in space.

7.5.2 Spatial Common Sense

Although a probabilistic result would be a better estimate, this is done using a Boolean function — allowing it to be simply included within the nearest neighbour search. It is simply used to filter out close nodes that are not accessible.

In practice, these movement checks are done by an expert system, though we believe a learnt function would perform ideally here. Close nodes are always marked as accessible, which filters out redundant checks. Visible nodes can generally be checked for accessibility using the visual sensors. Invisible nodes are assumed inaccessible, unless they were previously selected; in this case, they are assumed accessible until the animat steps too far away from the path back.

7.5.3 Algorithm

The algorithm relies on finding the closest accessible nodes on the topography, achieved by combining the nearest neighbour search with the spatial common sense. At any time, the active point on the topography can be reported. It can be a specific node, or along a path.

As the active point is changed, a quick check is made to see if there is a corresponding link in the topography. If not, then a new link is created.

The concept of areas is used to counter localisation problems. When the position is uncertain, a new area is created. As the animat confirms how two areas fit together, they are merged into one in a moment of revelation.

7.5.4 Parameters

The algorithm is mainly controlled by the parameters which refine the nearest neighbour search. Essentially, they restrict the nodes and edges selected even before spatial common sense is applied to filter out candidates.

First, there are parameters to limit the selection of nodes. These are defined proportionally to a step of an animat, corresponding to a length of 18 units. Only the points within a certain range are acceptable, and only a certain maximum of nodes can be processed (this avoids unnecessary complexity of the graph and improves the efficiency of the search).

```
const float selectAlwaysDist        = 72.0f;
const float SelectMaxDist             = 540.0f;

const int   closestPointsMax          = 4;
```

Secondly, the selection of points on edges is also restricted further. Notably, the parameters prevent points too close to the end nodes from being selected, as well as imposing a maximum limit. There is also a penalty added for selecting a point on an edge, as the algorithm generates better quality graphs when "normal" nodes are selected in preference.

```
const float minDistFromEnds         = 72.0f;
const float edgeDistPenalty         = 108.0f;
const float selectEdgeMax           = 360.0f;
```

7.6 Example

A simple terrain was learnt with various different parameters to show the versatility of the system. These are shown in *Figure 7.4*. The first map has a very fine setting; points are placed regularly, paths are not split, no objects are inserted into the map, and spatial common sense is disabled. The results look similar to regular grids as used in many robots or commercial game AI implementations. The second map is learnt using a much coarser setting, better suited to the system — and terrain learning in general. Path splits are allowed, nodes are not forced to be close together, objects are inserted, and spatial common sense is used. The result is a map that has all the necessary details for planning paths, as well as guiding the reactive behaviours.

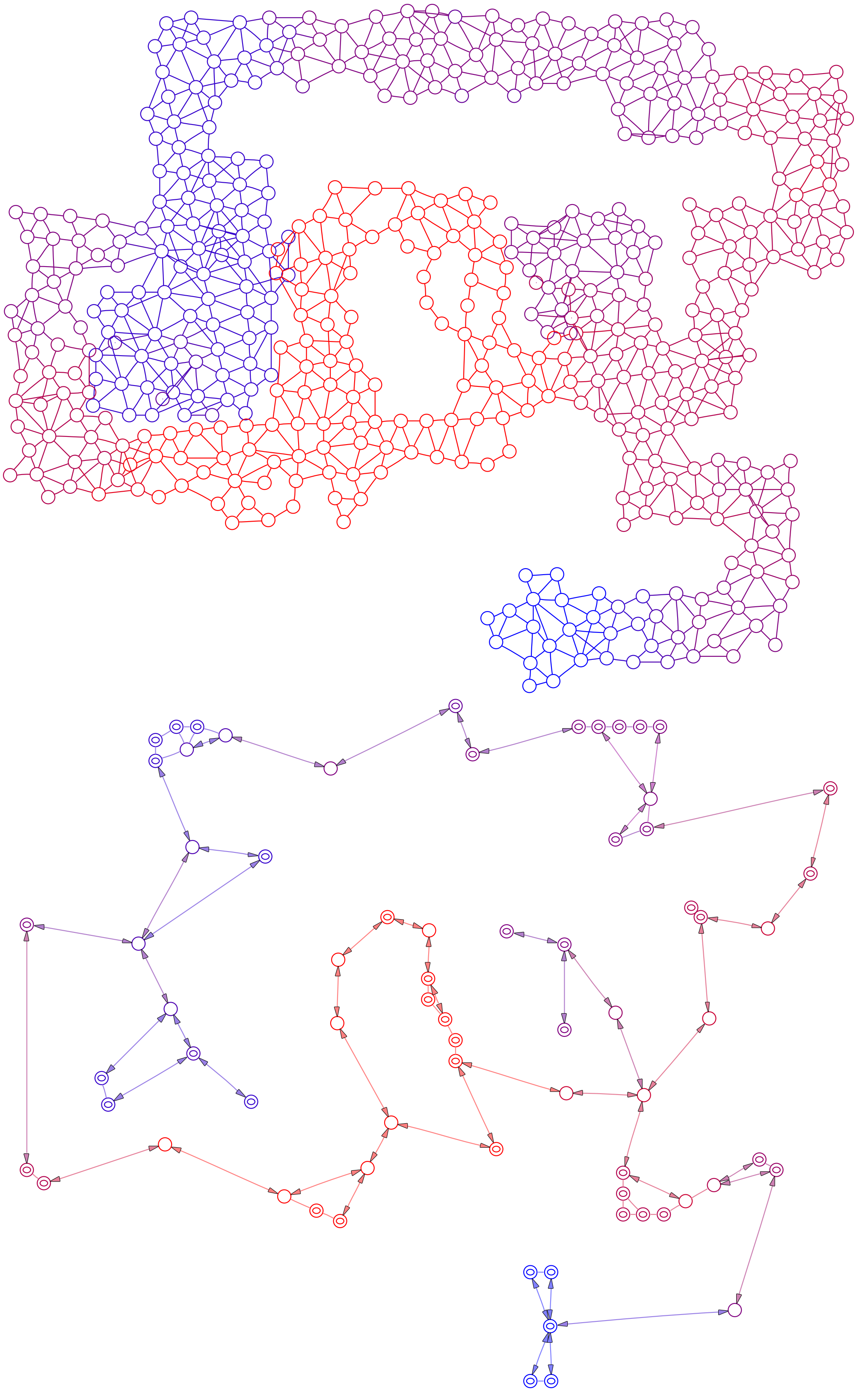


Figure 7.4: Above, a fine resolution map with no spatial common sense. Below, the same terrain learnt with SCS, inserted objects (double circles) and a coarse resolution. Colour merely indicates altitude.

Chapter 8

Pathematics: Deliberative Path-Planning

"It isn't where you came from, its where you're going that counts." —*Ella Fitzgerald*

Though some believe intelligent behaviour may arise from cunning arbitration of reactive motion, it is often necessary to influence the outcome with a more deliberative path-planner. This is done by reasoning over the cognitive model of the terrain, in order to generate the desired move sequence.

In practice, the path-planning exploits the learnt topography, and determines which routes should be travelled to achieve particular goals. Suggested headings are then passed to the parasites which can influence the movement to achieve these targets.

8.1 Overview

Many optimal path problems have been formalised over the decades (Martins et al., 1999) — and various solutions devised to efficiently compute the required result (c.f. review below). However, an overwhelming majority of these papers consider broad settings and unrestricted queries that are tuned for equally generic applications. Typically, when targeting particular scenarios, numerous assumptions can be made, and together with specific requirements, custom-tailored algorithms shine through.[1]

When applied to autonomous agents, things are no different. Current path-generation components are based on a selection of static shortest-path problems (single pair, single source, all pairs), using a variety of algorithms ranging from heuristic depth-first searches to building minimum spanning trees. They are used to determine shortest paths from scratch, or at best by looking up previous computations in a cache. Dynamic algorithms have been overlooked by the intelligent agent community, probably due to the overwhelming aspect of the theory or the overhead of the data-structures required by some solutions.

We believe intrinsic properties of embodied creatures (such as persistence and continuity) allow the development of a more efficient algorithm, better suited to the problem. Furthermore, by adding psychologically inspired characteristics to our design, it becomes increasingly capable of modelling human-like behaviour. In this chapter, a fully dynamic algorithm for agent path-planning is presented. It is continuous since computation of approximations is performed online, and can be halted at any point. The heuristic approach also allows the result estimates to improve over time.

[1] An example is the Intelligent Route Finding system that finds shortest paths based on human settings and learnt user preferences (Liu, 1996). Another is the page rank scheme (Page et al., 1998), that uses a customised network flow algorithm to determine the importance of a web-page.

8.2 Related Work

All path-planning research relies on *graph theory*. Let $G = (V, E)$ be a digraph, where V is the set of vertices of cardinality n. E is the set of directed edges connecting all the nodes, with $|E| = m$. Each edge (u, v) is associated to a length l_{uv}, denoted by the function $l : E \rightarrow \mathbb{R}$.

The simplest shortest-path problems are called *single-pair*, where only one path needs to be found. While these can be solved using a greedy heuristic search, more elaborate data-structures are required for *single-source* problems.

A spanning tree is a subset of the graph such that $T_r = (V, E')$ and $E' \subseteq E$, where E' contains no cycles.[2] For the tree to be valid, it should be possible to traverse the data-structure from each vertex u to the root r. The set of edges required during this traversal is called a path, where its total length $D_u^{(r)}$ is the sum of the individual edge lengths. A *minimum spanning tree* verifies:

$$D_v \leq D_u + l_{uv} \qquad \forall (u, v) \in E \tag{8.1}$$

Such a spanning tree is generally at the base of most shortest-path algorithms, and tends to be the format of the result too.

8.2.1 Static Shortest-Path Algorithms

The algorithm by the late Dijkstra (1959) is one of the most popular solutions to the single-source problem, based on a relaxation process that iteratively adds candidates to the spanning tree — starting with just the root node. Candidates are picked from the part of the graph directly accessible via one edge from the current tree. When nodes are selected with a best first strategy, they do not need further processing once their distance has been set; this assumes that no negative-length edges are present. Thus, this label-setting procedure can return within $O(n^2)$. Over the years, many changes to this fundamental algorithm have been made, notably improving the updates and searches of the data-structure storing candidates (with S-heaps (Johnson, 1972) and buckets (Dial, 1969) instead of an unsorted linked-list).

The Bellman-Ford-Moore algorithm performs successive approximations to solve the system of equations (Bellman, 1958; Ford, 1956; Moore, 1959). By maintaining a list of vertices to scan, redundant computation can be prevented and convergence can be detected. This implies the algorithm can run in $O(mn)$, as well as being capable of dealing with negative arc-lengths. In practice, this label correcting process can be enhanced further with specific temporary data-structures (Pape, 1974). Heuristics based on adjacency can also improve convergence in practice (Goldberg and Radzik, 1993), though the theoretical bounds remain the same.

8.2.2 Dynamic Re-optimisation Algorithms

While these initial approaches are static in nature, much recent work tackles dynamic algorithms that can re-optimise a minimum spanning tree. Indeed, in a sequence of problems, the $(k+1)^{th}$ problem often barely differs from the k^{th}. The literature has mainly focused on two sub-types of algorithms; the first handles changes of origin. Gallo (1980) noted that the sub-tree rooted at the new node is still optimal, and uses reduced costs relative to the existing shortest-path tree to compute the new one.

The second case deals with changes in the graph's specification; incremental algorithms can deal with edge insertions (and — less commonly — length decrease), while decremental algorithms handle edge deletion (with length increases, respectively). Fully dynamic algorithms handle both these cases, but semi-dynamic versions stick to one. While early approaches were memory-intense, Gallo (1980) provides an elegant solution that identifies these two scenarios, and updates part of the spanning tree with procedures similar to the static approaches mentioned above.

8.3 Autonomous Agents & Navigation

This section proposes a new approach to path-planning that extends the fixed destination paradigm.

[2]It also holds that $E' = |V| - 1$, since each vertex requires a connection to an antecedent, except the root.

8.3.1 Specification

In order to allow the flexibility of the higher-level interface influence the path-planning, the standard digraph model needs to be enhanced by assigning reward values to each of the nodes. These can be understood as implicit destinations, as described in the design section. This allows the full panoply of the animats desires to be understood by the routing, which remove the aliasing normally observed with single destinations. Formally speaking:

- *reward function* A time-dependant function that maps vertices onto real rewards values, $R : V \rightarrow \mathbb{R}$. This specifies how much interest the agent shows in a node. For the sake of soundness, we assume the animat can only collect the reward once.

- *cost function* Reminiscent of other graph algorithms, this is a dynamic function that assigns costs to edges $c : E \rightarrow \mathbb{R}$. Conceptually, this measures the expected distance between the agent's *state* at both ends of the path. This can be quantified using travel time, space, or a weighted combination thereof.

The astute reader will notice the parallels with network flow theory. Each node with a value can be seen as a "source," and the animat's location as the "sink." However, there is no maximum capacity of an edge, and the process does not attempt to maximise the same variables.

8.3.2 Queries

A planning component should expect virtual agents, ranging from the simplest to the most complex, to request the following types of information:

- `Path` Request of a specific single-pair path P. When it is generated, edges should be chosen that maximise:

$$-D_P + \omega \sum_{i \in P} R_i \tag{8.2}$$

  Informally, ω is the trade-off factor between path-length D_P and reward R collected at all nodes. When $\omega = 0$, "standard" physically shortest paths are returned, while $\omega > 0$ encourages the agent to wander more and collect the rewards.

- `Direction` The path-planning should suggest if there is any benefit in moving, and if so, which step should be taken to maximise long-term reward collection (i.e. $\omega \gg 1$). This is the recommended query, as it intrinsically deals with dynamic conditions by letting the underlying implementation decide moves, based on the data provided.

- `Distance` Often, before the path is even considered, an estimate of its length is required for decision-making. This operation should approximate this result more efficiently than by a full path query.

Any implementation of these queries should attempt to match the specification as closely as possible.

8.4 Pathematics

In this section, we catalogue interesting properties of autonomous agents that model humans, and design a path-planning algorithm that takes advantage of them.

8.4.1 Foundations

Proposition Agents show a high proportion of egocentric path queries.

Cognitive models of the environment are fundamentally based around the entity, due to the information being perceived subjectively. Reasoning is also primarily proprio-reflective, as the main purpose of deliberation is to anticipate how the agent's actions may affect events. Navigation queries are no exception.

Our solution employs a directed tree, rooted at the agent's location in the graph. This has some limitations, but the advantages of such a representation outweigh them. Moreover, the higher-level AI can be designed and optimised around this; queries with sources different to the root must be performed on the fly using A*.

Proposition At the very least, sub-optimality should not be considered a predicament, rather an intrinsic advantage — especially when stochastic in nature.

Path queries and length computations rarely require to be exact. Even with perfect information at a given time, the randomness of the environment during execution, limited scope and perceptual aliasing imply that highly accurate results are little extra benefit. Furthermore, when designing human-like agents, one often needs to randomly perturb the paths to encourage more interesting patterns (Ritchie, 1998).

Our paradigm stores approximate path lengths in each node of the directed tree. These estimates are corrected by the main iterative pass of the heuristic algorithm, using stochastic properties.

Proposition The cognitive model of the paths through the terrain has a high degree of persistence.

Due to the physical constraints imposed on the agent within the world, changes to its state remain limited. For example, in a couple seconds, you can expect an agent to walk at most a few meters. Also, the agent perceives this information incrementally. In practice, this translates into a strong persistence of the cognitive model. Admittedly, dynamic environmental conditions provide some sudden state changes, but in practice they remain rare and localised (within sensory range).

A dynamic approach (both incremental and decremental) that maintains a spanning tree, and re-optimises it from frame to frame takes advantage of this fact, reducing the redundant computation to a minimum. Our directed tree is also stored implicitly; rather than maintaining pointers, the distance estimates at each node can instead be checked to determine descendants.

Proposition Agents often need to trade-off cheap rough estimates with more expensive accurate results, as they see fit.

Robots and animats are continuous entities in time; they belong to the world, and can expect to exist at time $t+1$. Though executing an immediate decision at time t may have advantages, delaying for a while is often an option which can sometimes even prove valuable. When the cognitive process is not interrupted, the wait can imply discovering a better solution. This approach proves to be extremely human-like in nature, as both instinctive decisions and thinking times can be modelled.

A continuous algorithm based on successive approximations provides a variable level-of-quality solution. Greedy heuristics incite the results to converge quickly. By relying on simple atomic operations, the algorithm can return at any point.

8.4.2 Overview

At the core of our algorithm are the concept of *reward percolation* that enables us to provide an efficient approximation of the specification in 8.3.2. In essence, reward is amplified as it percolates towards the root. At the agent can then assume that the branch with the most reward is the best direction to head in.

The cumulated reward for a node u is defined recursively in terms of its neighbours $Adj(u)$ that are estimated further from the root:

$$C_u = (R_u + \sum_{i \in Adj(u), D_u < D_i} C_i) * (1 + |R_u|) \tag{8.3}$$

Our algorithm stores the distance estimates of each node in the graph. This is corrected over time, in a fashion reminiscent of the BFM algorithm where values monotonically decrease towards their final target. However, we rely on the precondition that this estimate is greater or equal than the optimal value. This

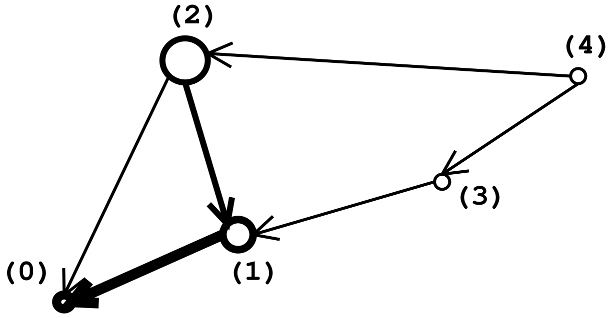


Figure 8.1: Percolation and accumulation of the reward in the spanning tree, causing the animat to select a slightly longer path to collect more reward.

allows us to perform a greedy traversal of the graph in the optimisation phase, which may remind the attentive reader of hill-climbing.

$$D_u^* \leq D_u \qquad \forall u \in V \tag{8.4}$$

Apart from the variables used to store distance estimates and reward, no extra information is required. During the optimisation, the graph is the only data-structure required, keeping things memory friendly and *extremely* simple to implement.

8.4.3 Description

Phase 1, Distance Estimation Updates

This is the main part of the algorithm. It is a heuristic approximation process that corrects the distance estimates for each node. Fundamentally, this is a label correcting process, guided by a greedy traversal of the graph. At a given node, adjacent edges are relaxed if they invalidate *Equation 8.1.* Informally, if the distance estimation of a node can be improved by taking the current edge, we update it. The next node chosen is the neighbour that was corrected the most. When no neighbour can be corrected, a new random start vertex is chosen (initially this is always the root, to preserve correctness of the immediate surroundings of the agent).

Algorithm 1: Greedy heuristic process that reoptimises a spanning tree incrementally.

```
while iterations below threshold do
    for each neighbour do
        update distance conditionally;
        if largest update then
            remember as best ;
        end
    end
    if there is a best neighbour then
        select best neighbour;
    else
        select random node;
    end
    increment number of iterations;
end
```

Though theoretical bounds are not improved with this approach, it allows the re-optimisation to converge extremely quickly in practice. Additionally, the quality of the distance estimates tend asymptotically to the most optimal; initial updates are thereby the most productive.

Phase 2, Reward Percolation

Node reward is accumulated during a traversal of the directed tree, from a randomly selected source towards the root. *Equation (8.3)* is applied directly to obtain the results at each node.

Maintenance Phase

This process simply intends to maintain the precondition of *Equation (8.4)*, when the graph conditions changes. Though this phase has many similarities with existing re-optimisation literature, it does not fall into that category as we only intend it to perform minimal updates to preserve validity — letting the main procedure correct existing estimations.

Due to our sub-optimality tolerance, we can afford not to explicitly take into account all updates; those that do not invalidate the spanning tree can be cast aside. There are only two operations on the graph that cause problems. These are well known scenarios: root node change, and edge length increase (or arc removal). While existing work advocates the separation of these cases, allowing independent research and implementation, we propose to regroup all maintenance operations into one pass.

- When the root changes from r to s, we observe the following property about the new distances:

$$D_u^{*(s)} \leq D_u^{*(r)} + D_s^{(r)} \qquad \forall u \in N \tag{8.5}$$

  Intuitively, any node can be reached by moving to the previous root (length $D_s^{(r)}$), and traversing the previous best estimated path (lengh $D_u^{*(r)}$, which naturally still holds if the previous estimate $D_u^{(r)}$ is suboptimal given 8.4). As r and s are often very close neighbours in the graph, the quality of the estimates does not suffer much.

- If a edge (u, v) increases length by k, similar observations about the subtree rooted at v can be made. All the descendant nodes' distance estimations are at worse offset by $+k$.

We take the approach of an approximate correction procedure, that uses an extremely efficient, cache-friendly $O(n)$ scan of the data-structure. It sacrifices precision over speed, which minimises the overhead of the necessary maintenance procedure.

Algorithm 2: Maintenance pass to preserve the necessary preconditions.

```
offset = distance to old root + sum of length increases;
for each node n do
    n.distance += offset;
end
```

8.5 Exploitation

8.5.1 Case Study

In *Figure 8.2*, all the armor items denoted as squares have a reward of 1. By applying the heuristic process repeatedly, reward flows towards the location of the animat from the different sources, getting amplified along the way. At the root, the animat can select which path to take based on the highest reward.

8.5.2 Issues

In practice, there are some things to watch out for.

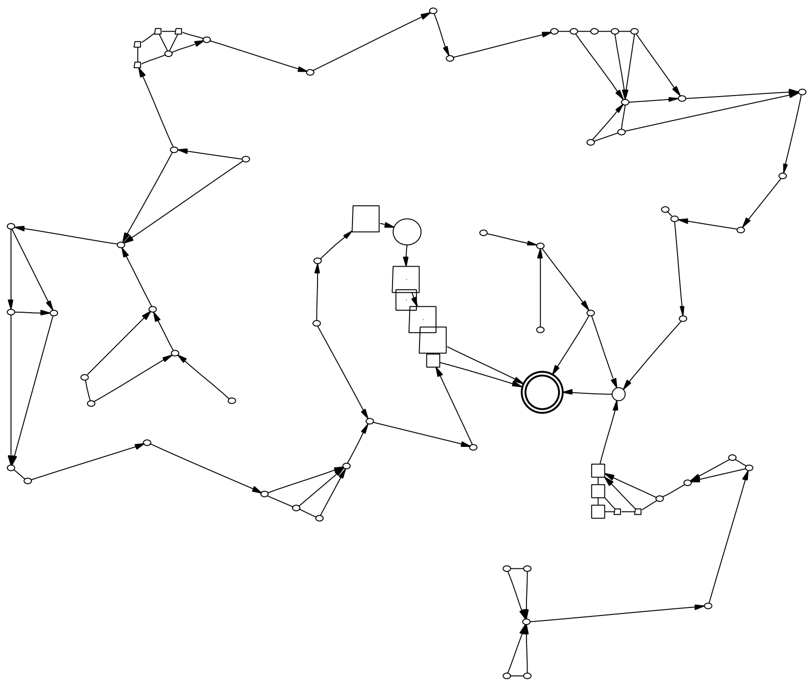


Figure 8.2: Percolation of the reward on the learnt topography. Squares indicate sources of reward that the animat is interested in. Reward flows towards the double circled root, inflating the size of each node proportionally to the accumulated reward.

Root Correctness

The distance estimates around the root need to be taken especial care of. Taking a few extra operations to assure they are upto date before each process will allow the algorithm to perform better, and the reward values to be returned without unexpected surprises.

For example as the agent moves, the old rewards that was percolated towards the root may persist, indicating to the agent he can retrace his steps and get high reward there too!

Momentum

Due to the dynamic and persistent nature of the process, some better solutions may be suddenly discovered when not enough computation was dedicated to the task in previous frames. This can cause the agent to want to turn around completely to exploit the better solution. Aside dedicating more power to the algorithm, the familiar concept of momentum can be introduced. When the animat is 30% as likely to turn around completely, he only does so in extremely beneficial situations (which do not arise often at all).

Reward Magnitude

Often, the animat needs to trade off close items with distant ones. When one is on the way to the other, the algorithm copes well by combining paths. When the paths are separate, a choice needs to be made. This can be done intrinsically when the reward is attenuated along each edge, so closer objects broadcast more "power" towards the root.

Problem Specification

When specific scenarios arise, there is the issue of how to communicate the conditions to the underlying process. For example, when Fred needs to be avoided in the staircase, does one scale the rewards as they are percolated through the area, or should the cost for the surrounding paths be simply increased? In this case, a combination of both tends to work best, but in general each such situation needs to be considered.

Chapter 9

Evaluation

This chapter first evaluates the navigation from a high-level point of view, taking the system as a whole. Practically, the competence of the movement in various different tasks is compared with a human player. The second part looks at each module individually, and does a theoretical analysis where possible.

9.1 Premise

The major part of the evaluation considers the system as a whole — just as the design chapter does. It is assessed in various different conditions, and compared it with select other solutions as well as human players.

9.1.1 Turing Test for Navigation

Since the navigation system strives for realistic behaviours, it seems appropriate to evaluate it with a test for humanlike behaviour. Alan Turing established the most popular of these tests, but within the context of natural language interaction. It involved placing two contestants behind a red curtain; one is a human, the other a computer. If the judge cannot distinguish one from the other, then the computer can be assumed to have a human-level intelligence (or at least NL capabilities).

A test for navigation can be established a very similar fashion; an artificial entity controlled both by a human and artificially can be presented to a judge. The red curtain becomes implicit, as the spectator cannot visually ascertain the method of control.

Our version of the test, however, is rather simplified since it does not include any form of interaction; the judges are only presented pre-recorded animations of the game simulation. A demo fully responsive to human commands would have required an additional natural language layer, which is by no means trivial — and possibly even crucial. The author does intend to tackle a fully interactive Turing test for navigation eventually.

9.1.2 Procedure

The judging process was split into five main tests, combining various situations with two major conditions. Namely, these are simple and complex layouts in both deserted and crowded circumstances. This navigation system was evaluated against human navigation, as well as two other game bots of almost commercial standard. The human candidates were selected with a wide range of skills, ranging from beginners to experts in the game. The existing bots were chosen for their support of the `Quake 2` platform.

Each of the volunteer judges was explained the purpose of each of the trials. There was about a dozen judges coming from gaming or robotics backgrounds, but also unrelated to AI. They were allowed to view the demos freely using identical third person cameras.

9.2 Navigation Trials

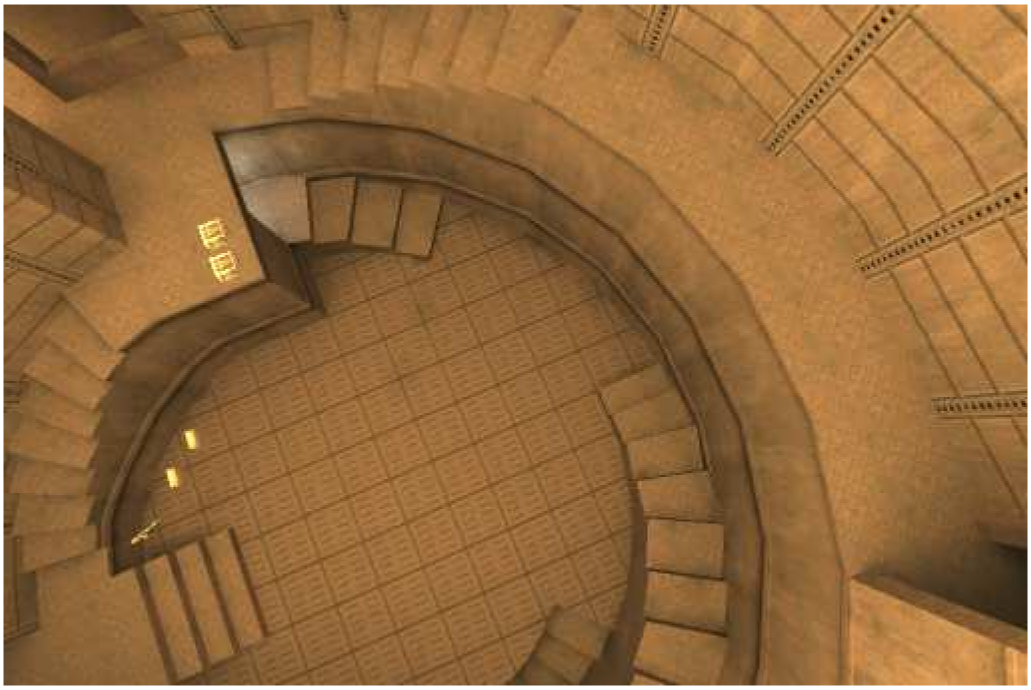


Figure 9.1: Picture of the terrain layout in the lobby trial.

9.2.1 Lobby

Overview

This is a simple test that involves going through a doorway and wandering down three flights of stairs into an open area. The first two stairs have a ledge on their left side and on their right side a wall that stops just before the third section. The whole path spirals around 270 degrees, getting increasingly closer before having clear sight to the target.

Candidates

1) A reactive bot is shown first, only using seeking, wall following and obstacle avoiding behaviours.

2) The human beginner is asked to perform the same task after having familiarised himself with the controls for a few minutes.

Description

The reactive bot does well; the door is crossed effectively and the seek behaviour does a good job of finding part of the way down the stairs. The left wall following is engaged once the target is behind the agent. A relatively sharp turn at the corner prevents the bot from falling off the landing. Both behaviours alternate until the target is reached.

The human player also does a good job of getting part of the way downstairs, but slightly struggles at the corner having to pause for a second to realign.

Results

Figure 9.2 shows how more people in fact ranked our reactive bot as most realistic, obtaining an average barely below 4, while the human beginner scored 2.5 on average realism. No judges thought the human was actually human, although about three quarters thought the reactive bot was.

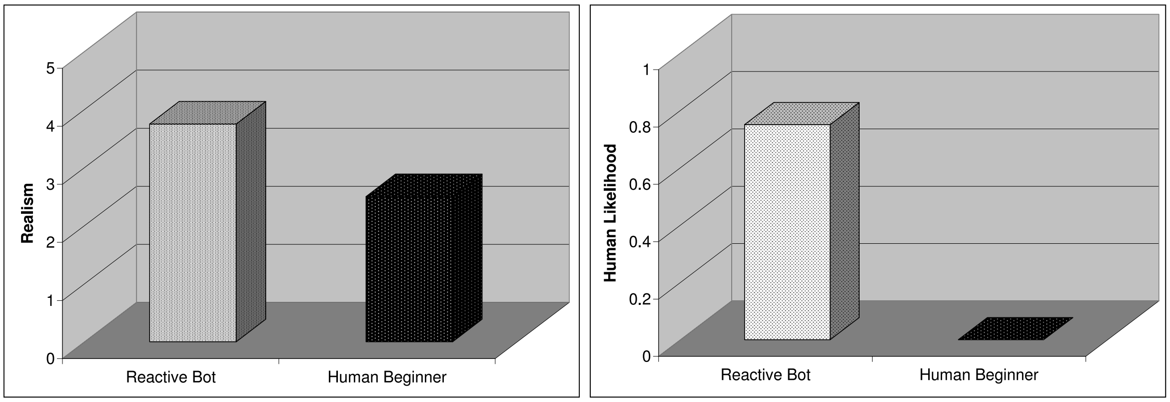


Figure 9.2: Comparing the perceived realism of the players and their likelihood of being human based on the judges votes. The reactive bot appears more realistic.

Analysis

The results are extremely surprisingly for two reasons. Firstly, the judges could not distinguish our reactive bot from a human, and in fact most thought it was — which is a major success in itself. However, despite the impressive display, some gamers pointed out some details that are indeed unrealistic (mainly not cutting corners). These are problems with the reactive nature of the solution. Additionally, the reader should not assume that such realism is achieved consistently. While this is one of many cases handled in a human like fashion, in some situations the system tends to strive for reliability over realism.

The second interesting point involves the inability of the audience to associate incompetence with humans. Presumably, this is due to the audience's desire to identify and mark down players that are unlikely of being artificial, which they wouldn't do outside the bounds of the trials. To the trained eye, the characteristics are obvious: sections of paths performed reliably, variable pauses before complex sequences, and the overall appearance of uncertainty.

9.2.2 Garden

Overview

A similar setup to the lobby is used at the garden side of the same level. The task is to follow a ledge until the staircase descending the other way can be taken. This is followed down through 90 degree turn, after which the target is just on the right.

Candidates

1) The reactive bot is used again, and is expected to perform in the same way as for the previous trial.

2) A deliberative bot is also utilised. It was allowed to explore that area, and when necessary shown around reactively.

Figure 9.3: Picture of the staircase that leads down to the garden.

Description

The reactive bot generally manages to get downstairs like in the previous trial. However, on some occasions the first turn proves lethal; the bot under-steers and falls off the ledge. A simple reactive strategy from then on allows it to find the target. This demo was chosen to evaluate the bot's mistakes and recoveries.

The first few steps of the deliberative bot are somewhat uncertain as it finds itself on a narrow turning path. Once on the straight portion, things smooth out and the bot has no trouble engaging the staircase. The rest of the way down is also no problem.

Results

In *Figure 9.4*, the bars show that the reactive bot scores around 2.2 in realism, while the deliberative bot reaches 2.8. However, when it comes to the "Is it human?" question, around 20% of the judges voted affirmatively in either case.

Analysis

The results regarding the reactive bot are somewhat predictable based on the previous trials. While falling off the cliff can be expected from human players now and then, the judges singled this out as looking artificial. Again in the context of a game and not a formal evaluation, this would no doubt be satisfactory.

Judges marked the deliberative bot as being artificial purely because of the "epileptic" oscillations in the narrow path. Most confirmed that they would have been fooled if it weren't for those initial few seconds. This happens to be one of the infrequent cases (discussed above) where the bot achieves its reactive task without realism. Once identified, the designer can take active measures to prevent problem — but only then.[1] This procedure is characteristic of the entire project, and will be scrutinised further in the discussion.

[1]This problem was actually fixed for subsequent trials.

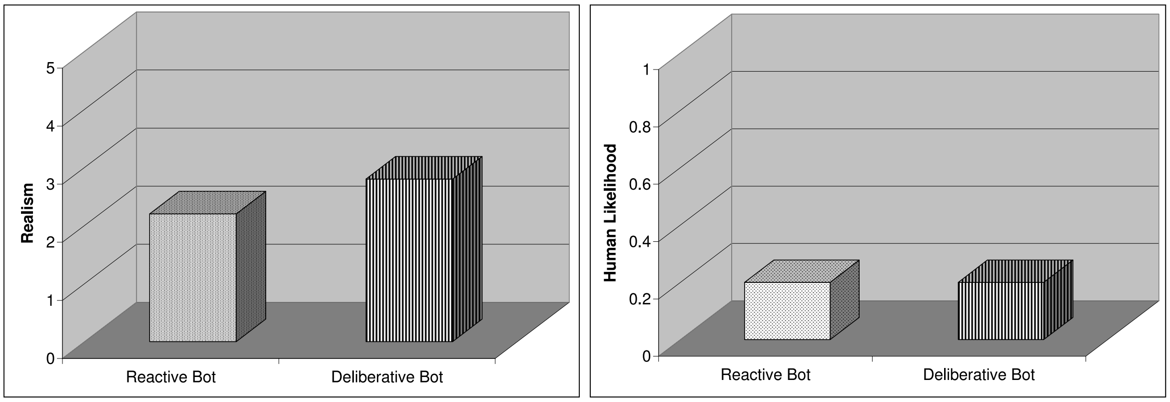


Figure 9.4: Differences in judging between reactive and deliberative bots. The second is slightly more realistic, but both remain visibly artificial.

Figure 9.5: Start and end sections of the cellar trial.

9.2.3 Cellar

Overview

This is a showcase for increasingly deliberative behaviour requiring knowledge of the terrain. The players start down in the "cellar," near the rocket launcher. They must then find their way up the stairs around the "lobby," through the door and around the corner towards the chaingun.

Candidates

1) The first candidate is the deliberative bot, who once again was allowed to reactively explore the area. When necessary, the developer accompanied it to demonstrate the desired start- and end-point of course.

2) The human expert was also asked to perform the same task.

Description

The bot does a really good job of exploiting the waypoints it laid down. Finding its way out of the seller is trivial and engaging the staircase causes no problems. The corners in the staircase are actually cut, saving the bot a few seconds. Once through the doorway, the small set of stairs near the chain gun is infringed slightly to cut the corner.

The human expert also performs in a very similar way. The corners are, however, not cut as much yet still maintaining extremely smooth movement.

Results

The barcharts in *Figure 9.6* reveal a very similar score for both candidates. The human expert scores slightly higher, achieving 4.4 average realism vs 4.2 obtained by the deliberative bot. As for the votes, both achieved very similar scores with around 40% of judges believing each of them was artificial.

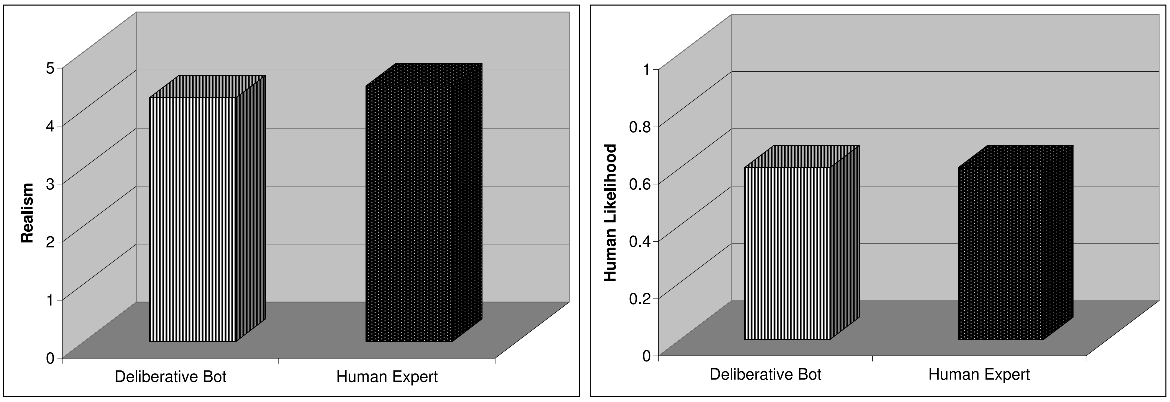


Figure 9.6: The human expert and the deliberative bot match up, with the edge attributed to the biological entity.

Analysis

This trial is fascinating for yet another reason. Both performances are extremely similar and could be classed as flawless — depending on the definition of "flawless." Therein lies the revelation of this demo; the subjectivity of the judging is often the deciding factor. In this case, part of the audience considered cutting the corner near the stairs a routing problem, while others thought it was extremely humanlike.

Some gamers considered both routes unrealistic from a strategic point of view, as the ledge is extremely vulnerable to snipers, and running into the dead-end near the chaingun is extremely risky too.

9.2.4 Atrium

Overview

This is a demo of a player finding his way through a crowd of 63 primitive bots. The start is at one side of the warehouse near the atrium. The player must find a way to the shotgun at the other side. As instructions, they are told to avoid the shorter route on the same floor, and prefer the conveyor-belt side of the basement. They must do this in both directions.

Candidates

1) This time the human with intermediate skills goes first.

2) The deliberative bot gets second spot, using a very similar setup procedure as before.

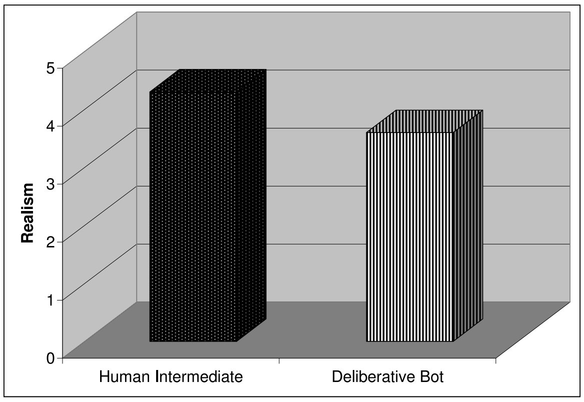
Description

The human has a relatively easy first section; there are few opportunities for collision with bots. Towards the end, two clashes are narrowly prevented, but one holds him back for barely a second. On the return journey, one small collision happens in the stairs but the clump of bots at the base is well circumnavigated. Along the conveyor belt, the human manages to anticipate the bots and avoid them, but a group of three bots blocks the path temporarily before he can continue.

The artificial player starts badly with a small collision in with a bot in the narrow corridor at the top of the stairs. A strange swerve happens at the bottom of the stairs, but it remains smooth and the path is still acceptable. A group of four holds the bot back around the conveyor belt for a few seconds, but the 8 others are avoided successfully. A narrow collision up the second stairway is prevented on short notice. On the way back, the bot is much luckier and has a smooth ride straight to the endpoint.

Results

The intermediate human player obtains a better realism score of 4.3, while the bot achieves around 3.6 on average, as shown in *Figure 9.7.* Fewer judges, in fact only around 20% thought the bot was human, compared with 60% for the human player.



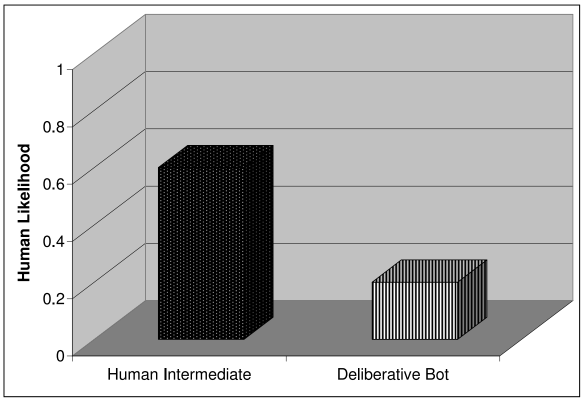


Figure 9.7: Judging human vs. bot in a dynamic populated environment. The impact of unrealistic non-navigational behaviours take their toll.

Analysis

This set of demos emphasises another extremely important issue; not only the navigation is taken into account during a trial. Admittedly, there are some specific behaviours that allow the bot to be distinguished, but many of the judges based their decision on the fixed turn at the midpoint. The human player did that in three parts, presumably visually exploring the environment, while the bot used one smooth scan.

To that end, the apparent failure of our bot is somewhat diminished, despite performing almost as well as the human avoiding bots and collisions.

9.2.5 Factory

Overview

In this trial, each player is given about 34 seconds of free time in a factory level to demonstrate its navigation abilities. Since there are no strict required elements, the bots must be judged on technical stills and artistic interpretation of the terrain.

Candidates

1) Gladiator Bot — As described in the background review of *Chapter 2*, this is arguably known as the best Quake2 bot to date, and became a commercial implementation for another popular game.

2) The human expert is there as a benchmark.

3) Eraser Bot — The first truly competent bot for Quake2 has many fans, though it's starting to show its age.

4) Our own deliberative bot is tested last to compare its abilities to other bots as well as the human in a freeform setting.

Description

The gladiator bot shies away from two staircases, seemingly changing its mind or not being aware of their presence. While human players do tend to step around corners, this bot tends to glide excessively. Towards the end, a seemingly voluntary jump is made in two stages from the first floor down to the basement. The bot loses quite some health in the process.

The human's movement comes across as really smooth, deliberate and efficient. There are no collisions and many items are collected.

The Eraser bot has a relatively rectangular path, until it falls off the ledge. Ending up on top of a crate, it ends up stuck there for almost ten seconds. Thereafter it resumes to a normal item collecting routine.

Finally, our own deliberative bot has a slight tendency to drift left, due to the characteristics of the obstacle avoidance behaviour. This causes the bot to hug walls, excessively infringing on small edges. Only select items are actually picked up despite the bot touching them, since it already had full health and armour.

Results

The scores of the final test are revealed by *Figure 9.8*. Faith in the judging system is confirmed by the human player scoring very highly in both metrics: 4.5 and 100% respectively. Both third party bots perform equally poorly for varied reasons, scoring 1.8 and 0%. Our own deliberative bot achieves slightly better with an average of 3 realism and barely 20% of guesses as human controlled.

Analysis

The fact that everyone managed to recognise the human which reveals that such a free-form test is an appropriate setting. With the expert also achieving maximum realism, this confirms the scale used by the judges as adequate.

The particular instance of obstacle avoidance behaviour of our deliberative bot was evolved just before the trials to prevent the oscillatory problem cited above. This proved to have a small flaw which sadly hampered the realism of the bot. However, none of the other bots were tweaked and tuned before the run, so it seemed fair not to pay any particular attention to our solution.

Despite the small problems, our system achieved the runner up score for realism, and also fooled the second most judges into believing it was human (after the expert).

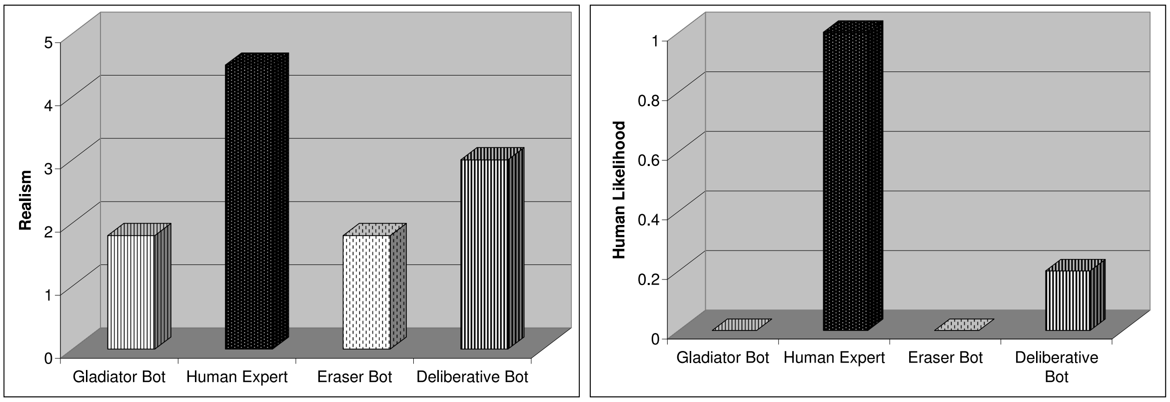


Figure 9.8: Comparing the perceived realism of all four players and their likelihood of being human based on the judges votes.

9.3 Modular Analysis

While an in-depth empirical analysis[2] of each component of the navigation system would prove extremely enlightening, this falls beyond the scope of our project (mainly due to the time constraints of developing a full system). However, a brief discussion can nonetheless establish the major benefits and pitfalls of each module, in view of extending them to a formal study as the system witnesses further improvements.

9.3.1 Motion Controller

Advantages

Neural networks are extremely well suited to this kind of motor control. When trained correctly, they offer an extremely smooth navigational policy. Despite such reactive behaviours requiring foresight (or simply a broader horizon) to be realistic, our model does a good job of faking this in most cases.

Evolution allows the creation of working models extremely simply, using very few fitness parameters. EANN provide a reconfigurable framework, which can be applied to creating pretty much any reactive behaviour with few changes.

Disadvantages

It can take a bit of time to understand and patch up loopholes in fitness functions, but once that is done one can truly focus on the major issue: tweaking the parameters of the various equations. The first developer can assist others by providing higher-level concepts that affect these behaviours, but third parties will still have to experiment for long enough to be able to generate the desired results. At this point, it might have been easier to devise explicit mathematical models instead!

Genetic algorithms do suffer from a variation in the performance during lifetime learning, which could be fixed by using alternative optimisation techniques that only rely on scalar feedback values — such as reinforcement learning. This would probably exhibit a slightly more consistent performance increase.

There are also issues due to the limited perception of the module; humans can navigate reactively with more anticipation due to their ability to process visual information. This system does not even belong in the same league, as its horizon is very limited. While this is something higher levels of intelligence can simulate, it would be interesting to compare both reactive models on the same grounds. On the similar note more complex time dependent patterns would require something else but a simple perceptron.

[2]This was done somewhat informally during the development phase.

9.3.2 Reactive Behaviours

Advantages

Purely from the point of view of animat navigation, the parasitic approach fairs well — despite the simplicity of the model. The resulting behaviours are skilled, and convincing enough to be human-like. Additionally, the approach is straightforward for developers thanks to learning (compared to even simple arbitration schemes) and for end users (since all they have to do is tweak the fitness components).

Technically speaking, the numerous benefits for machine learning should also be acknowledged — since the field is emerging as one of the most capable for navigation purposes. Firstly, the search space finds itself greatly reduced; both the host and parasite can be treated as completely separate problems, and the parasite itself can be abstracted as a modular solution in nature. Secondly, this lends itself very well to the incremental design approach, whereby the solution is improved module by module over time. Since the machine learning can rely on the "frozen" components, it performs much better than any dynamic approach.

Disadvantages

Despite the concept of artificial evolution being extremely useful from some aspects, it causes some serious issues with the quality of the online learning. Indeed, when a genetic algorithm is used, it is tough to guarantee any kind of base performance — due to the manner in which evaluation is done in trials. Any solution that guarantees reasonable fitness would most likely suffer from premature convergence.

Although the process of tweaking behaviours is simpler using a high-level fitness function than having to rewire the underlying implementation, it can still be criticized. It does admittedly take time to get used to the process of adjusting the behaviours, but this process is generally done once only. Together with this, the default values provided in this dissertation are a very good starting point for further experimentation.

Figure 9.9: Amidst the crowd in the atrium trial.

9.3.3 Topography Learning

Advantages

The solution proves generally useful and generates good quality maps, despite their coarseness. Node removal and edge rotting allows them to remain valid over the course of time.

Simulating spatial commonsense with full embodiment was established as somewhat of challenge but remained worthwhile due to its many benefits. Together with node position optimisation, it crafts a good guide that links into the reactive behaviours extremely well.

Disadvantages

While the graphs learnt are fairly coarse, they remain somewhat more precise than the terrain models that humans have been empirically shown to use. This is due to the fact that our reactive model is not as well developed, and needs additional assistance.

Obstacles are also not stored explicitly; this causes many problems as the system cannot assume much from the absence of nodes and edges in the topography. As a consequence, the exploration process is affected. It cannot be guided by the intrinsic nature of the terrain representation.[3]

9.3.4 Pathematics

Advantages

The solution itself proves very powerful in practice. The routes it can generate combine paths and goals successfully, taking into account complex desires of animats. With regards to efficiency, our approach also fares well. The heuristic process can return at any point, while its greedy nature leads to good initial solutions.

Disadvantages

Exploiting the algorithm to its full potential requires reinterpreting the higher-level requests onto practical parameters. These can change the input conditions of the graph (connections and distances), as well as the reward setting process. This is a nontrivial task since there are usually many ways of executing the orders. Notably, determining if the avoid behaviour should correspond to a reward scaling or a distortion of space is still an open question.

While the incremental dynamic nature of the heuristic process can cause problems, these can be prevented once identified, although they need to be acknowledged. Small precautions can be taken to make sure they do not affect the behaviour in a negative fashion. For example, this includes keeping the distance estimates around the root node as accurate as possible, and making sure there is no residue reward left in previous locations.

9.4 Global Impressions

This section looks at the trials, analysing how the results reflect on the design of the entire system.

9.4.1 Realism

The solution devised does perform surprisingly well with regards to realism. Human-like behaviours can be generated in most cases, though they seem to collapse in occasional situations that the designer has not been able to test. The evolved behaviours assure that the goal is generally still achieved, but not in the intended fashion.

This realism problem is tough to fix, reminiscent of programmed behaviours in standard game AI. Scripts need to be understood and debugged in many situations in order to guarantee that they will perform as expected. With our learning solutions, short cuts cannot be taken either; the emergent "features" of the learning need to be identified and remedied when appropriate.

[3] Although it may be possible to devise an algorithm or additional modifications to the current format.

9.4.2 Reliability

The system generally proves reliable, using the parameters provided throughout this dissertation. Specific checks for consistency need to be made after the behaviours have been evolved; this small debugging time needs to be taken into account in the development process.

Thanks to the evolution and the selection of the fitness function to reward achievement over manner, the behaviours tend to be very reliable. This has a negative effect on realism in some case, as discussed. A longer evolution period can generally remedy this.

With regards to requests in specific terrains, the topography and the planning to be exploited fully, guaranteeing the desired result. However, the animats cannot perform reliably until the terrain has been explored sufficiently. Granted, the reactive behaviours can in most cases lead to the chosen destinations, but not in arbitrarily complex layouts. Reactive arbitration of the behaviours cannot do this either, so a minimal amount of deliberative processing is required. This would no doubt take the form of a short stack-like history of planar constraints that need to be satisfied, which would guide the animat out of mazes. Alternatively, the learning of the topography could be integrated with the reactive behaviours, allowing similar exploration or guidance policies until the target is reached.

9.4.3 Dynamism

The system performs extremely well dynamically, since multiple levels of the architecture support dynamic conditions. The lower-levels take into account a constant stream of information from the surroundings reactively, and the higher-levels adapt the plans based on the changing conditions of the environment.

However, there is too little feedback between the reactive behaviours and the planning. Currently, two things cause plans to change based on their execution: the current edge length increases as the animat takes more time to get to its destination, and the likelihood to chose the same path again once the animat has turned around is scaled by 0.25. However, this does not allow the animat to persevere, waiting to force its way through a temporary hazard (currently it'd just turn around and use another path). A certain amount of feedback from the lower-levels, and analysis of the benefits of waiting should be performed.

9.4.4 Summary

In the process of developing our complete navigation system, some interesting discoveries were made, especially with regards to the synthesis of human-like behaviours.

- Levels of Dynamism — Each level of the architecture needs to be dynamic for complex situations to be handled correctly. A strong sense of feedback is needed for the coordination of the levels, and the coherence of the entire system.

- Heuristics and Reactivity — Components should not be allowed to slow down the system for efficiency and responsiveness reasons. While programming tricks can be used to convert deliberative searches into incremental procedures, our research shows that specialised algorithms can be devised with heuristics, or simply on a fully reactive basis.

- Complete Learning — Getting reliable behaviours requires a thorough evaluation of the fitness in as many situations as possible. This is not something pre-evolution handles perfectly, as trials need to be finite, and the chances of all situations being covered are extremely unlikely. Rare situations are often encountered during a lifetime, requiring adaptation — or a helping hand from the designer.

- Incremental Design — Intensive human design and debugging is necessary for realistic behaviours to be provided. Learning tends to create reliable motion when left to its own devices, but realism is something that requires a tremendous amount of time and effort — even more so than just reliability!

- Request Interpretation — The commands passed to the navigation system via the higher-level interface need to be interpreted in a coherent fashion. This is required both for the sake of obtaining the desired result reliably, and in a realistic fashion.

While these are deductions that can be drawn from the analysis of the trials, there are more to be extracted from the experience of developing the system itself. These will be given in the retrospective overview of *Chapter 10*.

Chapter 10

Conclusion

Navigation systems are extremely interesting to research. The project has been equally challenging to develop, and was fascinating to watch unfold. Despite still having a long way to go, much has been achieved and a solid base has been established.

10.1 Retrospective Overview

Looking back at the entire project, and analysing it with regards to the aims defined in the introduction, how well does the system perform?

10.1.1 Interface Design

The loose definition of navigation as solving problems in space does allow a great improvement of the interface specification. Instead of specifying target points in space, the user describes the whole problem using rewards and costs. By providing abstract functions, this works extremely well in practice, and remains a simple interface.

Developing the system around the interface was surprisingly smooth; in most cases, the path-planning handles all the rewards and costs, and controls the lower-level motion around on request. When a topography is not present, requests are only mapped onto reactive desires and targets.

However, it seems a problem with the interface is the lack of natural language understanding. Admittedly, this is not fundamentally part of navigation, but the major part of the source code used to create the animats in the demonstration was in fact primitive NL processing — allowing the designer to interact and experiment with the behaviours. If this system was to be improved, NL would be the next step forward.

10.1.2 Complete System

The project has managed to develop a substantial working system, however it still falls short of being complete. Much of the requests given via the interface can be satisfied, but some intricate situations cannot be handled by the architecture. Alternate solutions are generally found, for example going around the long way instead of taking the ladder, but this solution is not always an option.

That said, it would only take a small amount of work to extend the system to handle infrequent cases like ladders or platforms, and given the existing framework these could be integrated fairly simply.

10.1.3 Flexible Architecture

The architecture defined allows navigational architectures to be crafted with relative ease. The skeleton needs to be established, and learning components can be inserted arbitrarily. Our specific architecture is relatively reconfigurable, though it becomes less customisable towards the upper-levels (since it is not really required).

However, most of the work remains in the domain of programming. Such a fastidiously monotonous task can be greatly simplified by the development of a user interface which allows components to be laid out and connected in a visual fashion.

10.1.4 Real-time Solution

As far as efficiency goes, the system does well considering the premise and requirements; dynamic navigation is performed autonomously in a fully embodied animat. The solution is fast enough to simulate many of these animats on low-end consumer hardware, and many algorithms can be customised to have minimal overhead (memory and computation).

However, in the short term, it seems unfeasible to have the full panoply of features enabled, as well as having multiple such agents in a modern computer game. When barely 5% of the budget can be utilised for AI, many corners need to be cut, and navigation would involve mostly static plans and no obstacle avoidance.

10.1.5 Embodiment

There should be no denying it; implementing a fully embodied animat is not a minor task. Developing the interface with its environment is something that can be done once and for all, although the framework used defines flexible standards that can be developed incrementally in a backwards compatible fashion. On top of this, the overheads of designing an architecture capable of coping with such restricted information can not be neglected either.

However, there is little doubt that such constraints improve the realism and genuine feel of the animat. Most human mannerism are implicit consequences of their embodiment, and simulating this body allows us to obtain a wide range of desired properties. Specifically for navigation, path smoothing is no longer a problem, as the simple concept of momentum does a tremendous job.

10.1.6 Summary

In brief, there are some things to note about the development aspect, with regards to targeting complete human-level navigation systems.

- Interface Design — The top down specification of the system by designing the higher-level interface first was extremely helpful during the development. This allowed the definition of the underlying structures, and behaviours using these structures. When the system is incapable of performing a task, an additional function is added to fill the gap and provide complete functionality.

- Completeness — The development of a complete system revealed the inconsistencies in many existing separate solutions, and the amount of work required to fit everything together. Making the system easily extendible also proves important to handle novel situations that were not in the original design.

- Reconfigurability — Designing a complete system revealed the need for flexibility. Behaviours, functions, data-structures and their interaction change so often over the course of the design that a really universal framework shines through. Only when a reconfigurable architecture smoothes this process, which should even be as intuitive as possible.

- Efficiency — Programming efficient behaviours is a matter of selecting or designing the right solutions in the underlying implementation. Heuristics and reactive approaches should be preferred, and constraints should be set on memory and processor consumption when necessary.

- Embodiment — This is a great way of obtaining genuine behaviours; when done in a biologically inspired fashion and develop the underlying representations in a suited fashion, extremely interesting emergent behaviours arise.

This project has only been moderately successful in these tasks, but yes has mostly striven in the right direction and identified the need for such approaches.

10.2 Discussion

This section raises various issues with specific details of the thesis, and underlines why they are problematic in general.

10.2.1 Fitness Functions

It seems this project has reached the epitome of fitness functions. Granted, very simple ones can be employed to obtain good behaviours, but relatively complex scripts are required to obtain human-level navigation. The functions describe an intricate fitness space with hints, details, and special cases; it has become so elaborate that almost the entire problem has thereby been modelled. This takes meticulously time-consuming iterative development, which negates the whole point of using a learning approach. The solution might as well be crafted manually.

10.2.2 Evolution & Believable Agents

While this may seem a good idea at first, artificial evolution is no more likely of generating realistic behaviours than any other AI optimisation technique. Making something behave in a humanlike fashion with an EA is a matter of modelling the body as precisely as possible, and letting the fittest survive. This was not done here, and will probably take another couple of decades to be even thinkable.

In some contexts, this project succeeded in creating humanlike behaviours with evolutionary solutions, but this should not be attributed to them in any respect. In fact, the key component was meticulous tweaking of the fitness functions, and expert design of the models.

10.2.3 Expert Design

Much of AI development nowadays relies too heavily on expert knowledge. While this is something that some veterans advocate, it does not really sink in to the extent it should until it's too late. Admittedly learning techniques are often used, but rarely is the design of AI systems placed in the hands of the AI.

The same mistake was made here. The few learning algorithms introduced here were mostly convenient from an external point of view, allowing behaviours and factual knowledge to be assimilated. Still too much work was left in the hands of the designer.

This could have been greatly simplified by the use of imitation instead of fitness functions, or using some sort of structured evolution to organise the internal architecture in an incremental fashion.

10.2.4 General Applicability

All along this dissertation, many parallels with fully functional agents were present — some more obvious than others. Fundamentally, there is much overlap between the study of agent architectures and navigation.

Firstly, we believe that the parasitic approach would have mixed success when applied to generic agents. For motor control, they remain an extremely useful tool capable of providing sophisticated blended behaviours in the very realistic fashion. However it remains to be shown if they can handle symbolic representations; we expect them to work in some situations but prove more problematic in others.

Moving onto the learning of the terrain, the procedure would apply extremely well to learning facts about life. Spatial common sense would be replaced by common sense (defining how one can get to a specific situation from a current state), and the algorithm would coarsely learn relationships between situations. Naturally, robust reactive behaviours would again be relied upon. There will also need to be a few adjustments to take into account the assumptions made for navigation, and also to deal with the increase in the size of the animat's state.

Finally, with regards to Pathematics applied to generic problem solving, we believe it would also perform surprisingly well. Combining many different desires is also a requirement in life, and the many other properties of our solution desirable in navigation would remain desirable in any implementation.

That said, all this needs to be substantiated!

10.3 Future Work

We would like to investigate the "emergent capabilities" of the system within a multi-agent framework. This would involve developing the natural language interaction with the system, and potentially investigating social learning algorithms.

From a scientific point of view, each of the modules developed really need to be evaluated in their own rights. For a technical point of view, aside from polishing the system and extending its capabilities, we would like to improve the following aspects.

10.3.1 Topography Learning

While the framework supports things such as node *removal* or edge *deletion*, it would be extremely interesting to apply these procedures to create more humanlike struggles with map learning. Beyond just forgetting places and path segments, this would require graceful deterioration and habituation. For example, humans can remember paths segments as a whole, but forget little details. They have to re-explore the location to rebuilt the "model" of the path. Graceful deterioration would allow such memory losses, and allow surprised to be expressed when it is once again discovered.

We'd also like to develop the localisation algorithm further. Moving beyond identifying key objects within the field of view, the system requires further sensory analysis to solve the "kidnapped animat" problem. By using a heuristic statistical matching of numerous visible objects, tracked probabilistically over the course of a few seconds, the animat should be able to localise itself. Using active exploration to test localisation theories, interesting behaviours should arise.

10.3.2 Pathematics

Though our approach performs well for agents in large terrains (outperforming existing implementations), we do not expect it to scale to huge graphs. Like humans, the cognitive process of agents is often limited, and applying it to a consequent data-structure would give poor results. In this case, we believe the best improvements would be incurred by modelling the terrain using a more human-like approach, with a hierarchy. Our existing algorithm would need to be adapted to suit this new paradigm.

The current implementation is suited to single agents that intend to plan their own movement. Path queries about other agents may be required for anticipation purposes. While our design would lend itself well to simulating multiple agents separately, some benefit may be gained by merging these computations, moving a step closer to dynamic *all the pairs* algorithms.

Throughout the paper, associations of path-planning with decision making have been noted. At times, it has taken some restraint to remain within the domain of navigation. We believe an adapted version of this algorithm would be ideally suited to human-like deliberation, though the specification of the problem would have to be redefined.

10.4 Final Word

As a navigation system, ours is usable in its current state. It needs to be expanded slightly to support the more advanced contraptions of the environment (like ladders), but our flexible framework allows this easily, and it would be just a matter of time.

The realism of the system is reasonable in most situations, though it does require extensive monitoring and tweaking to iron out the idiosyncrasies. This, once again requires spare time and effort. Despite being an improvement over existing solutions, the motion remains shy of human-level realism.

The research involved creating a complete system, which revealed major issues. Essentially, creating human like behaviours is something that can be done by extensive expert design, but there should be more emphasis placed on higher-level learning — allowing the system to learn almost entirely what to do and practically design itself.

Appendix A

Flexible Behaviours

A.1 Obstacle Avoidance

```python
def Avoid():

        # global variables
        global r_turn
        global r_brake
        global g_collision
        global g_fitness

        # remove symetry to reduce search space size
        g_inverted = 0
        if x_symetric and s_distances[0] > s_distances[s_sensors-1]:
                # swap the values around
                s_distances.reverse()
                r_turn = -r_turn
                # remember we did this!
                g_inverted = 1

        # build up input array
        s_distances.append(r_turn)
        s_distances.append(g_collision / 2.0)

        # fetch the result by simulating the perceptron
        output = avoid_turn.simulate( s_distances )

        # check if value is too big
        if output[0] > x_clamp:
                output[0] = x_clamp
        if output[0] < -x_clamp:
                output[0] = -x_clamp

        # flip back if necessary
        if g_inverted:
                output[0] = -output[0]
                r_turn = -r_turn

```

```
        # collision penalty accumulates
        fitness = 0.0
        if s_collision:
                g_collision += 1
                fitness -= x_collision * g_collision
        else:
                # reset collision counter
                g_collision = 0

        # reward lazzy behaviour, penalise big increments over small changes
        fitness -= Squared( abs(output[0]) * x_change )
        fitness -= Squared( abs(output[0]) * x_trouble * (1-s_distances[1]) )
        # reward persistence of movement
        fitness -= Squared( abs(r_turn - output[0]) * x_persistence )
        # oscillation penalty
        if r_turn * output[0] < 0.0:
                fitness -= x_oscillation * abs(r_turn - output[0])

        # report the fitness to the turn component
        avoid_turn.fitness( fitness )
        g_fitness = fitness

        # output turning direction
        r_turn = output[0]
        r_brake = sqrt(s_distances[1])

        # commands passed back implicitly via r_* variables
        return
```

A.2 Wall Following

```
def Follow():
        global r_turn
        global r_brake

        # concatenate previous decision as extra input
        s_distances.append( r_turn )
        # fetch the result by simulating the perceptron
        output = follow_steer.simulate( s_distances )

        # check if value is too big
        if output[0] > x_clamp:
                output[0] = x_clamp
        if output[0] < -x_clamp:
                output[0] = -x_clamp

        # squares penalise big increments over small changes
        fitness = -Squared( abs( output[0] ) * x_change )
        # reward the presence of a wall left
        if s_distances[0] > 0.0:
                fitness += x_presence
        fitness += (0.5 - abs( s_distances[0] - 0.5 )) * x_proximity
        # punish collision with wall
        fitness -= Squared( x_collision * s_distances[1] )
```

```
        # and punish excessive oscillation
        if r_turn * output[0] < 0.0:
                fitness -= x_oscillation * abs( r_turn - output[0] )

        # report the fitness to the turn component
        follow_steer.fitness( fitness )

        # turning direction passed back implicitly
        r_turn = output[0]
        r_brake = 0 #s_distances[1]

        # commands passed back automatically via r_* variables
        return
```

A.3 Seeking Behaviour

```
def Seek():

        global g_previous
        global g_fitness

        # consistency check
        if len( g_previous ) != len( s_distances ):
                g_previous = [0 for i in s_distances]

        # preprocess angles    into another python sequence
        copy = []
        min, max = 100, -100
        for a in s_angles:
                # store angles relatively to desired orientation
                relative = abs(a / 90.0 - s_orientation)
                copy.append( relative )
                # and isolate the two extremities
                if relative > max:
                        max = relative
                if relative < min:
                        min = relative

        # compute the average of the distances
        average = Max( s_distances ) * Average( s_distances )

        fitness = 0.0
        # process each of the sensor values one by one
        for i in range( len(s_distances) ):

                # build up array with selected inputs
                input = [ abs((copy[i] - min) / (max - min)),
                                s_distances[i],
                                g_fitness,
                                abs(s_orientation),
                                g_previous[i] ]
                # let the neural network compute the desired change
                output = seek_adjust.simulate( input )
```

```
                # actually update the distance, and store for next time
                s_distances[i] += output[0]
                g_previous[i] = output[0]

        # finally, punish for diverging from target
        fitness -= abs( s_orientation ) * x_divergence

        # report the fitness value back
        seek_adjust.fitness( fitness + g_fitness )

        # and return modified distances
        return s_distances
```

Appendix B

Organ Specifications

```
class Motion : public Organ
{
public:
        void Move( const Vec3f& direction );
        void Turn( const Vec3f& orientation );
        void Jump();
        void Duck();

}; // class Motion


class Physics : public Organ
{
public:
        bool Collision();
        bool isPlatform();
        bool isLadder();
        bool isWater();
        bool isAir();

//----------
//  Odometry
//----------
        void Movement( Vec3f& m );
        void Orientation( Vec3f& o );

}; // class Physics


class Vision : public Organ
{
public:
        const std::vector<Entity>& FetchVisibleEntities() const;

        float CheckForObstacles( const float angle, const float steps ) const;

}; // class Vision
```

Bibliography

Anonymous (1997). Reaper bots don't cheat! http://www.mindspring.com/ win32ch/reaper.htm.

Arkin, R. C. (1998). *Behaviour-Based Robotics*. The MIT Press.

Arkin, R. C. and Balch, T. R. (1997). Aura: Principles and practice in review. *JETAI*, 9(2-3):175–189.

Bellman, R. (1958). On a routing problem. *Quarterly Applied Mathematics 16*, pages 87–90.

Brooks, R. A. (1991). Intelligence without representation. Number 47 in Artificial Intelligence, pages 139–159.

Burgard, W., Cremers, A. B., Fox, D., Hahnel, D., Lakemeyer, G., Schulz, D., Steiner, W., and Thrun, S. (1998). The interactive museum tour-guide robot. In *AAAI/IAAI*, pages 11–18.

Burgess, N., Recce, M., and O'Keefe, J. (1994). A model of hippocampal function. pages 1065–1081.

Castillo, P., Merelo, J., Prieto, A., Rivas, V., and Romero, G. (2000). G-prop: Global optimization of multilayer perceptrons using gas.

Champandard, A. J. (2001). A server client based approach to ga class design http://geneticalgorithms.ai-depot.com/tutorial/ga-class.html.

Champandard, A. J. (2002a). The dark art of neural networks. *AI Game Programming Wisdom*.

Champandard, A. J. (2002b). Return to castle wolfenstein: Ai analysis http://ai-depot.com/gameai/wolfenstein.html.

Connell, J. H. (1992). Sss: A hybrid architecture applied to robot navigation. In *ICRA - IEEE Conference on Robotics and Automation*, pages 2719–2724.

D. E. Rumelhart, G. E. Hinton, R. J. W. (1986). Learning internal representation by error propagation. *Parallel Distributed Processing*.

Dial, R. B. (1969). Algorithm 360: Shortest path forest with topological ordering. *Communications of the ACM 12*, pages 632–633.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerishe Matematik*, 1:269–271.

Epic (2000). Unreal technolgy http://unreal.epicgames.com/.

Fear (2002). Official website http://fear.sourceforge.net/.

Feltrin, R. (1998). Eraser bot http://impact.frag.com.

Ford, L. R. (1956). *Network Flow Theory*. Rand Co.

Gallo, G. (1980). Reoptimization procedures in shortest path problems. *Rivista di Mathematica per le Scienze Economiche e Sociali 3*, pages 3–13.

Gat, E. (1992). Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *National Conference on Artificial Intelligence (AAAI)*, pages 809–815.

Goldberg, A. V. and Radzik, T. (1993). A heuristic improvement of the bellman-ford algorithm. *Applied Mathematics Letters 6*, pages 3–6.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley.

Gruau, F. (1994). *Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm*. PhD thesis.

Gruau, F. (1995). Automatic definition of modular neural networks. *Adaptive Behaviour*, 3(2):151–183.

Hans Blaasvaer, P. P. and Christensen, H. I. (1994). Amor: An autonomous mobile robot navigation system. In *IEEE, International Conference on Systems, Man, and Cybernetics*, pages 2266–2271.

Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems*. The MIT Press.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proc. Natl. Acad. Sci. USA*, 79:2554–2558.

Husbands, P. (1998). Evolving robot behaviours with diffusing gas networks. In *EvoRobots*, pages 71–86.

id Software (2001). Quake 2 source code ftp://ftp.idsoftware.com/idstuff/source/q2source-3.21.zip.

Johnson, E. L. (1972). On shortest paths and sorting. In *Proc. 25th ACM Annual Conference*, pages 510–517.

Konolige, K., Myers, K. L., Ruspini, E. H., and Saffiotti, A. (1997). The Saphira architecture: A design for autonomy. *Journal of experimental & theoretical artificial intelligence: JETAI*, 9(1):215–235.

Laird, J. (2001). Personal communication about the soar quakebot's navigation.

Laird, J. E. (2000). An exploration into computer games and computer generated forces. *Computer Generated Forces and Behavior Representation 8*.

Laird, J. E. and Duchi, J. C. (2000). Creating human-like synthetic characters with multiple skill levels: A case study using the soar quakebot. *AAAI 2000 Fall Symposium Series: Simulating Human Agents*.

Liu, B. (1996). Intelligent route finding: Combining knowledge and cases and an efficient search algorithm. In *European Conference on Artificial Intelligence*, pages 380–384.

Marom, Y. and Hayes, G. (2001). Interacting with a robot to enhance its perceptual attention. In *Towards Intelligent Mobile Robots*.

Martins, E., Rasteiro, D., Pascoal, M., and Santos, J. (1999). The optimal path problem. *Investigac ao Operacional*, 19:43–60.

Mitchell, M. (1998). *An Introduction to Genetic Algorithms*. The MIT Press.

Mohammadian, M. and Stonier, R. J. (1995). Fuzzy logic and genetic algorithms for intelligent control and obstacle avoidance. *Complexity International*, 2.

Moore, E. F. (1959). The shortest path through a maze. In *Proc. International Symposium on Theory of Switching*, volume 2, pages 285–292.

O'Keefe, J. and Nadel, L. (1978). *The hippocampus as a cognitive map*. Oxford: Clarendon Press.

Page, L., Brin, S., Motwani, R., and Winograd, T. (1998). The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project.

Pape, U. (1974). Implementation and efficiency of moore-algorithms for the shortest route problem. *Mathematical Programming 7*, pages 212–222.

Polge, S. (1996). Reaper bot ftp://ftp.cdrom.com/pub/quake/quakec/bots/reaper.

Reece, D., Krauss, M., and Dumanoir, P. (2000). Tactical movement planning for individual combatants. *Proceedings of the 9th Conference on Computer Generated Forces and Behavioral Representation*.

Reece, D. A. (1995). Soldier agents in a virtual urban battlefield. Technical report.

Reynolds, C. W. (1987). Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21(4):25–34.

Reynolds, C. W. (1999). Steering behaviors for autonomous characters.

Reynolds, C. W. (2000). Interaction with groups of autonomous characters.

Riedmiller, M. and Braun, H. (1993). A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proc. of the IEEE Intl. Conf. on Neural Networks*, pages 586–591, San Francisco, CA.

Ritchie, J. (1998). *An Artificially Intelligent Agent for the Computer Game Quake*. PhD thesis.

Rosenblatt, F. (1962). Perceptrons and the theory of brain mechanisms.

Rosenblatt, J. K. (1995a). DAMN: A distributed architecture for mobile navigation. In *Proc. of the AAAI Spring Symp. on Lessons Learned from Implememted Software Architectures for Physical Agents*, Stanford, CA.

Rosenblatt, J. K. (1995b). DAMN: A distributed architecture for mobile navigation. In *Proc. of the AAAI Spring Symp. on Lessons Learned from Implememted Software Architectures for Physical Agents*.

Rumelhart, D., Hinton, G., and Williams, R. (1986). Learning internal representations by error propagation. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, 1(8).

Saffiotti, A. (1997). The uses of fuzzy logic in autonomous robot navigation. *Soft Computing*, 1(4):180–197.

Schmajuk, N. A. and Thieme, A. D. (1992). Purposive behavior and cognitive mapping: a neural network model. pages 165–174.

Seymour, J. (2001). A flexible and extensible framework for modelling and simulating physical agents.

Simmons, R., Goodwin, R., Haigh, K., Koenig, S., O'Sullivan, J., and Veloso, M. (1997). Xavier: Experience with a layered robot architecture.

Stentz, A. (1995). The focussed d* algorithm for real-time replanning. In *IJCAI*, pages 1652–1659.

TALWAR, S. K., XU, S., HAWLEY, E. S., WEISS, S. A., MOXON, K. A., and CHAPIN, J. K. (2002). Behavioural neuroscience: Rat navigation guided by remote control. pages 37–38.

Thrun, S., Bennewitz, M., Burgard, W., Cremers, A. B., Dellaert, F., Fox, D., Hahnel, D., Rosenberg, C. R., Roy, N., Schulte, J., and Schulz, D. (1999). MINERVA: A tour-guide robot that learns. In *KI - Kunstliche Intelligenz*, pages 14–26.

Trovato, K. (1990). Differential a*: An adaptive search method illustrated with robot path planning for moving obstacles and goals, and an uncertain environment. *Journal of Pattern Recognition and Artificial Intelligence*, 4(2).

Trullier, O. and Meyer, J. (1998). Animat navigation using a cognitive graph.

Trullier, O. and Meyer, J.-A. (1997). Biomimetic navigation models and strategies in animats. *AI Communications*, 10(2):79–92.

van Waveren, J. P. (2001). *The Quake III Arena Bot.* PhD thesis.

Wan, H. S., Touretzky, D. S., and Redish, A. D. (1994). Towards a computational theory of rat navigation. pages 11–19.

Wilson, S. W. (1987). Classifier systems and the animat problem. *Machine Learning 2*, pages 199–228.

Worden, R. (1992). Navigation by fragment fitting: A theory of hippocampal function. 2(2):165–188.

Young, T. (2002). Pathengine library http://www.pathengine.com/.