Adding Vision to Khepera: An Autonomous Robot Footballer

Tom Smith
toms@cogs.susx.ac.uk
School of Cognitive and Computing Sciences
University of Sussex

Dissertation for MSc, Knowledge Based Systems
Supervisor Phil Husbands

March 25, 1998

Abstract

This project investigates the evolution in simulation of robot controllers capable of performing a *hard* task, playing football, in the real world.

It is argued that for any reasonably interesting task, robot controllers are too difficult to design, and that an evolutionary methodology must be employed. It is further argued that evolution in the real world is too slow for such a task, and that evolution in simulation must be used to produce good robot control structures. The techniques of *minimal simulation*, where the robot controller is forced to ignore certain features through making those features *unreliable*, are used to construct a simulated environment for a robot with vision system. A fixed architecture neural network provides a sensorimotor control system for the simulated robot, and evolution is used on an encoded version of the network to produce good solutions.

Two experiments are presented; finding a white stripe in a dark arena, and finding a tennis ball and pushing it into a goal. In both scenarios, good controllers capable of performing the same behaviours in simulation and in the real world, are evolved only once sufficient unreliability is incorporated into the simulation. The success in evolving in simulation a robot controller incorporating *distal* visual environment input data and displaying the same behaviours in both simulation and the real world, goes some way to addressing the arguments that evolution in simulation is only suitable for toy problems.

Acknowledgements

Above all I would like to thank my supervisor Phil Husbands for giving me the confidence, Nick Jakobi for showing me what a robot looks like, and Anil Seth and Matthew Quinn for much good advice.

Thanks also to all in the Masters Lab and Centre for Computational Neuroscience and Robotics for innumerable suggestions. Finally a special thanks to Nils Hulth for teaching anything and everything about computers.

Contents

List of Figures

Chapter 1

Introduction

"The question of whether a computer can think is no more interesting than the question of whether a submarine can swim."

E.W. Dijkstra

"Some people believe football is a matter of life and death. I'm very disappointed in that attitude. I can assure you it is much, much more important than that."

Bill Shankly

"When the seagulls follow the trawler, it is because they think that sardines will be thrown into the sea."

Eric Cantona

Robot controllers are difficult to design. The problems are considerable in the production of a control system robust enough to cope with the complexity and uncertainty of the real world. Research into the application of biologically inspired techniques has provided a possible solution, enabling the evolution of extremely complex robot controllers. However, opinion is divided on the issue of whether evolution in the real world, or evolution in simulation, is more suitable to the production of robust controllers. This project investigates one method of evolving robot controllers in a *minimal simulation*. The controllers are then transferred on to a real robot, where they must perform a complex task - playing football.

This project is aimed at addressing the major argument levelled against evolution of robot controllers in simulation, namely that such simulations suffer from scaling up failure and are only practical for small 'toy' problems. Recent work in the area of evolution in simulation [31, 32] has provided a potential answer to the arguments that simulation of a complex scenario will be unrealistic, or prohibitively time-expensive to construct. The theory of minimal simulation argues that the controller can be forced to ignore simulation features not present in the real world, through making those features *unreliable*. Real world controllers that successfully demonstrate the same behaviours in simulation and in the real world have been evolved in such simulations; this project is the first to attempt to evolve controllers capable of *on-board* processing of real visual data for a complex task requiring more than a single behaviour. The use of long distance, or *distal* visual information, as opposed to local proximity detector information such as infra-red sensor data, enables much more sophisticated navigation strategies to be utilised - this project evolves robot controllers displaying such strategies.

The task used in the project is football. It has been argued [39] that football should be a standard problem in robotics research, covering a wide range of domains from low-level mechanical and visual processing problems right through to group coordination and classical AI representation and planning issues. This project uses a simplified version of the game; the single robot must find the ball and push it into the goal. No other players are involved, but it should be emphasised that the task is *hard* by robotics standards. The robot must identify the ball, distinguishing it from the two goals, and keep tracking it while moving forwards, even when the ball fills its entire field of view (which will occur with the robot still some distance from the ball). In addition, the controller must be able to operate over a range of lighting conditions - controllers basing strategies on a single set of simulation conditions will fail completely in the real world, unless the simulation is extremely well matched to the real conditions. The controller must also cope with the motion of the ball - the tennis ball used in the project is far from perfectly round, and the collisions far from elastic.

The task is made more difficult by the robot and vision system used. The Khepera [36] is well suited as a standard research robot platform, but suffers from serious problems when used to perform complex behaviours. The robot's extremely small size renders it susceptible to problems caused by uneven surfaces, and picking up dust and other debris. When pushing a ball, the robot would frequently get stuck on a ridge on the floor. The vision system also suffers from problems with size - only a one-dimensional view of the environment is possible, and the range and sensitivity with which input intensity can be distinguished is low. Unless the ball was extremely well lit, the vision system would not pick it up at all, even against a black background.

One yardstick by which this project must be measured is whether the robot controllers perform the same behaviours in the real world and in simulation. It is this transfer of behaviour, or crossing the *reality gap*, that distinguishes minimal simulation techniques from Artificial Life work. Ideally, one would like the controllers to consistently score goals from any point on the pitch, but in practice a robot capable of finding the ball, heading towards it and pushing the ball forwards is fairly impressive. Chapter 9 shows the same controllers operating in both simulation and the real world, consistently finding the ball from any point in the arena, and pushing it forwards. Judged on the criterion of crossing the reality gap, and playing a simple game of football, the project is a success.

Chapter 2 outlines arguments for why evolution should be used in the design of robot control structures, while chapter 3 describes a new methodology aimed at ensuring control structures evolved in simulation operate reliably in the real world. Chapter 4 introduces the actual robot and vision system used, and investigates the vision system performance. Chapter 5 outlines how the methodology from chapter 3 is applied to the development of a robot vision system simulation. Chapter 6 describes two areas of investigation - navigation and robot football - in which a sighted robot with evolved control structure could be used, and details a variety of algorithms useful for visual processing. Chapter 7 goes through the evolutionary scenario and neural network controllers used in the project, while chapter 8 describes the simulated evolution of controllers able to perform a simple task, finding a stripe, in the real world. Finally, chapter 9 describes the setup and results for the evolution of a robot controller capable of playing football.

Appendix **??** contains the high level description of the source code, while appendices **??** and **??** contain the code itself, and user guide. Appendix **??** has experimental results from the characterisation of the robot vision system. Appendices **??** and **??** show full results for the stripe tracking and football playing experiments. Appendix **??** covers the actual parameter settings used in the experiments (for the overall code implementation of the experiments see appendix **??**). Appendix **??** details investigation of one successful neural network controller, while appendix **??** gives details on the fourth European Conference on Artificial Life Autonomous Robot football tournament, and finally appendix **??** gives an introduction to genetic algorithms and neural networks.

This dissertation assumes a basic knowledge in the fields of evolutionary computing and neural networks. Appendix **??** gives a brief introduction to, and overview of the terminology used in, these two areas.

Chapter 2

Robot Controllers

This chapter outlines the arguments for use of evolution over more traditional methods in the 'design' of autonomous robot controllers. The use of evolving neural network controllers is also examined (see appendix **??** for an introduction to the evolution of neural networks).

2.1 Problems with Hand Design

The arguments are well rehearsed for the use of artificial evolution over hand design, in particular the traditional artificial intelligence representation/planning paradigm, in the development of robot controllers. The traditional artificial intelligence approach is argued to separate the robot control system artificially from the environment, through the intermediate representation stage; witness 'Shakey's' [48] extremely slow sensor-motor cycle. In advocating the hand design subsumption[1] approach [3, 4], Brooks attacked the traditional 'perceive-model-plan-act' paradigm on the grounds that symbolic representation[2] is not needed for, and indeed may slow down, most of intelligent activity [6]. Planning is also seen as superfluous; agents react to external stimuli (the agent has 'internal states' which may modify/govern reactions), and the speed of the tightly coupled sensor-motor process enables the agent to operate in a dynamic environment.

However, [8] identified the critical problem facing subsumption and other approaches to the design of robotics controllers as *coherence*, or how a complex organism can mediate between the many possible competing behaviours. The traditional approach to design of a such complex system relies

[1]A control system approach based on behaviours being partitioned into a set of task-orientated activities operating in parallel.

[2][38] and more explicitly [10] make the point that subsumption uses representations of some sort, e.g. signals are passed representing that an edge has been detected in the visual field. However, these are not symbolic representations in the generally accepted definition of the term, and cannot be manipulated or reasoned with.

on decomposition into separate modules, typically responsible for separate functions[3]:

> Modular programming methods in classical Artificial Intelligence ... lent themselves quite nicely to a homuncular form of explanation.
>
> [9]

It is by no means clear that the designers of an autonomous system can assume such decomposition is valid, making hand-design of a complex robot controller extremely difficult indeed. Evolution is argued to be the only approach capable of coherently organising all interactions with the environment and between separate parts of the robot [25]. Evolutionary techniques must be implemented at some level, at the least to modify pre-designed modules to operate together, and possibly to evolve the entire system.

2.2 Evolutionary Robotics

The rise of biologically-inspired approaches to robotics is based on the premise that hand-design cannot provide coherent systems complex enough for autonomous operation; "Interesting robots are too difficult to design" [25].

[56] outlines the basic approach:

> the evolutionary methodology is to set up a way of encoding robot control systems as genotypes, and then, starting with a randomly generated population of controllers, and some evaluation task, to implement a selection cycle such that more successful controllers have a proportionally higher opportunity to contribute genetic material to subsequent generations, that is, to be 'parents'.

Chapter 7 describes the evolutionary scenario, e.g. how parents to breed are selected and the application of genetic operators such as crossover and mutation, and also outlines the control system and method of encoding. The design of the fitness evaluation and the architecture of the evolution process are crucial for success, and much work has been done in this area [45, 22]; chapter 9 describes the evaluation algorithms used for the football experiment in detail.

[3]Although subsumption takes a *behavioural* decompositional approach.

2.2.1 Neural Networks as Controllers

Within the evolutionary robotics field, several approaches can be distinguished. Higher level methodologies tend to operate on fixed architecture systems governing the output response for a given input. Koza's genetic programming language [40] is set up in such a way as to allow evolution of the control algorithm, while classifier systems [28] enable a set of rules to be evaluated and evolved over time.

Other approaches allow investigation of more basic aspects; the recent focus on neural network controllers [27, 49, 1] reflects the argument that evolution must be allowed to operate on a more primitive level than merely altering the control program or rules [25].

This project uses neural network controllers for a robot football player; evolution is used to produce networks that are *evaluated in simulation.* Qualitative evaluation of the final network controller evolved is then carried out in the real world.

Examples of evolved neural network robot controllers include walking behaviours for many-legged robots [33, 22], room-centring given visual input [11], simple shape discrimination [32, 26], grasping behaviour [50], and 'predators' chasing 'prey' [19].

Chapter 3 goes on to outline the problems with evolution in the real world and evolution in simulation, and describes new work aimed at solving the simulated evolution problems.

Chapter 3

Real and Simulated Evolution

In the following chapter, the differing problems with evolution in both simulation and the real world are outlined. The *minimal simulation* approach [31, 32] is then introduced as a potential solution to the problems facing evolution in simulation.

3.1 To Be or Not To Be: Real World versus Simulation

The issue of evolution in the real world versus evolution in simulation has traditionally been the trade-off between realism and time.

3.1.1 Problems with Simulation

It is argued that simulation cannot realistically model the features required for robust operation in the real world; robots evolved in simulation may completely or partially fail in the real world - the so-called "reality gap" [34]. Attempts to increase the simulation complexity merely result in expenditure of vast amounts of modelling and computing time, e.g. Webb's discussion on implementation details of modelling cricket phonotaxis:

> [it] would require a great deal of effort to build a computer model that reflected the real situation well enough to make strong claims that the mechanism actually works. And it was certainly the case here that a simple simulation was quite misleading about how the mechanism would perform.
>
> [55]

The issue is complicated further when the time-dependent dynamics of real systems are considered. [54] utilise an evolutionary approach to produce hardware robotics controllers, evolving uniquely efficient designs. The final solution is typically analytically intractable, based on the specific dynamics of the particular hardware employed. Simulation of the dynamics was impractical, and only evolving the actual hardware was found to be of use. [24] makes the point that:

> a computation of an output from an input is the same computation whether it takes a second or a minute, [but] the dynamics of a creature or robot has to be matched in timescale to that of its environment.

In other words, the temporal dynamics of an agent in simulation must be matched to its temporal dynamics in the real world for the agent controller to cross the reality gap.

A second problem with simulation is that of noise. In the real world, sensors and motors are not ideal; simulations must model the imperfect and non-deterministic elements of the environment.

3.1.2 Problems with the Real World

Evolution in the real world suffers from a different problem; that of the inordinate time required. With a large population evaluated over number of generations, the real time evolution cost can be prohibitively large. [17] required 65 hours to evolve efficient collision-avoidance controllers, and ten *days* to evolve learning controllers to perform the same task [18]. Bearing in mind that more complex tasks will clearly require longer evaluation times, and typically more than one evaluation per individual to eliminate random effects [44], the real world evolutionary time problem is clearly of critical importance. [5] argues, "the world is its own best simulation". But is that simulation fast enough?

A further practical problem with evolution in the real world is that of change/breakdown in the robots and the environment. Batteries will run out, wires will become tangled up, lightbulbs will blow, wheels will get snarled up with dust and other debris, etc. One doesn't want to return to an evolution experiment after a week to find that on the second evaluation the robot fell off the table.

Approaches to the problems of real world realism versus simulation time have involved initial simulation, followed by a period of on-line real world development [42]. [7] has suggested a continuous cycle of simulated and real world development, but section 3.2 looks at a radically different approach.

3.2 Minimal Simulation

[31, 32] outlines a new approach to the problems of evolution in simulation (section 4.2). He contrasts the *base set aspects* of the situation, those that may have some bearing on the robot behaviour, with the *implementation aspects*, those which must not be allowed to affect behaviour.

Base set aspects are those simulation features present in the real world upon which the robot behaviour might be based. For robustness, these are modelled noisily. Implementation aspects are those simulation features either not present in the real world (perhaps arbitrary regularities), not thought relevant, or not easily modelled. Instead of arbitrarily setting these aspects, or allowing some random distribution, these must be made *unreliable*, e.g. randomly set for each trial to one of 'on', 'off', 'big', 'little', 'randomly distributed', etc. The only practical evolutionary strategy is to ignore them completely.

The key point is that the base set need not be comprehensive, or particularly accurately modelled; minimal simulation base sets will only evolve controllers dependent on a small number of features, possibly not able to exploit the full real world situation, but able to cross the reality gap. However, the implementation aspects must be modelled unreliably.

[32] goes on to describe two successful experiments where minimal base set aspects were used to evolve-in-simulation networks for discrimination of shapes, and corner-turning following the direction of a previously seen light. Both networks were very successfully seen to perform the same behaviours in the real world situations. The comparison between simulation evolution time and theoretical real world evolution time is astonishing, the simulation being several orders of magnitude faster.

This powerful approach formalises the experimenter's previously thorny problems of *what* to simulate, and *how* to simulate it, and as such goes some way to solving the problems outlined in section 3.1. The addition of such hand-constraints - 'here is what the controller is allowed to use as input, and here is what it must not use' - will lead to many more evolutionary solutions crossing the reality gap. It is now possible to make simulations more general than the real world; agents can be evolved to ignore certain features, so can be used in differing environments. Imagine the real world evolved robot equipped with fancy vision systems whose evolution environment was a white laboratory; once sold to an interior decorator with a taste for pink, our robot refuses to work at all. This would not be true of the minimal simulation evolved alternative with colour specified as an unreliable implementation aspect; over each evolutionary trial, the simulated world would change colour, shade and intensity. Controllers based on colour would not work reliably on every trial so would achieve low fitness.

However, two notes of caution should be sounded. First, the researcher must identify the base set aspects. Second, as Jakobi points out, identification of the implementation aspects is by no means trivial; many will arise indirectly as features of the simulation program code, others will creep by unnoticed in the simulation set-up[1]. In essence, these are the representation and frame problems of traditional AI rearing their ugly heads [15]; what is relevant and irrelevant in *this* environment to *this* behaviour? However, here we have the advantages that under-specification is allowable - minimal base set aspects still evolve robust controllers - and that the two aspects can be specified 'off-line', i.e. before the controller is active.

These two points aside, the radical envelope of noise hypothesis is a big step towards solving the problem of simulation realism. Chapter 4 goes on to describe the actual robot used in this project, and outline how a minimal simulation of the robot and vision system can be set up.

[1]Reminiscent of the abilities of the US Military network trained in simulation to recognise tanks. Poor performance in the real world was explained once the simulation designers realised every scene containing a tank also contained a cloud, and that in fact the network was recognising clouds in the real world extremely accurately. As a minimal simulation implementation aspect, clouds might be present or not, in large or small quantities, with varying shapes and colours. The network simply could not reliably base output behaviour on clouds.

Chapter 4

Robots and Vision

This chapter introduces the hardware used in the project - the Khepera robot and the K213 vision system - before describing experiments investigating the vision system in a number of different scenarios (appendix **??** gives full details of the experiments performed). This characterisation of the relevant features of the vision system is used to identify key features necessary for the simulation described in chapter 5.

4.1 The Khepera Robot

Throughout this project, the standard Khepera research robot [47] has been used, see figure 4.1. The basic Khepera is a small (55mm diam. x 30mm high) two-wheeled robot equipped with eight infra-red proximity and ambient light sensors, with an on-board Motorola 68331 processor. Plug-in modules can be added; this project uses the K213 vision turret described in detail in section 4.2. Power is provided either externally through a serial cable[1], or through on-board batteries allowing around 30 minutes of autonomous operation.

The Khepera is particularly well suited as a standard research platform:

- Standard research robot, with existing research on simulation of performance characteristics.
- Plug-in modules, including the K213 vision module.
- Simple BIOS, and memory space for downloadable sensor-to-motor control programs.
- Small size, allowing experiments in small arena.

[1]provides both power (from the mains) and downloading of compiled files (attached to the computer serial port).

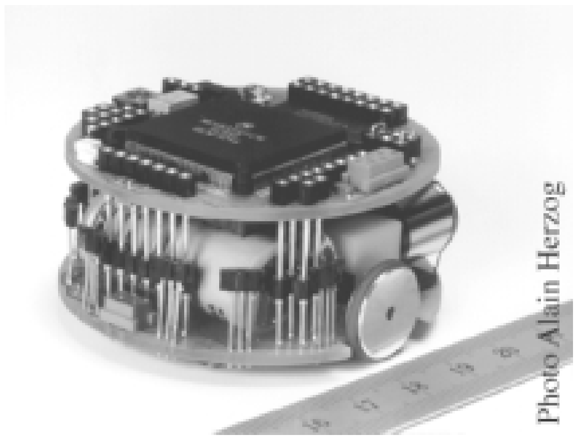


Figure 4.1: The Khepera Robot (photo Alain Herzog)

However, the Khepera suffers from performance limitations due to its small size. Dust from the environment seriously affects the wheel speeds, e.g. constant forward motor output can result in spinning on the spot if one wheel picks up a hair. If the robot is not on an extremely smooth surface, it can easily get stuck - especially when pushing something such as a tennis ball (the robot and ball would often get stuck on ridges in the floor). The vision system performance is also limited by size - for serious vision work, a one-dimensional input is not sufficient. The input sensitivity is extremely crudely controlled by the 'iris' (section 4.2.1) - either a more sophisticated algorithm, or a wider range of input intensity, would be an advantage.

4.1.1 Controlling the Khepera

The on-board Khepera processor supports a low-level BIOS [37] which in turn calls the basic microcontroller functions to control the robot. Multi-tasking[2] is supported, allowing several tasks to be carried out simultaneously. For instance, one task might be to explore an arena. This would be interrupted the moment another task, looking for interesting visual input such as edges, alerted the controller to the fact that an object might be nearby. A third task would then take over to go towards the object[3].

Software control is possible by downloading compiled programs onto the robot. The Khepera compiler takes standard ANSI C code using the BIOS functions, and produces a compiled file suitable for straight transfer to the

[2]In practice, most multi-task operations can be carried out by a single task, but splitting the different tasks up allows for more accurate timing (possibly where one task is waiting for the results from one or more other tasks).

[3]Similar to the subsumption approach [3].

robot. A typical file control loop would call for various sensor readings, setting the motor speeds on the basis of those readings.

4.1.2 Sensors and Motors

Figure 4.2 shows the distribution of the sensors and wheels on the robot. Each of Khepera's eight sensors has two detection modes, both active at any one time. Active infra-red emission and reception is used for object proximity detection, while the same receiver also measures incident ambient light.

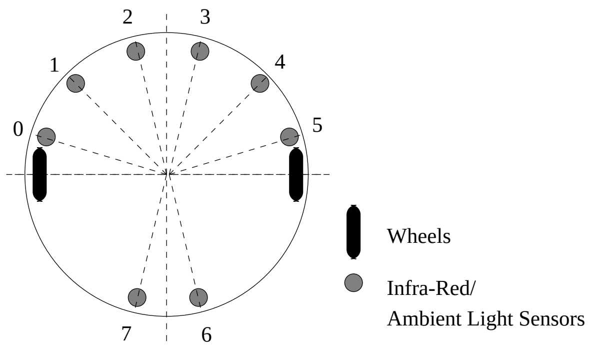


Figure 4.2: Khepera's eight infra-red/ambient light sensors and two wheels

Simulating the sensor characteristics and readings is only possible given experimental performance data; previous work [36, 30] has investigated the response of the sensors in active infra-red mode as a function of the distance from an object, the angle of the object to the sensor's line-of-sight, and the angle with which the infra-red hits the object's surface. [30] also investigates the ambient light sensor response to different lighting levels over varying distances, although this project makes no use of ambient lighting detection.

The two wheel speeds are set by the control program; feedback is used to ensure that the actual wheel speed is not significantly different from the set speed (see [30] for details on the feedback algorithm, a classical PID regulator).

4.2 K213 Vision Turret

The K213 vision turret is a module which simply plugs in to the basic Khepera package, see figure 4.3. The turret scans a one-dimensional line, returning grey-scale values (ranging from zero, or dark, to 255, or bright) for each of the 64 pixels on the line, i.e. the incident ambient light falling on each of the pixels over a set integration time. An 'iris' measures the total ambient light level, setting the integration time to avoid both under- and over-exposure of the pixel grey-scales. Figure 4.4 shows a schematic of the vision turret.

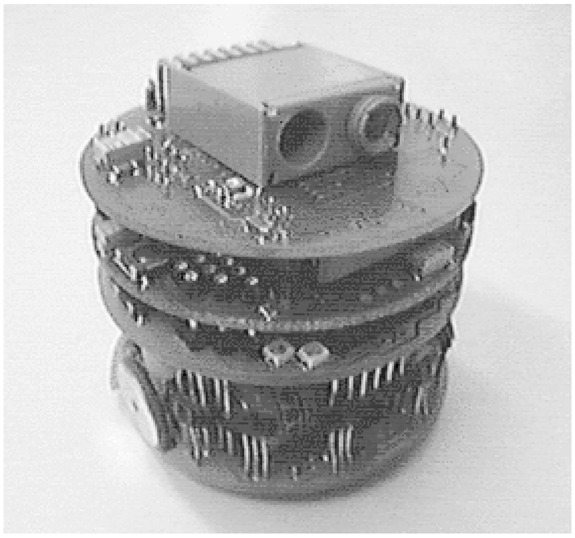


Figure 4.3: Khepera with the K213 vision turret

The vision turret is continually updating the visual pixel array (the time between scans is adjustable). Calling the turret for visual data will return the current array, although some basic pre-processing is possible, e.g. the position of the brightest and darkest pixels can be returned.

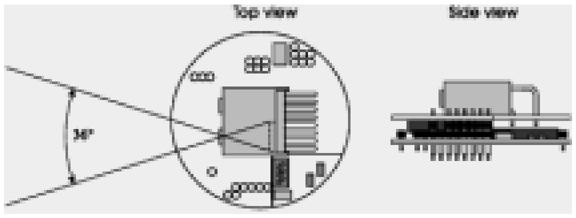


Figure 4.4: K213 Vision Turret. The top 'eye', or iris, measures overall incident ambient light, while the bottom 'eye' scans a one-dimensional line of 64 pixels, returning the incident light for each (the time over which the incident light is integrated is set by the iris).

4.2.1 The Iris

The K213 iris performs a similar role to that in human vision, scaling the incoming light intensity. In humans, the iris opens or closes, allowing more or less light in over some time period. By contrast, the robot iris changes the time period over which light is allowed into the 'eye'.

While reducing the possibilities of absolute saturation or of not receiving any light input, the iris has the effect of removing any absolute interpretation from the size of the visual input - scanning a uniform white background may produce exactly the same input (or higher, or lower) than when scanning a uniform black background. The overall ambient light falling on the iris controls the integration time, making the visual input data difficult to predict for a given background. Section 4.3 explores the effect on the input of varying light levels and distance from various background scenes, including uniform and striped backgrounds, and objects such as tennis balls.

The vision turret can return the current value for the ambient light sensor, enabling data to be scaled to absolute values. However, the iris cannot be turned off, and the integration time cannot be set from the controlling software - the iris effect on the integration time is hardwired [46]. Exploration of the effect of dividing input by the integration time produced somewhat unpredictable results; for stability in the input range, data was used with iris scaling effects included.

4.3 Visual Data

Appendix **??** gives results from experiments on the visual input data given a variety of backgrounds, with differing light intensities (a desktop lamp either illuminating the background, or turned off), and distance from the backgrounds. Each background was viewed for 100 scans of the vision turret, and the average value with errors (the average value plus and minus one standard deviation) plotted for the pixel inputs. Note the vision turret was called to return only one in four data values, thus only 16 values are shown on the graphs.

The results show four main points, crucial to modelling the visual system:

1. The pixel input is noisy, with differences over time. In most cases, the standard deviation is small, but non-zero. Only where the input is saturated does the deviation reach zero, see figure 4.5.

2. The input is noisy in a spatial dimension as well as over time - viewing a uniform background does not produce a straight line, see figure 4.6. These differences will be due to uneven lighting conditions, unevenness in the background itself (uniform black will never be exactly uniform black all over) and other effects extremely difficult to model.

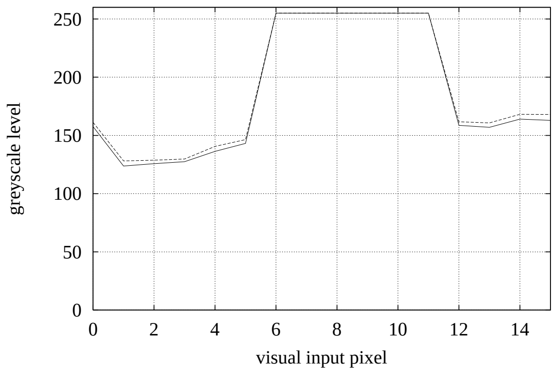


Figure 4.5: White stripe viewed from 20cm, light on

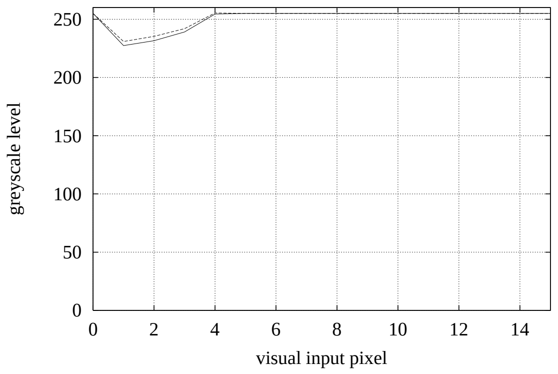


Figure 4.6: Black background viewed from 20cm, light on

3. The vision system inputs are not perfect - figure 4.6 shows a uniform black background, where three pixels on the left are returning lower values than the other pixels. This is *not* an artefact of the background not being perfectly uniform, either through uneven lighting conditions or uneven background colour. Pointing the robot in different directions at different backgrounds with different lighting produces the same under-valuing of the light intensity by those same three pixels.

4. Given a uniform background, in general it will not be possible to decide whether the background is light or dark from the visual data alone. Figures 4.6 and 4.7 show uniform white and black backgrounds viewed from 20cm with the light turned on. The black background saturates the pixels (clearly the iris has overcompensated), while the white background has reduced the integration time sufficiently to fall below saturation. Only a large white stripe on a black background

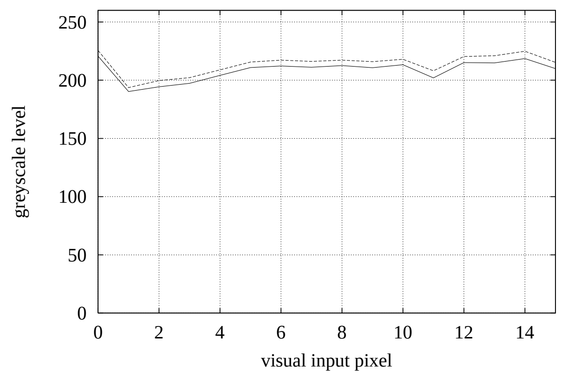


Figure 4.7: White background viewed from 20cm, light on

(figure 4.5) shows clearly that white is brighter than black.

It is clear that to model the visual input data for the Khepera adequately, the effect of both the environment and the vision system itself must be taken into account. Chapter 5 outlines how Jakobi's minimal simulation techniques [31, 32] can be applied to a simulated Khepera vision system.

Chapter 5

Modelling a Sighted Khepera

This chapter describes the simulation of a Khepera robot with vision capability. The basic hypothesis behind Jakobi's minimal simulation [31, 32] is that the controller must not be allowed to rely on features not present in the real world. The simulation need not be particularly accurate (indeed noise will play a major part) in the details of the real world, or base set, aspects, but must be positively unreliable in the details of non-real world, or implementation, aspects. By specifying an unreliable model, we avoid the extremely expensive computational overhead in producing an accurate simulation (with full ray-tracing, exact mapping of objects, and other difficult problems) which may only apply to one specific situation. See appendix **??** for the actual simulation details, e.g. the type and distribution of noise used in the various routines.

5.1 Robot Simulation

The simulation of the Khepera robot motors and on-board sensors is adapted from [31] and [52].

5.1.1 Motors

The new orientation and position are calculated on the basis of the left and right motor speeds (see chapter 7 for details of the neural network output), and the number of updates per second in simulation[1]. The simulation is crude in that a lookup table is used to avoid computationally expensive angle calculations when relating the robot orientation to direction of motion. Finally, noise is added to the motor speeds, position, and orientation.

[1]This was discussed in section 3.1.1 - to cross the gap between simulation and reality, the update speed must be similar to that in real life, i.e. how long the neural network will take to produce motor output given sensor input. Strategies evolved on the basis of moving a certain distance between updates may fail completely if they do not move the expected distance in reality.

5.1.2 Sensors

The ambient light sensor mode was not utilised at all; only the infra-red proximity detection capability was modelled. Simulation of the sensors is adapted from Jakobi's work on corridor-following minimal simulation [32], where the readings are roughly based upon the robot being within range of an infinite wall.

The arena is modelled as four infinite walls [52]. Given the orientation of the robot to the nearest wall, a lookup table gives the distances at which each sensor will detect the wall (which is scaled by the robot's distance from the wall). If the robot is near a corner, i.e. within range of two walls, the table will update each sensor distance with the nearest value. For objects in the arena, the simulation uses the distance and angle subtended by the object (already found for the vision input) to return a rough distance from each sensor to the object. Finally, the sensor distances are converted to input readings via a linear transformation, see equation 5.1.

$$input = \begin{cases} ir_{high} & distance \leq minrange \\ ir_{low} & distance \geq maxrange \\ ir_{high}(1 - \frac{distance - minrange}{range}) & \text{otherwise} \end{cases} \tag{5.1}$$

5.1.3 Interaction with the Walls

The robot interaction with a wall is extremely hard to model accurately - a large number of factors will govern whether the robot bounces off, stops still, skews round to face the wall head on, or some other action. The simulation needs to be set up in such a way as to discourage the robot to come close to the walls (robots doing this in the real world are unlikely to do well). However, this discouragement should not be in the form of some fitness evaluation term; early controllers are unlikely to be able to avoid walls, but they should not be penalised for the fact that they are able to move while other less 'good' controllers simply stand still. The solution is to make the robot-wall interaction unreliable - controllers are allowed to hit the wall without being penalised, but they will not be able to rely on the effects of such interaction. The unreliability implemented provided four possible consequences of hitting the wall:

- The robot does nothing, stopping still.
- The robot is bounced back a random distance (0-2cm), and spun in a uniform random orientation (-0.15 to 0.15 radians).
- The robot is moved in a uniform random direction (-3 to 3cm), and spun in a uniform random orientation (-0.6 to 0.6 radians).

- The robot is moved in a Gaussian distributed random direction (with deviation 1.5cm), and spun in a Gaussian distributed random orientation (with deviation 0.3 radians).

Later controllers will evolve to avoid the wall - good controllers cannot hit the wall and manage to track the ball or push it towards the goal consistently.

5.2 Aspects of Vision

The simulation of the vision system relies heavily on the four points identified in section 4.3 during the characterisation of the system (experimental details in appendix **??**). The key to the vision simulation is the addition of noise at a variety of scales. Identification of the implementation aspects enables the simulation to force them to be unreliable bases for robust behaviour.

Four features in the vision input data (incorporating features in both the real world visual data and Khepera vision system) were identified as implementation aspects, which must be varied from trial-to-trial:

- In the real world visual data, two implementation aspects were identified:
  - The background lighting must be varied across trials.
  - The absolute values for object colours must also be varied, so controllers must evolve to cope with a variety of slightly different coloured objects in the environment.
- Two features were also identified in the Khepera vision system:
  - The performance of individual pixels given identical inputs is not reliably the same - this should be varied across trials.
  - The input angle of each pixel is not easily modelled; it might be assumed that light incident over the whole angle is used. However, it may be that the pixel inputs 'bleed' into neighbouring inputs (i.e. the input angles overlap), or alternatively that the input angles are fairly small. To minimise simulation time, inputs are not integrated over some area, but taken from a single ray along the centre of the pixel input angle. However, this angle is varied randomly over each evaluation to forbid controllers using strategies based on fine differences in angle of view between the pixels.

5.3 Adding Noise to Vision

For the full vision simulation, noise was added at two separate levels (appendix **??** gives details). Evaluation noise was set for the four implementation aspects, while noise was also added over each iteration of the simulation:

- Evaluation noise was constant over each trial, and was used for the four implementation aspects identified above:
  - Background lighting was set to one of low, medium, high, or randomly selected from Gaussian or uniform distributions.
  - Object colours were varied in a Gaussian distribution around a mean identified from experimental data (shown in appendix **??**).
  - Pixel characteristics were set to one of all zero, or Gaussian or uniform distributed random numbers (different for each pixel).
  - The pixel input angle was varied as the pixel characteristics, choosing randomly between no change, or small uniform or Gaussian distributed variation.
- Iteration noise was varied between each simulation update:
  - If all inputs are the same, all are set to zero before adding noise and background levels (thus uniform backgrounds of any colour are indistinguishable).
  - Gaussian distributed noise is added to model the iris at each iteration (same for all pixels).
  - Finally, each pixel receives Gaussian distributed noise on each iteration.

We now have the framework for the robot and vision system simulation. The robot is modelled simply, and the interaction with the walls is made unreliable. A hierarchy of noise is added to the visual input data, ranging from the actual colours present in the simulation, the lighting levels on each evaluation, through to the individual pixel characteristics. Chapter 6 goes on to explore research on visual systems in nature, and how visual processing can be used to help the evolution of efficient solutions.

Chapter 6

Visual Systems

The following chapter describes two possible projects for a visually guided evolved robot. The possibility of exploring artificially evolved navigation strategies in comparison with real animal navigation is certainly one fruitful avenue of exploration. The evolution of controllers to perform a complex task - football - is a second more engineering-style project. The chapter goes on to explore a variety of visual processing algorithms, two of which was used in the project. The second half of the chapter is tied back to the first by research showing that animals do use visual processing of the sort described, when navigating through the world.

6.1 Visual Tasks

6.1.1 Navigation

One original aim of this project was to use the robot and vision system in the investigation of possible navigation strategies for an embodied agent in a simple environment, without the use of any representational models such as those used in previous work [43]. [35] describe a series of experiments exploring the use of stored pictures in honeybee navigation; so-called *retinotopic coordinate* visual patterns. The honeybees navigate by associating particular views of an area with particular motor strategies - perhaps triggered by some combination of features in the landscape. In a similar vein [2] has argued against the need for animals to have cognitive maps of any description, while [13] identifies at least four methods by which familiar landmarks can aid navigation without the use of some mental map.

[35] go on to identify a need for research on evolved creature navigation strategies to pin down the key mechanisms at work in visual navigation. [14] is currently exploring the behaviour of neural network controller animats finding a hidden food source with reference to a nearby landmark; one extension of this would be to see whether similar strategies were evolved

in simulations of real robots, and whether those strategies were successful when evaluated in the real world.

However, the focus of the project changed to the more practical evolutionary robotic footballing controller design, once I realised the 4th European Conference on Artificial Life (ECAL) was to have the first ever Autonomous Robotics Football Tournament based on Khepera robots using the K213 vision system.

6.1.2 Football

There is no doubt that robot football incorporates many of the hard problems facing both embodied and simulated agent research. A footballing robot control structure must be able to respond quickly to input, including extremely noisy visual data, selecting from a variety of different behaviours - finding the ball and goal; trying to score or save goals; tackling/blocking opponents - in real time. The robot footballing domain also covers such issues as multi-agent collaboration, machine learning, and the traditional AI favourite of planning. Indeed, [39] argue that robotic football should be considered as a standard problem for robotics and AI.

The simplified game played at ECAL, with only one robot per team, still presents considerable challenges. The evolution of a controller to identify and move towards the ball, finally pushing it in a certain direction, is certainly non-trivial. The controller must cope with the range of lighting conditions likely to be encountered in the arena, somehow distinguishing the ball even when the ball fills its entire field of view. The controller must also allow for irregularity in the ball's motion - tennis balls certainly do not roll in straight lines.

The football environment as defined for the 4th ECAL conference consists of an arena painted black, two indistinguishable goals painted grey, a yellow tennis ball, and two autonomous (i.e. not connected to a computer via a cable) robots viewed as black and white striped cylinders. The limits on the size of the robots meant that, in practice, only Khepera robots using the K213 vision turret were eligible. Each robot starts facing the opposing back wall, at a randomly chosen point (although, in practice these were agreed before the games). Seven halves of five minutes were played, each half ending early if a goal was scored or if neither robot touched the ball in two minutes. The tactics were clearly to find the ball and try to push it towards the opposing goal[1]; not an easy task given noisy visual input and the very basic Khepera vision system and on-board processing. Chapter 9 and appendix

[1]Although on the day, one suspects simply sitting in the robot's own goal would have been a fairly successful strategy!

?? describe the evolution in simulation and real world evaluation of the footballing robot, and appendix **??** gives results for the ECAL tournament.

6.2 Visual Processing

The real world is both complex and noisy, and visual input data invariably reflects this; the data returned from the Khepera vision turret is no exception (see figures 4.5 to 4.7). A variety of image processing techniques exist both to emphasise the main image features present and reduce noise.

Several factors must be taken into account when considering image processing:

Background noise Visual input will inevitably contain random noise in both a spatial and temporal dimension.

Background variation The visual scene will contain large scale variation - processing should take into account such problems as those raised by some areas falling in shade and others in sunlight.

Input range The range of input should utilise the full potential of the processing units - there is no point having a unit which saturates with only a tiny fraction of the possible input. Either the input should be preprocessed, or the unit operating range altered. Ideally, one wants all visual input to fall within the active range of the processing units, and to utilise the whole range, not just a small part.

DC bias The overall level of the visual input. In bright sunshine, every object will have high intensity, but the same objects viewed in the dark will have low intensity.

6.2.1 Digital Image Processing

The need for processing of the image data is balanced by the requirement that such processing should be fast for real-time on-board operation. This constraint effectively removes the possibility of using such techniques as Laplacian transformation of Gaussian convoluted data [51] and other computationally expensive methods. Several 'quick and dirty' methods are detailed below, to both remove noise and emphasise contrast.

Removing DC bias

The simplest type of filter removes the base level of input, setting all inputs on the basis of the smallest input, see equation 6.1:

$$g(x) = f(x) - f(x_{min}) \tag{6.1}$$

$f(x_{min})$ smallest input value

Thus the active range over which the processing units must operate is reduced. This was one form of processing used in the project.

Smoothing Filters

Linear smoothing filters are based on the premise that the value at each pixel is approximated as an unweighted mean of the pixel values in the surrounding neighbourhood. Note this may apply to both spatial smoothing, where noise is reduced across the visual field, or temporal smoothing, where fluctuations over time for a single pixel are reduced to a mean level, see equation 6.2:

$$g(x) = \frac{1}{M}\sum_{S} f(m) \tag{6.2}$$

x current pixel
$g(x)$ current pixel, processed input
$f(m)$ mth pixel, unprocessed level
S neigbourhood of current pixel
M no. of points in S

Sharpening Operators

Smoothing filters simply average over the pixel's neighbourhood. Sharpening operators relate this *expected* pixel value to the *actual* pixel value, see equation 6.3:

$$g(x) = f(x) - \frac{1}{M}\sum_{S} f(m) \tag{6.3}$$

Now we have some kind of simple edge detection emphasising the high frequency features of the input - if the value at x is radically different from what would be expected given the neighbourhood values, the operator will return high values. By contrast, if the value at x is the same as expected from the surrounding pixels, the operator will return zero. The added bonus is that some basic style DC bias removal has taken place (which is a side-effect of all operators which work on the differences between neighbouring pixels).

Contrast Operators

The next enhancement is to weight the sharpening operator by the inverse mean intensity to produce a contrast operator, where differences between object intensities are similar, whatever the background lighting levels (equation 6.4):

$$g(x) = \frac{f(x) - \frac{1}{M}\sum_S f(m)}{\frac{1}{M}\sum_S f(m)} \tag{6.4}$$

Weighted Contrast Operators and Predictive Coding

Predictive coding [21, 53] uses a similar scheme to the contrast operator, but weights the neighbouring pixel levels on how far from the current pixel they are. The key is over how large an area the mean intensity is calculated - too large and real differences in the background (part of the picture in shadow, part in the light) will be included, but too small and the mean will give big weighting to the actual intensity differences of the pixel level one is trying to ascertain. Full predictive coding uses statistical estimation theory based on the signal to noise ratio and lateral spatial correlation to alter the size of the sample over which the mean is calculated.

Here I use a simplified weighting system, where the mean is calculated over the neighbouring three pixels, each of which is given some weighting. The mean is *not* used to weight the contrast, thus low lighting levels are likely to have very low contrast levels; analogous to the robot not seeing in the dark. Processing is only performed in the spatial dimension; some form of motion detection processing would be of value. However, objects are likely to move fairly slowly in the football arena! Equation 6.5 gives the visual processing used in this project:

$$g(x) = f(x) - \sum_{i=-1}^{i=1} A(i)f(x+i) \tag{6.5}$$

x current pixel
$f(x)$ current pixel, unprocessed level
$g(x)$ current pixel, processed level
$A(i)$ weighting on ith neighbour

Further work could explore the possibility of using full predictive coding for the visual input, and the effects of using time-based predictive coding (in essence a basic motion detector).

6.2.2 Insect Visual Processing

[41] investigates the visual processing in the fly retina, finding evidence that both predictive coding and matched amplification (scaling the input up to match the range over which the neurons can process) are used. Thus there is evidence that the sort of algorithms discussed in section 6.2.1 are actually implemented in real animal retinal networks.

Chapter 7 looks at genetic algorithm techniques, the evolutionary scenario, and the neural network controllers used in the project.

Chapter 7

Genetic Algorithms and Neural Nets

This chapter describes the genetic algorithm and neural network utilised in the project. In particular, the parent selection and breeding schemes are described. Appendix **??** gives an introduction to genetic algorithms and neural networks.

7.1 The Genetic Algorithm Scheme

The ideas of Genetic Algorithm search techniques [28][1] come from evolutionary biology, in particular Darwinian natural selection. Appendix **??** gives an introduction to the area, and explains the terminology used.

See figure 7.1 for the high-level details of the genetic algorithm employed. This project uses a distributed genetic algorithm, where each solution is considered to occupy a unique position on a two-dimensional toroidal grid. Initially, the grid is seeded with random genomes (each bit randomly chosen from a uniform distribution ranging from -0.5 to 0.5), each of which maps onto a neural network controller for a simulated Khepera robot (see section 7.5 for details of the network, and how the genotype is mapped on to the phenotype). Each solution is then evaluated (see chapter 9) and the main program loop entered.

On each generation, the algorithm iterates $PopulationSize$ times, choosing a random location on the solution grid. A *mating pool*, consisting of the current location plus the neighbouring eight, is set up centred on the randomly chosen grid location, and the mating pool solutions ranked in order of fitness. Roulette selection (section 7.2) is used to find the parents,

[1]See [29, 20] for good overviews of GAs and their uses.

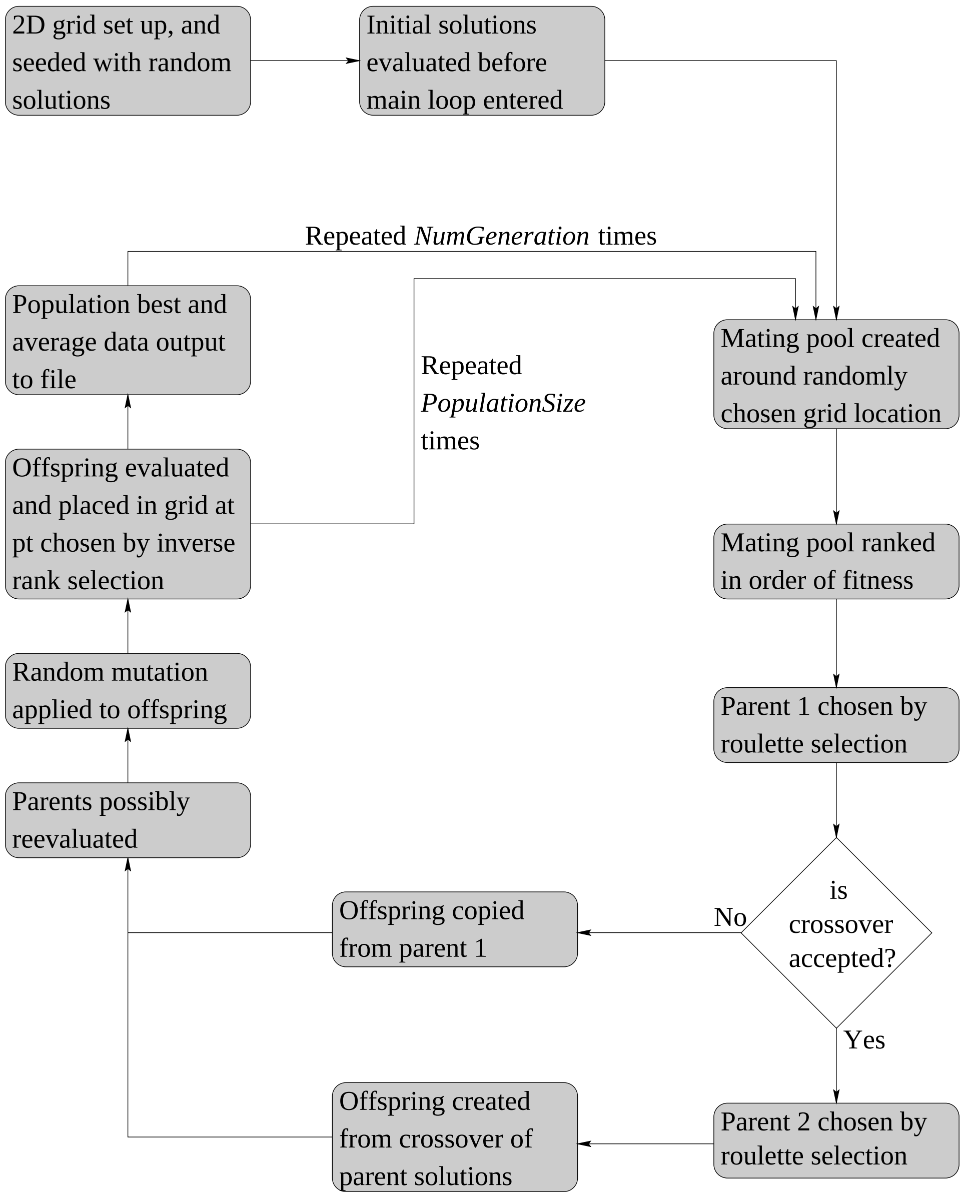

2D grid set up, and seeded with random solutions
Initial solutions evaluated before main loop entered
Repeated *NumGeneration* times
Population best and average data output to file
Repeated *PopulationSize* times
Mating pool created around randomly chosen grid location
Offspring evaluated and placed in grid at pt chosen by inverse rank selection
Mating pool ranked in order of fitness
Random mutation applied to offspring
Parent 1 chosen by roulette selection
Parents possibly reevaluated
Offspring copied from parent 1
No
is crossover accepted?
Yes
Offspring created from crossover of parent solutions
Parent 2 chosen by roulette selection

Figure 7.1: Flow chart of Genetic Algorithm setup

and breeding produces an offspring solution which is evaluated and placed in the solution grid (section 7.3). The parent solution(s) are reevaluated (if the reevaluation probability is accepted) to avoid poor solutions being randomly evaluated as highly fit, so having a high chance of being chosen as parents each generation, and also not being replaced. Note that updating is *asynchronous*, in that the population grid is not all updated at one time (a synchronous version of this algorithm might seed the children into a new pool, replacing the old pool with the new one once full), and also that no location is guaranteed to be in any mating pool on a particular generation (although this would be fairly unlikely).

Average and best population fitnesses are output each generation, and the best individual genotype printed to file (this can be used to seed a new run with mutated copies). No test is used to see if genetic convergence has occurred, i.e. has one or more best genotypes spread throughout the population reducing the population's ability to avoid local minima; the algorithm is simply iterated a set number of generations before finishing.

7.2 Rank Based Roulette Selection

The project employs a rank based roulette selection (see [29] for further details). Each solution has a possibility of being picked dependent on the ranking of that solution when compared with others in its mating pool. Thus one solution scoring much higher than other solutions in its mating pool will receive no higher probability of being picked than a solution in a different pool scoring marginally higher than its potential partners. With selection based on the actual genotype scores, such a solution scoring very highly compared to its neighbours would be likely to overrun the population, being picked every time for mating. However, rank based selection avoids this problem; the best solution in the neighbourhood always has the highest probability of being picked, but never at the expense of other solutions being chosen.

A second advantage of rank based selection is that for any reasonably complicated fitness evaluation, there is unlikely to be a linear relationship between the fitness difference of two solutions, and how much better one is than the other in completing the desired task. It is legitimate to argue that one solution is better than the other, but to go further and argue that it is a given amount better is not justifiable. The solutions should be ranked for the selection procedure.

Equation 7.1 gives the roulette selection algorithm; the probability of the ith ranked member being selected from a mating pool of size N (where ranking is 0 to $N-1$, with 0 being the fittest):

$$P(i) = \frac{N-1-i}{\sum_{j=0}^{j=N-1} j} = \frac{2(N-1-i)}{N(N-1)} \tag{7.1}$$

Note, the lowest ranked, i.e. $N-1th$, solution has zero probability of being selected for breeding. This has the effect of ensuring the highest ranked solution is not replaced with the child when roulette selection on inverse ranking is used to find where the child solution will be placed, a form of *elite* GA where the best solutions in each generation are always preserved.

7.3 Solution Breeding

The first parent is chosen through roulette selection. If the crossover probability is accepted, the second parent is also chosen (it is possible for the same solution to be chosen for both parents) through roulette selection, and the child solution created through one-point crossover, see figure 7.2.

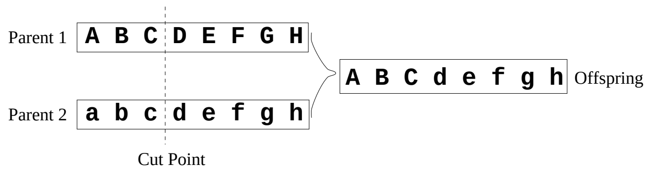


Figure 7.2: Offspring solution created from one point crossover of two parents

Finally, mutation is applied by seeing if the mutation probability is accepted, and randomly changing selected bits on the genotype by a uniformly distributed random number in the range -0.5 to 0.5 (if crossover is not accepted, the single parent is randomly mutated to produce the child solution). The child is evaluated, and placed in the grid at a position chosen by roulette selection on an inverse ranking of the mating pool.

7.4 Fitness Evaluation

The actual evaluation of a particular phenotype is the most important part of the evolutionary process. It is the evaluation that provides the pressure for 'good' solutions to emerge, indeed the evaluation defines the meaning of 'good'. A full theoretical discussion of fitness evaluation design is beyond the scope of this project, but [45] gives a good overview of the sort of problems faced when trying to scaffold a complex behaviour up by rewarding *a priori* identified simpler behaviours which may be needed for the full task at hand.

In an environment with any element of chance, i.e. random starting conditions or noisy sensors, a series of fitness evaluations will be distributed around the 'true' fitness value. Simply taking one evaluation is likely to result in wrongly identifying good and bad solutions. One method is to take a series of evaluation trials, returning perhaps the average score (as an approximation to the true fitness), or the lowest score (ensuring the controller is robust to all possible scenarios). See chapter 9 for details of the fitness evaluation functions used in the football experiment, and investigation into the effect of using average, median and lowest fitness evaluation scores.

7.5 The Project Neural Network

Appendix **??** gives an overview of artificial neural network research, and a description of the basic types of network. Genetic algorithms and neural networks used together are an extremely powerful combination. In principle a large enough multi-layer network can map any input on to any output; GAs are well-placed to find good networks for a particular behaviour, where the desired input-output mapping may not be known for any particular time step (making supervised learning algorithms such as back-propagation impractical). For instance, in a neural network controller for a robot, only the desired robot behaviour is known, not the input-output behaviour of the neural network itself.

For the football controller experiment (chapter 9), this project utilised a direct encoded fixed architecture feed-forward neural network with synaptic weights fixed over lifetime, shown in figure 7.3. The 19 inputs, consisting of 8 visual inputs, 8 infra-red proximity detector inputs, one compass (set to one if facing the opposing goal, zero otherwise) and two recurrent motor connections, are fully connected to a layer of 16 hidden units. The 16 hidden inputs receive an additional input from a bias unit, considered to always have an activity of 1.0 (so even if all sensory inputs are zero, the hidden layer will receive this bias input). Finally, each of the 16 hidden units is connected to the three outputs; two correspond to the left and right motors, while the third corresponds to a 'ball-present' unit.

The motor output neurons are used to set the left and right motor speeds (which are recurrently connected as inputs, so the network has some 'memory' of what happened on the previous time step), while the 'ball-present' node is used in the fitness function to encourage the controller to identify the ball and actively go towards it, rather than just being in the vicinity by chance.

Each of the links between units in figure 7.3 is associated with a genetically specified weighting; a pre-synaptic[2] output value is associated with some post-synaptic input. Equation 7.2 shows the full input for the ith hidden layer unit, from N input units (note the inclusion of the bias unit activity

[2]from biology, referring to the unit at the initiating end of the synapse. Post-synaptic refers to the unit at the receiving end of the synapse.

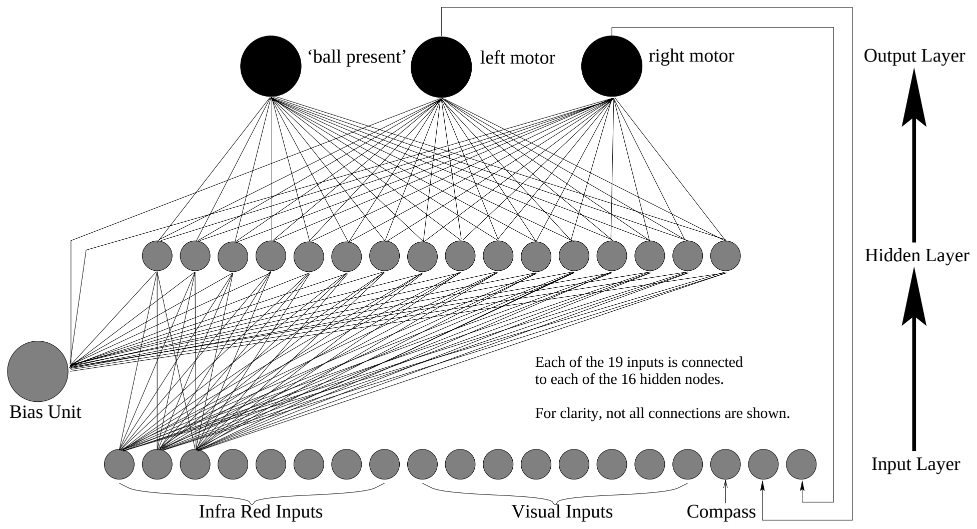


Figure 7.3: Feed-forward neural network used in project, with IR input

1.0):

$$input_i = a_{i,bias} + \sum_{j=0}^{j=N} (a_{i,j} A_j) \tag{7.2}$$

i hidden layer unit
j input layer unit
N no. of input units
$a_{i,j}$ synapse i to j weighting
A_j activity of jth unit

Each unit in the neural network contains an input, output, and synapse weightings. At each time step the input is initialised to zero, before adding in all input values (the pre-synaptic output weighted by the corresponding synaptic weight), some of which may be recurrent connections using the previous time-point outputs. Finally, the output is calculated on the basis of the transfer function, see section 7.7.

7.6 Genotype to Phenotype Mapping

Each solution genotype consists of an array of numbers, initially randomly distributed over the range -0.5 to 0.5, but not constrained within any range thereafter; in principle random mutation can produce much larger/smaller values, although in practice values were rarely seen outside the -1.0 to 1.0 range. Each type of unit was built up from the genotype:

Input Units The weightings from each input unit to each hidden unit were genetically specified; the first $NumInputs * NumHidden$ bits on the genome refer to these weightings.

Hidden Units Each hidden unit part of the genome holds a bias weighting value, by which the bias unit activity 1.0 is multiplied before adding as an input. The next three bits contain weightings to the three output units.

Output Units The two motor outputs hold a bias weighting value, and weighting values corresponding to the recurrent connections to each of the hidden layer units. The 'ball-present' unit holds only a threshold value (if the input is greater than this value, the unit output is set to one, otherwise to zero).

7.7 Unit Transfer Function

The input units are scaled to range from 0.0 to 1.0 (although processing may reduce this further for the visual inputs), and the mapping between input and output activity of these units is direct, i.e. an input of x will produce an output of x. Both the hidden layer units and the motor units have a choice of transfer functions; initially a sigmoid function (equation 7.3) was used. However, this was replaced with a much less computationally expensive linear function (equation 7.4) which produced a very similar input-output mapping, see figure 7.4.

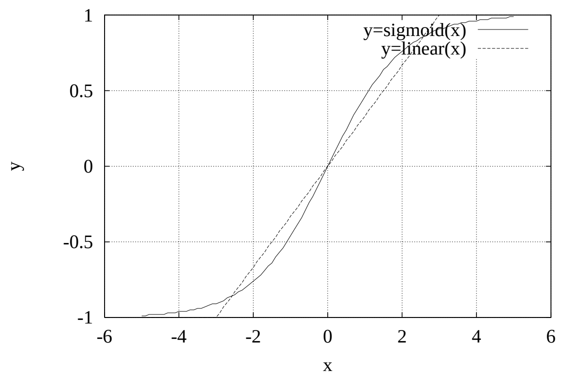


Figure 7.4: Sigmoid and linear transfer functions

$$output = \frac{2}{1 + e^{-input}} - 1 \tag{7.3}$$

$$output = \begin{cases} 1.0 & input \geq threshold \\ -1.0 & input \leq -threshold \\ input/threshold & \text{otherwise} \end{cases} \tag{7.4}$$

The addition of transfer function noise (in practice, a small random element is added to the input, before the output is calculated) is a subtle factor influencing the evolutionary process. At one level it forces controllers to be robust with respect to small changes in input; in some sense equivalent to using a different controller at each time step, with slightly different synaptic weights. Over an evaluation, a sample of controllers is used, centred on the actual controller being evaluated (similar to the idea of lifetime learning exploring the phenotype space around the evaluated phenotype). At another level, the noise can actually push the network into different behaviours; random change might be used in some sort of random *behaviour selection* where there is no unequivocal *best* behaviour.

An extension to the network used here might allow the value of the parameters in these equations, currently fixed, to also evolve for each unit. Thus the linear threshold for each unit could evolve to a suitable value to cope with the range and no. of inputs (especially useful when a variety of processing techniques might be applied to the visual inputs, but not to the infra-red input).

Chapter 8 goes on to explore efforts to evolve controllers able to find a white stripe on a dark background - testing the theory of minimal simulation evolution in practice.

Chapter 8

Experiment 1: Spot the Stripe

This chapter explores applying the minimal simulation ideas outlined in chapter 3 to a simple stripe finding task. This scenario is used to illustrate the effect of adding *unreliability* to certain aspects of the simulation. The experiment also highlights certain features of the controller and simulation required to ensure robust operation in the real world. Appendix **??** gives full details of the experiment, while appendix **??** contains the parameters used.

8.1 Predators and Prey

[19] explore a simulated coevolutionary scenario in which two Khepera robots compete. The *predator* robot is equipped with the K213 vision module (chapter 4), while the *prey* is able to move at twice the speed (both robots are equipped with the standard Khepera sensors). Both robots use fixed architecture feed-forward network controllers with evolved synapse weights, connecting sensor and vision inputs through a single hidden layer to the motor outputs. Lifetime learning is used to adjust synapse weights over the robot's evaluation. [19] go on to analyse the results using coevolutionary analysis techniques introduced by [12], and find a number of interesting tactics developed by both the predator and prey. However, here we are concerned primarily with the techniques used to model the vision turret performance, and how the visual input is used in the neural network.

8.1.1 The Predator's Visual Environment

[19] model the environment as a uniform white arena, with the prey modelled with a black 'turret' which enables the predator to see it from any point in the arena. The K213 vision turret's preprocessing feature is

utilised to return the position of the darkest spot in the predator's visual field - i.e. the position of the black prey on the white background. The predator's visual field is split into five sectors, each corresponding to an input to the neural network. If the prey is in one of the visual sectors, that neuron input is one, otherwise the input is zero.

Three crucial points affect the model, and its application to the real world [16]:

- The predator sees the *exact position* of the prey; the simulation has no noise in the visual input.
- The prey has *no dimension*. However close the prey is to the predator, the visual input is the exact centre of the prey.
- If the prey is outside the predator's visual field, *no visual input* is received by the predator whatsoever; all neuron visual inputs receive zero.

[16] outlines work on transferring the controllers evolved in simulation under the above conditions into reality. However, he describes the displayed behaviours in reality as "far from optimal". The next section outlines experiments aimed at enhancing the simulation to a point where the visual controllers evolved are able to cross the reality gap. The scenario used is an extremely simplified version of the [19] setup - the robot must simply find a well defined white stripe on a black background.

8.2 Closing the Reality Gap: Minimal Simulation Experiments

The simulation set up is extremely simple, based on minimal simulation as described in chapter 3. The environment is defined as a black arena with a single white stripe on the wall.

The robot is modelled with only four visual inputs (no Khepera on-board sensor inputs are used), each of which is one if the corresponding visual sector contains the brightest visible pixel, zero otherwise. The network is more complex than that used by [19]; the visual inputs are fully connected to the five hidden layer neurons, in turn fully connected to both the two motor units and recurrently to each other (the recurrent connections are activated on the next time step). Finally, the motor units are recurrently connected back to the hidden layer as two inputs. The motors are unrealistically modelled as changing only the robot orientation, not the x and y coordinates, and the fitness evaluation is simply the length of time that the robot is 'looking' at the stripe (bonus is given for having the stripe at the centre of the visual field).

All controllers evolved were downloaded and tested on a real robot. It was soon found that the conditions used in [19] were simply not producing controllers working in the real world. Several variations were explored before controllers able to cross the reality gap were evolved.

8.2.1 Increasing the Unreliability

Four different scenarios were investigated before robust (in the sense of crossing the reality gap) controllers were evolved to find the stripe:

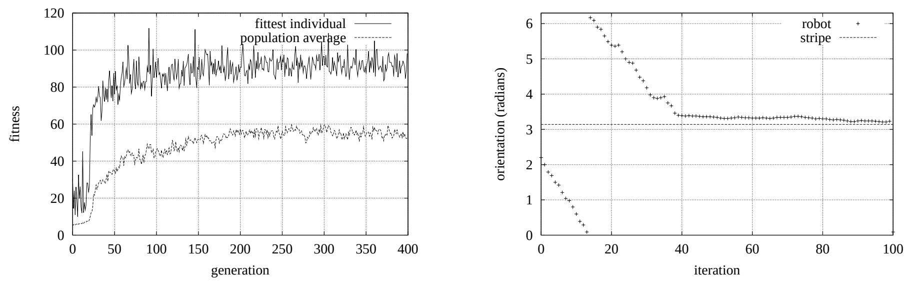
Experiment A The conditions were as in [19]. The stripe was given no dimension (the visual input was always the pixel corresponding to the centre of the stripe), and the visual input was zero if the stripe was outside the visual field. In simulation, controllers evolved quickly to accurately find the stripe. However, the downloaded version performed poorly; the robot did not move at all. This can be explained by the visual input never equalling zero; the controller's condition for rotating to find the stripe, i.e. when all input is zero, is never satisfied so the robot does not move.

Experiment B The same setup as in experiment A, but with the added condition that if the stripe is not within the visual field, the brightest input is randomly chosen. This introduces the reason for recurrency; in some sense the robot must 'recognise' the regular movement of the stripe across its visual field, possibly with reference to the motor outputs. In simulation, the controllers evolved are extremely fit, finding the stripe very fast, but in reality they are far from optimal. Although the robot circled 'looking' for the stripe, none found it and stopped.

Experiment C As experiment B, but now the stripe has a given width. When the stripe is in the visual field, the input is randomly chosen from those pixels covering the stripe, while if the stripe is outside the field of view, the input is randomly chosen from all pixels. Again, the simulated controllers have high fitness, but fail in reality; the robot sometimes found the stripe, but would then 'lose track' after moving towards or away the stripe (thus changing the perceived size of the stripe).

Experiment D As experiment C, but with the final condition that the stripe width changes on each fitness evaluation. In simulation, the controller performs as the two controllers above. However, *the evolved controllers work in the real world*, consistently finding the stripe of whatever width.

Figure 8.1 shows data for one run resulting in controllers working in the real world.



(a) Average population fitness (b) Best individual performance

Figure 8.1: Stripe finding with 4 brightest pixel inputs

To evolve a stripe-finding controller, it is seen that unreliability (in this case randomly varying the stripe width) has to be applied over different evaluation trials. Only controllers capable of finding different width stripes in simulation are seen to find stripes in the real world. This can be explained by the strategies used by the evolved controllers - given a simulation with only one stripe width, the strategy is likely to be based on some very specific detail, perhaps the robot stops spinning only when one certain visual input is high, and others are low. The real world stripe is unlikely to match this simulation stripe, especially when factors such as distance from, and angle of view to, the stripe are taken into account. A controller able to operate reliably in a simulation with varying stripe widths will operate in the real world - and as a bonus the simulation need not include such details as the varying distance and angle of view to the stripe.

Four interesting points can be made:

- To match the temporal controller dynamics in simulation to the real world, it was found useful to run the simulation to produce a trial controller, and timing the controller on the real robot before setting the simulation update speed and running the full evolutionary routine.
- In simulation, the x and y coordinates are not changed by the motor output, obviously not true of the real world version. The successful evolved controllers tended to move in small spirals (the fastest way if finding the stripe is to spin on the spot; x and y motion is not useful) before stopping when looking at the stripe. However, several controllers then *moved the robot towards or away from the stripe*, i.e. motor output is not necessarily zero once the stripe has been found,

but the left and right motor speeds are constant. The simulation unreliability in the stripe width enabled these controllers to cope with the stripe changing in size as the robot moved closer/further from the stripe (an effect not included in the simulation).

- Once good stripe-finding controllers had been evolved in simulation, the environment was varied. The same controllers, with no further evolution, were used in a scenario where the stripe was not stationary but moving round the arena. The controllers were able to track the moving stripe, even with occasional changes in the direction of motion (although this was not found for high stripe speed, or when a large random element of motion was included). This behaviour was also seen in the real world; moving the stripe slowly produced a rotation of the robot. Evolution has produced something for nothing; controllers evolved on the capability of finding an object are also capable of tracking that same object.

- The controllers found are still not ideal, often stopping at light patches on the arena walls (either due to uneven shading or lighting). As these are real areas of higher intensity, they are given the same status as the stripe to be found. No way of distinguishing between these and the stripe is possible unless more information is used from the scene; the full grey-scale capabilities of the vision system must be used to progress to more complex environments.

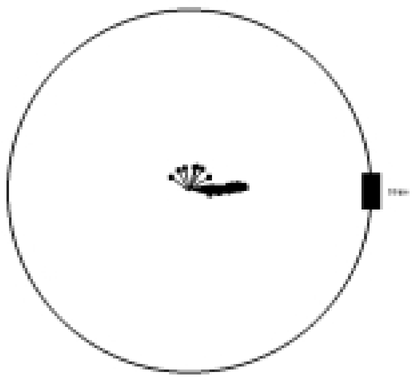


Figure 8.2: Stripe finding, controller in simulated evaluation

Figure 8.2 shows the simulated evaluation of the controller - once the clockwise turn has brought the stripe into view, the controller moves forwards slowly keeping the stripe dead centre. This behaviour was shown in the real world. Once the downloadable controller had been established, two more experiments were carried out. The visual inputs were increased from four to eight and then sixteen, so increasing the accuracy of the stripe positioning. Once the conditions described in the fourth experiment above were implemented, controllers able to work in the real world were evolved with different numbers of visual inputs.

8.3 Adding Noise and Implementing Grey-Scale Vision

Appendix **??** describes further experiments where the noise was increased in the simulation described above. Random data was added to the wheel speeds, robot orientation, and the neural network transfer function. It was found that the controllers coped with the extra noise, evolving to similar fitnesses as with lower noise. The input data was also changed to return grey-scale intensity data - again, controllers evolved successfully to find the stripe. Evaluation in the real world found them to also be successful at finding a real stripe *once the simulated stripe width was varied over trials.*

This chapter has outlined how real world controllers can be evolved in simulation for a simple problem, given enough unreliability in the simulation. The setup makes no pretense of evolving the most efficient network for the problem - it is unlikely that such a large network with so much recurrency is needed for such a simple task, and no investigation was done on the most efficient number of visual inputs. However, the principle of unreliable simulation has been established. Chapter 9 goes on to implement a minimal simulation of the full grey-scale vision system for a much harder scenario; namely robot football.

Chapter 9

Experiment 2: Robot Football

The following chapter describes the evolution of controllers capable of playing football. Three fitness evaluations were investigated, and improvements to the basic simulation made before controllers operating in the real world were evolved. Screen-shots from real world operation video data are shown, and the chapter concludes with some preliminary analysis of the network controller strategy. Appendix ?? gives full details and results of the experiments carried out, while appendix ?? contains the parameters used in the experiments. Appendix ?? shows analysis of the neural network controller. For a full explanation, a complex systems analysis is required; presented here are qualitative arguments.

9.1 Fitness Evaluation

The fitness evaluation function was investigated, and three different versions used; a version rewarding simply how near the ball was moved to the goal; an incremental evolution scenario initially evaluating controllers on 'looking at' the ball, seeding these into a scheme rewarding controllers for hitting the ball, and finally seeding these into the fitness function scenario of how close the ball was moved towards the goal; and also a full scaffolding evaluation rewarding all the behaviours above - seeing the ball, hitting the ball, moving the ball towards the goal, and scoring goals.

In practice (see appendix ?? for details), the simple "how near is the ball to the goal" function failed to produce controllers scoring above zero - the task is too hard for such an open-ended evaluation. The incremental evaluation was no better than the full scaffolding function (although faster in the final evaluation, it required the previous evolution scenarios as well), and more complicated to implement. The remainder of this section uses

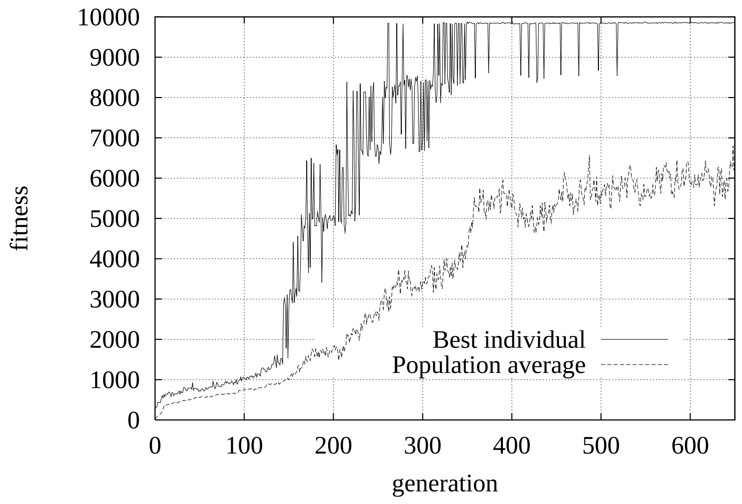


Figure 9.1: Scaffolded evaluation function, fitness over generations

the full scaffolding function described in equation 9.1 - the use of a fitness evaluation rewarding behaviours identified as necessary to scoring goals may restrict the possible strategies evolved, but guides the evolutionary process into good area of the search space. Figure 9.1 shows the population best and average fitnesses over the evolution. Fitness scores around 300 are controllers repeatedly hitting the ball, above that are controllers pushing the ball towards the opposing goal, while fitnesses around 10000 are controllers scoring goals (see appendix **??** for details of the actual fitness parameters).

$$\begin{aligned} fitness &= \frac{TimeLookingAtBall}{TotalTime} + f(ClosestDistToBall) + \\ & \sum_N S(HitBall) + g(BallToGoalDistance) \end{aligned} \tag{9.1}$$

$f(dist) \propto dist$ linear with distance with upper/lower thresholds
$g(dist) \propto (C - dist)^2$ rises as the square of distance ball moved
$N \leq 20$ no. of ball-robot collisions
S score for activity

9.1.1 Evaluating a Fitness Distribution

Given such a scenario with a large element of noise, it is vital to perform several evaluations of fitness for a single controller. There are several ways to use this distribution of fitnesses; appendix **??** investigates the effects of returning the lowest fitness (producing very robust controllers); the average fitness (highly skewed by such a fitness evaluation as that used here, where scoring goals produces a fitness of 10000, but dribbling the ball the wrong way only scores a fitness of 300); and the median fitness score. The median score is a better measure of such a skewed distribution - less weighting is given to extreme scores - and is used in the remainder of this section.

9.2 Enhancing the Simulation

The controllers evolved under the median fitness evaluation scheme and the full scaffolding function were downloaded onto the real Khepera robot and tested in an arena built to the ECAL football tournament specifications (see appendix **??**). However, the behaviours were not optimal.

Two problems emerged. First, the infra-red sensors were clearly not picking up the black walls (verified by testing the infra-red readings near the wall) - the black absorbs infra-red rather than reflecting. Second, the controller was circling to find the ball, then darting forwards once it was in view. However, the forwards motion always occurred slightly after the robot had circled past the ball, and careful matching of the simulation and real world network update speeds made no difference to the behaviour. Two changes were made to the simulation to coutneract these problems:

9.2.1 Removing the Infra-Red Inputs

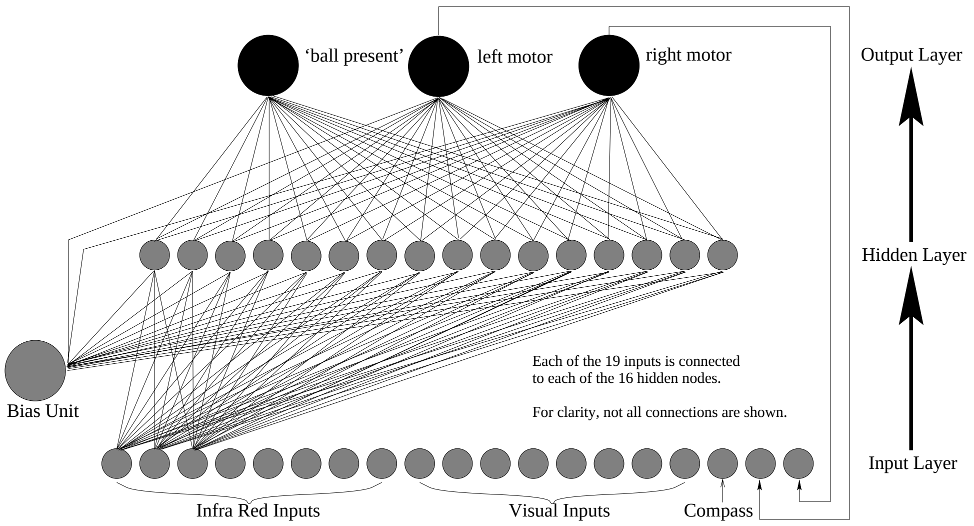


Figure 9.2: The old feed-forward neural network

The first enhancement to the simulation was to remove the infra-red inputs; figures 9.2 and 9.3 show the old and new versions of the neural network. The number of inputs has been kept constant by using eight extra visual inputs, doubling the vision sensitivity. To counter the problem of the controller

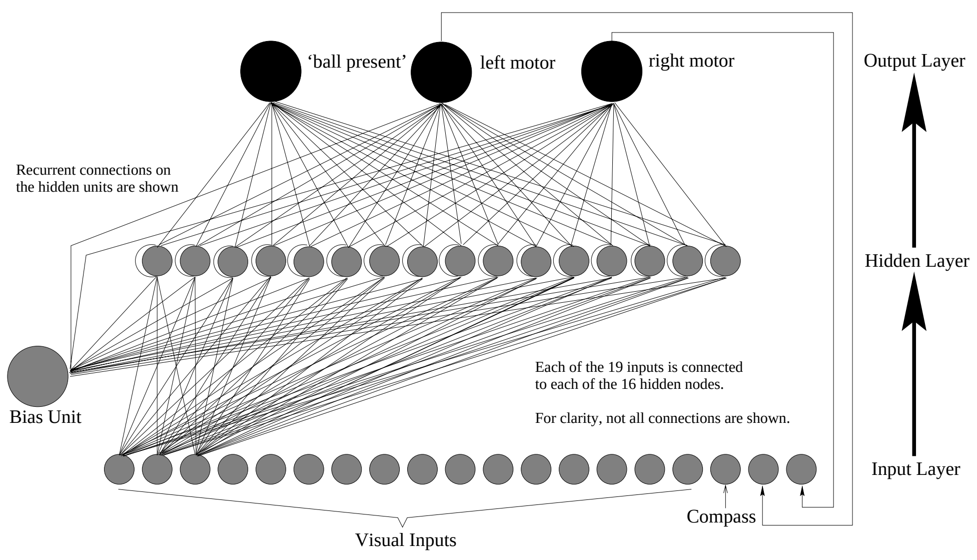

'ball present'
left motor
right motor
Output Layer
Recurrent connections on the hidden units are shown
Hidden Layer
Bias Unit
Each of the 19 inputs is connected to each of the 16 hidden nodes.
For clarity, not all connections are shown.
Input Layer
Compass
Visual Inputs

Figure 9.3: The new feed-forward neural network

possibly not being able to see the ball when pushed right up against it, an extra recurrent connection is included on the hidden layer units back to themselves (active on the next time step). Equation 9.2 shows the new activity for a hidden layer unit:

$$input_i = a_{i,i'}A_{i'} + a_{i,bias} + \sum_{j=0}^{j=N}(a_{i,j}A_j) \tag{9.2}$$

i hidden layer unit
j input layer unit
i' hidden layer unit, previous time step
N no. of input units
$a_{i,j}$ synapse i to j weighting
A_j activity of jth unit

9.2.2 Random Momentum

The problem with the robot circling past the ball is a combination of the *momentum* of the robot, and the non-instantaneous mechanical action of the wheels. The simulation updated the robot position on the basis of the motor outputs set on that time step - in effect the new motor speeds were set instantly and no factor included for previous activity. In reality, the robot would have both angular and linear momentum, thus would not be moving solely on the basis of the new wheel speeds. The time taken to set the new wheel speeds is also a factor - neural network update times of the order 0.1 seconds are comparable to the time taken for the mechanical setting of the wheel speeds.

The first enhancement of the simulation was to average the wheel speeds over the last two time steps, an extreme simplification of the physics involved. Evaluation of the controllers in the real world showed this to be evolving some very odd strategies: robots were jerkily moving back and forth, which in simulation averaged out to be a smooth motion. Including noise into this enhancement introduced the crucial unreliability to the simulation. The simulated controller was updated on a wheel speed chosen randomly between the wheel speeds on the last two time steps, thus the robot cannot simply set wheel speeds and assume that a certain output will be produced:

- The robot is not able to *rely* on the effects of its motor output if this output has changed.
- The robot can reduce the noise on its position update by changing motor speeds infrequently, and by small amounts. Thus the robot moves smoothly.

The controllers evolved under this scheme were extremely efficient in both simulation and the real world, finding the ball quickly and heading towards it. Initial behaviours were fast circling, followed by moving straight towards the ball when in the field of view. A further behaviour was that the robot was able to move both ways towards the ball - overshoot in the circling was quickly corrected by turning back slightly (although this was only seen if the ball hadn't passed out of the field of view). Only controllers evolved with this random momentum scheme showed this behaviour.

9.3 A Footballing Robot

Controllers from the 500th generation were tested in the real word, and the behaviours seen to be extremely similar to those displayed in simulation. Extending the simulation to 3000 generations, the best evolved controller produced extremely good footballing skills in both simulation and reality.

Figure 9.4 shows the population fitness over generations, while 9.5 shows the best controller evaluated over 100 trials. Note that the controller scores in roughly 25% of the trials (a score of near 10000), but only scores near zero on 5% (a score of just over 300 corresponds to dribbling the ball, but away from the goal).

Figure 9.6 shows a simulated evaluation of the best controller - the controller centres quickly on the ball, and pushes forward until the goal is reached. Compare this behaviour with the real video screen-shots in section 9.4, showing the robot dribbling the ball across half the length of the pitch.

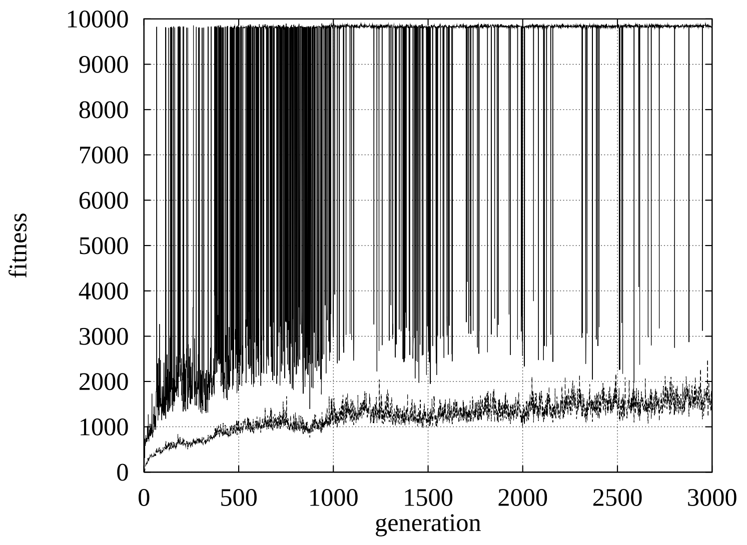


Figure 9.4: Football expt - fitness over generations, median of trial fitnesses

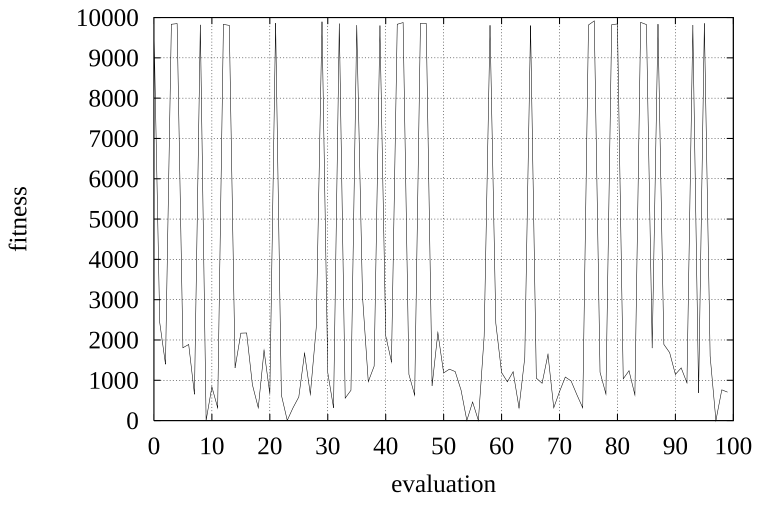


Figure 9.5: Football expt - fitness over trials, median of trial fitnesses

9.4 The Real Robot: Video Footage

This section shows captured video footage of the real robot in action. A Windows-95 `.avi` file was created from videoed footage - the screen-shots shown are postscript versions of `.gif` files extracted from this footage. Consecutive frames are 0.5 seconds apart (the video capture of the robot was updated twice every second).

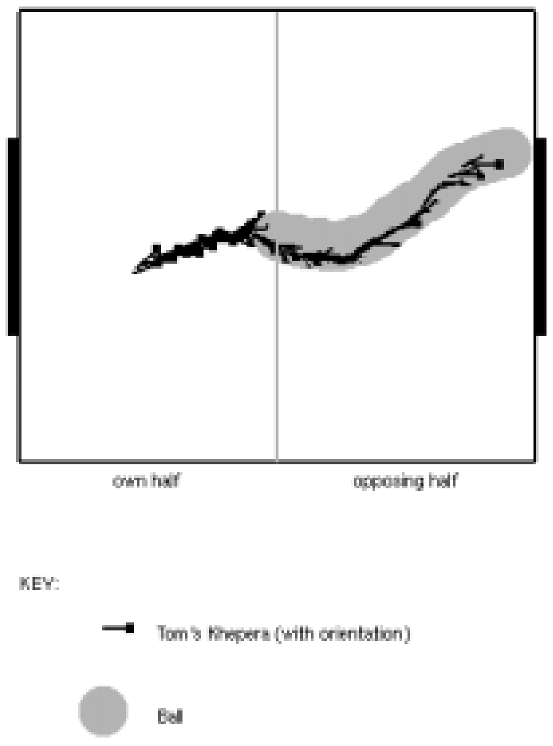


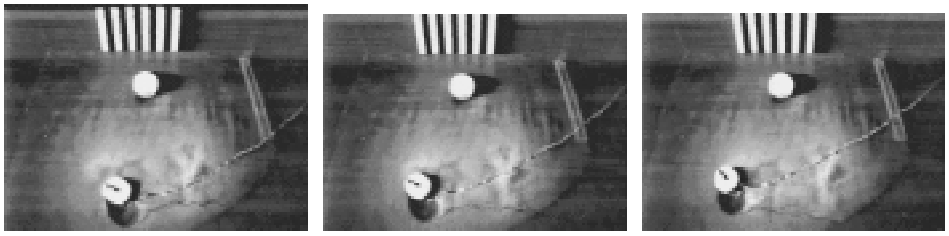
Figure 9.6: Football expt - simulation evaluation. Note, the figure shows the evaluation every 2 updates (roughly 0.5 seconds)

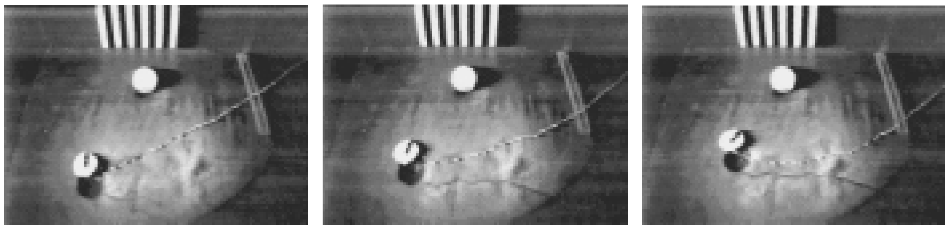
The 30cm ruler on the right of the pictures gives some indication of the scale, and the line on top of the Khepera robot indicates the direction of view of the vision system, i.e. 'facing forward'. For the controller to work consistently, bright light was needed on the pitch[1]. This high lighting level accounts for the glare on the floor. The serial cable shown in the screen-shots provides the robot with power only; the connection with the computer is broken once the control program is downloaded.

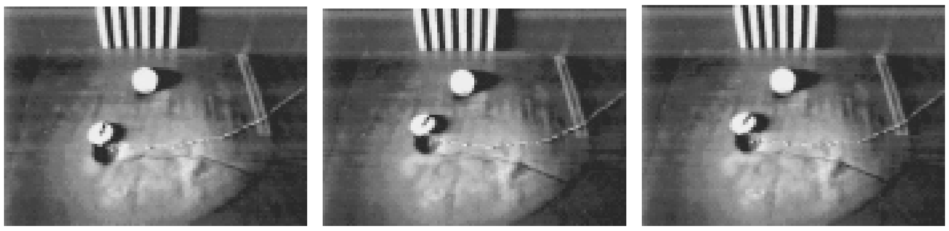
[1]Although the controller was fairly reliable even in fairly dark conditions.

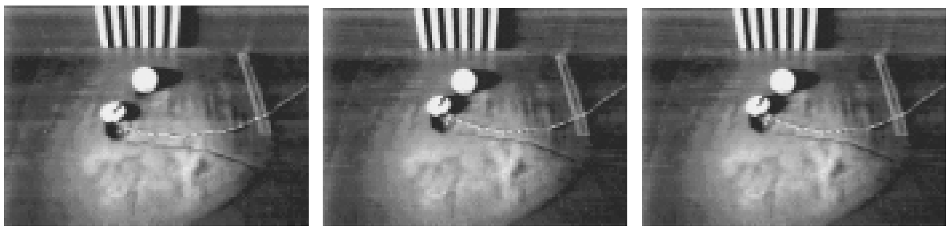
9.4.1 Finding the Ball

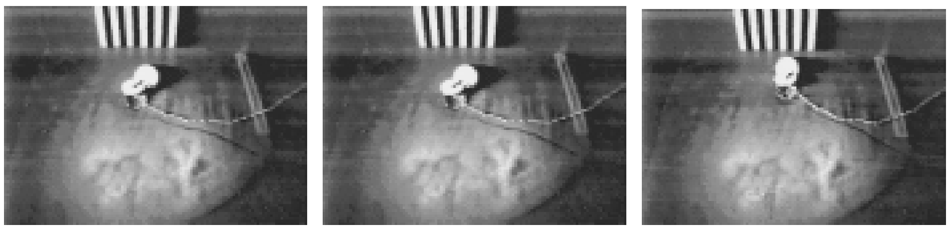
Frames 1-15 show the Khepera (with serial cable still attached) turning sharply to find the ball, and moving towards it, despite the goal also being placed nearby. Once the robot reaches the ball, it carries on pushing forward, 'dribbling' the ball.


frames 1-3


frames 4-6

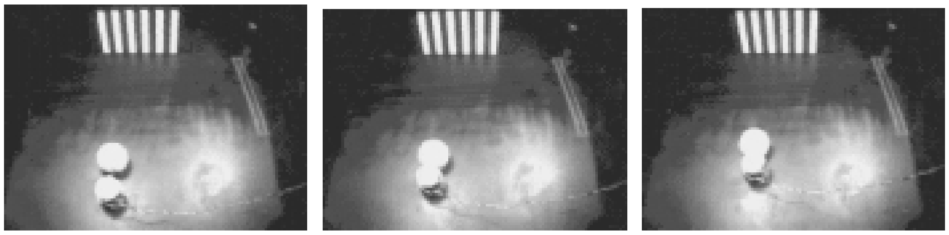

frames 7-9

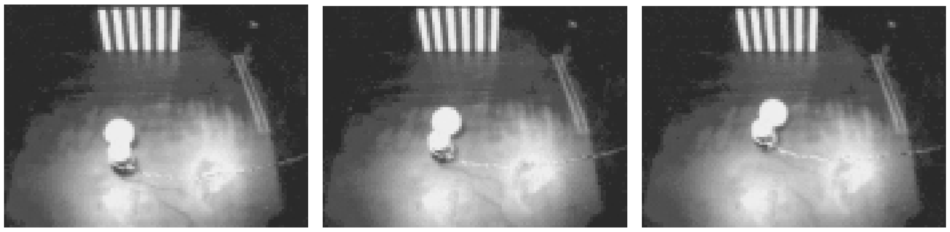

frames 10-12

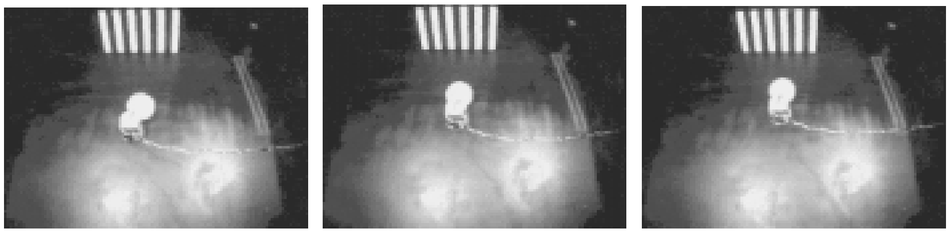

frames 13-15

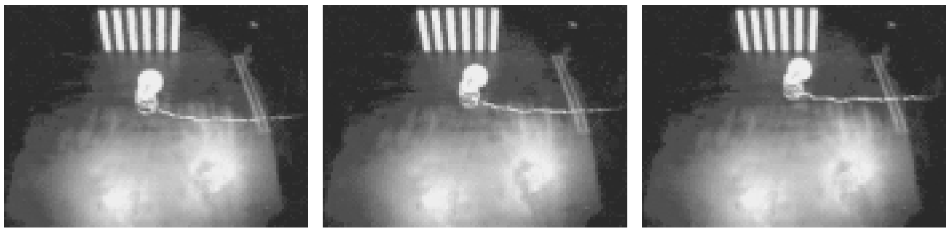
9.4.2 'Dribbling' the Ball

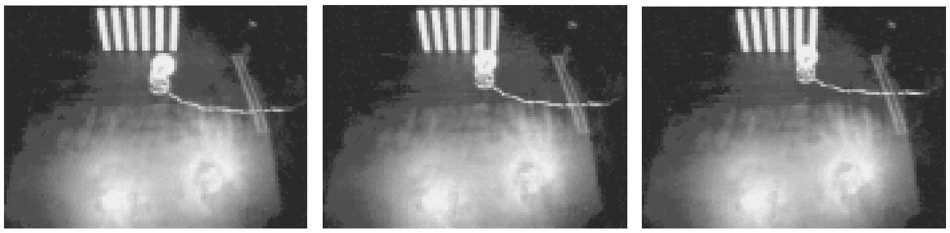
The robot approaches the ball from the bottom of the screen, and dribbles it more than half the length of the pitch (roughly 60cm) towards the opposing goal, finally scoring. The robot keeps the ball centred, despite it rolling away to the right (frames 4 through 8).


frames 1-3


frames 4-6

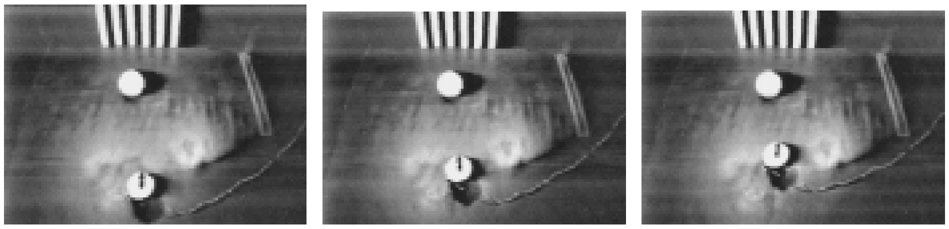

frames 7-9

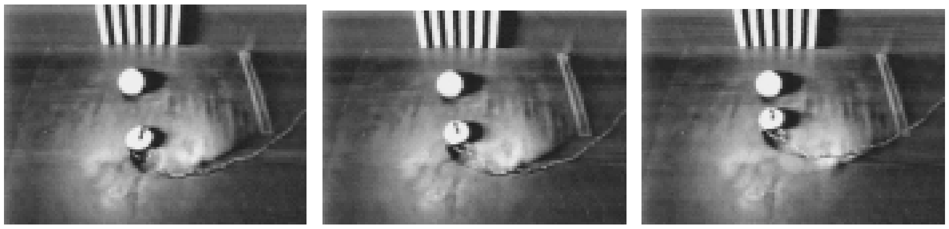

frames 10-12

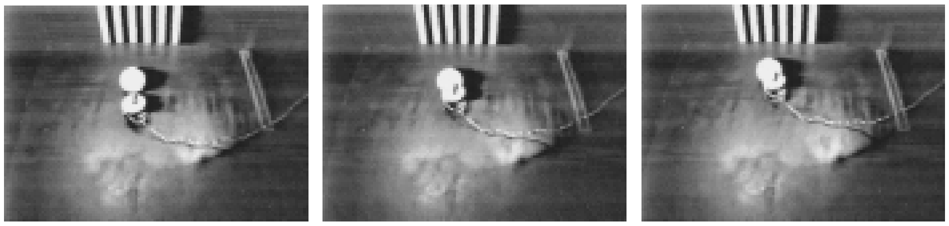

frames 13-14

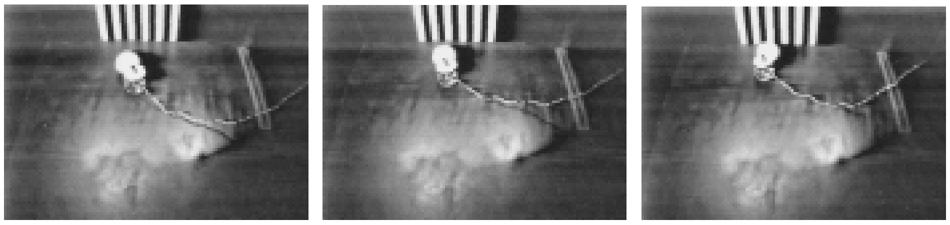
9.4.3 The Penalty Shoot-Out

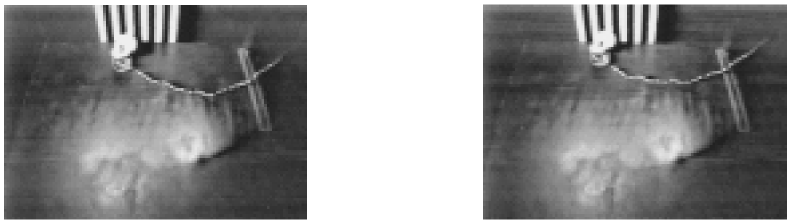
This scenario shows the Khepera (with serial cable still attached) start from the bottom of the picture, and head straight towards the ball, pushing it in to the striped goal. Note the slight turns made in frames 4 and 5 to line the ball up dead-centre.


frames 1-3


frames 4-6

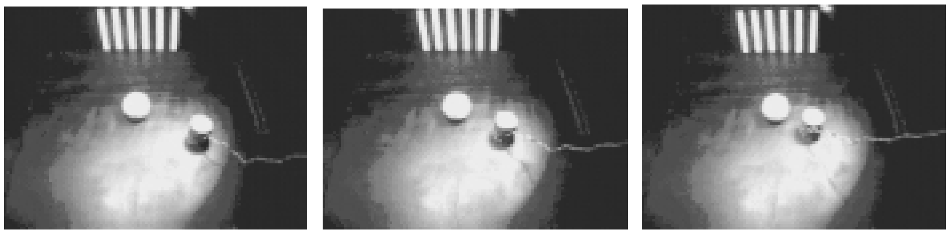

frames 7-9

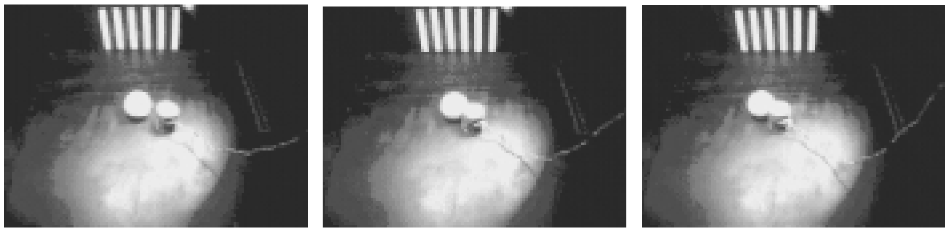

frames 10-12

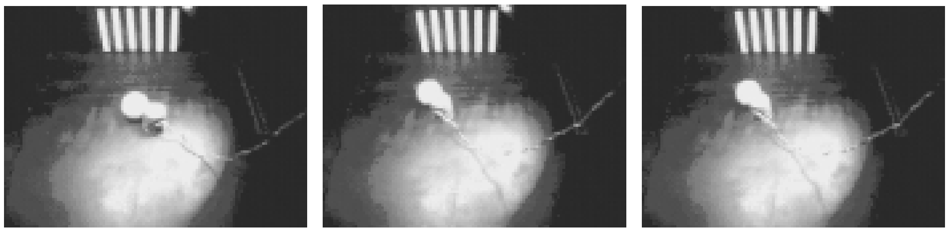

frames 13-14

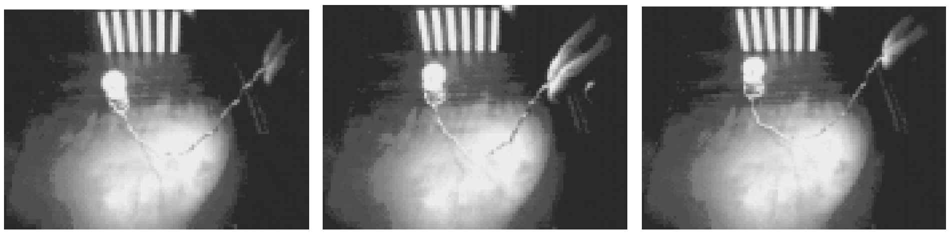
9.4.4 The Trick Shot

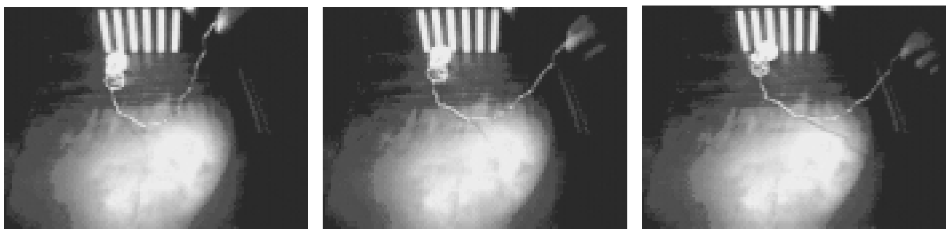
This scenario shows the Khepera approaching the ball from across the goal, yet still turning it into the net.


frames 1-3


frames 4-6


frames 7-9


frames 10-12


frames 13-15

9.5 Neural Network Analysis

Appendix **??** contains full details of the qualitative analysis of the network controller used on the real robot. Figure 9.7 shows the neural network activity of the controller during a simulated evaluation covering 40 seconds (note the visual inputs I0 to I15 start from the left hand side of the visual field, and are fully connected to the hidden layer units H0 to H15, which in turn are recurrently connected to themselves and to the left and right motor neurons, ML and MR).

Analysis of the actual genotype (shown in appendix **??**) gives evidence for the H12 and H14 neurons acting as multi-purpose bright object detectors. Either will saturate positively with a bright object in the right field of view, and saturate negatively with a bright object in the left field (see figure 9.7 - both are high positive on time point 10 when a bright object lies in the right field (inputs I8 to I15), and both are high negative just before time point 90, when a bright object is covering inputs I0 to I7). The behaviour of the right motor is extremely dependent on these two neurons with their high negative synapse weightings. A bright object in the right field causes the right motor to shut down, while an object in left field causes the right motor to speed up, thus the robot turns towards the object.

The H12 neuron also drives the left motor, but the high positive synapse weighting means the motor behaviour is reversed with respect to the right motor. Thus given a bright object in the visual field, the robot will extremely efficiently centre it; the only position where the two motor speeds are equal, at which point the robot will move towards the object. In the absence of any bright object input, the left motor is more fully on than the right (see appendix **??**, figure **??**), thus the robot's base behaviour is to circle clockwise until a bright object enters the visual field, at which point this will be centred and moved towards.

The analysis presented here clearly does not tell the full story. The role of the recurrent connections, how the robot recognises when it is hard up against the ball yet can't see it, and how bright objects of different sizes are discriminated (so the robot does not home in on the striped goal) have not been touched on. However, for further explanation, qualitative analysis of this type is not sufficient and a full complex systems analysis is needed.

In this chapter we have seen how evolution in a noisy minimal simulation can produce robust controllers capable of operating in the real world. The non-trivial task at hand, finding a tennis ball and pushing it towards a goal, is performed in both simulation and the real world - the controllers have successfully crossed the reality gap. For proper analysis of the effectiveness of the controllers, more runs and example controllers are needed. Also, I suspect that the simulation noise level is now slightly too high for the controllers to get much better - seeding the best controllers into a less noisy environment[2] might produce better strategies for getting onto the right side of the ball to push it into the opposing goal every time. However, what we have is a clear indication that good controllers can evolve in simulation in a relatively small number of generations to perform a complex task in the real world.

[2]Similar to the idea that early in development of an animal, accurate vision may be too 'expensive' to process, so vision actually gets more accurate as the animal control system develops.

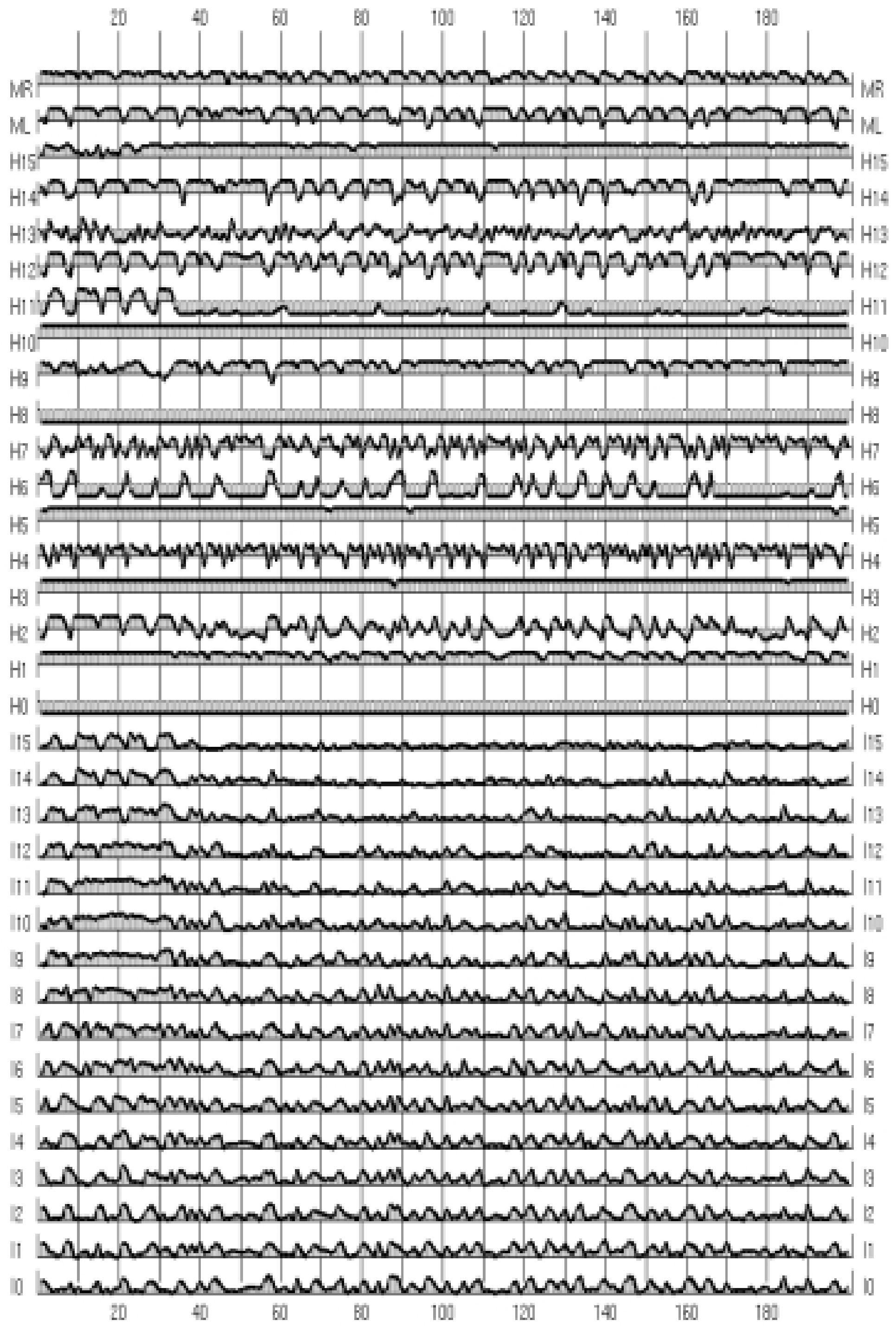


Figure 9.7: Football expt - network activity for best controller in simulated evaluation. I0 to I7 refer to visual inputs on the left side of the visual field, I8 to I15 to inputs on the right side (all inputs are DC offset processed, so there is always at least one zero input).

Chapter 10

Conclusions

10.1 Project Conclusions

We have seen how controllers evolved in a noisy *minimal simulation* can successfully complete a non-trivial task in the real world. A simulation of the Khepera robot and vision system was constructed, incorporating *unreliability* in six key areas:

- Robot-wall interactions.
- Robot momentum.
- Background lighting.
- Object colours.
- Individual vision system pixel performance.
- Vision system pixel input angle.

In addition to these aspects, noise was added on to almost every simulation feature before successful controllers were evolved.

A genetic algorithm evolutionary approach was used to produce neural network controllers mapping input sensor data to output motor behaviour. The evaluation of the robot controllers was carried out in the simulation described above, and selective pressure forced the evolution of better and better controllers. Several visual processing schemes were explored; the project controllers used a simplified version of the predictive coding algorithm on visual input data.

Two scenarios were investigated. The first experiment, finding a single white stripe on a dark background, was used as a test-bed to ensure controllers evolved in simulation were capable of operating in the real world. The second experiment, finding a tennis ball in a black arena and pushing it into a goal, was then used to show the techniques of minimal simulation are capable of scaling up to a real world problem requiring visual environment input data.

Finally, the best controllers were downloaded onto the real robot and tested in reality. Qualitative analysis of their behaviours was used to modify the simulation (in particular, a random momentum element was added) until controllers evolved displaying the same behaviours in simulation and reality - crossing the reality gap. Analysis of the successful controller network showed key neurons acting as bright object detectors in both the left and right fields of view. The same neurons can drive both motors to centre bright objects before moving towards them.

10.2 Further Work

The range of domains covered by the problem of evolving a controller for a sighted robot footballer is huge, and this project can only hope to have investigated a small fraction of them.

The actual robot and vision system used was fixed by the size requirement for the football tournament. However, both pieces of hardware suffered from serious defects, primarily due to their extremely small size. On the robot, unmodellable noise was a major problem; dust picked up from the environment can stop one wheel entirely, producing very biased motion, e.g. spinning on the spot when both wheels are set to constant speed. Of the four Khepera robots used in the project, only one was remotely reliable, and only for fairly short periods of time. The vision system also suffered from problems; the iris effect removed the possibility of telling white from black from the visual data alone. Also, occasionally the system provided very odd data - given a display of a bright stripe on a black background, the returned data sometimes showed a dark stripe on a bright background. More limiting was the problem of the data returned; for serious vision work a one-dimensional grey-scale array does not provide enough visual environment data. As a basic research robot platform, the Khepera is extremely well suited. However, for reliable operation over longer periods of time, performing complex behaviours in a 'real'[1] environment, it is unsuitable.

Evolving in simulation introduces problems with the modelling of a visual environment - it is my belief that only a model based on some type of noisy minimal simulation scheme can cope with the complexity of the real world

[1]Real in the sense of an environment not artificially created for the robot.

visual environment. More realistic simulations are just too hard to design, e.g. [45]. However, this is not to say that the minimal simulation approach is a simple recipe for success. Considerable insight into the problem was needed before a successful simulation was produced.

The use of vision introduced the fields of digital image processing and information theory. To produce controllers operating in more complex visual environments, further investigation into the effects of using more complex processing than used in this project is definitely needed. Predictive coding would seem to be an obvious target of research.

The issue of what type of controller to use, e.g. high-level control programs or low-level primitive controllers such as neural networks, was also skipped over in the project. This project used the latter, but the fixed architecture direct encoding scheme employed allowed no scope for evolution to change the connections and nodes in the network. It would be of interest to investigate more sophisticated encoding schemes employing biological morphogenesis techniques allowing controllers to evolve which are complex enough, but no more than necessary, to produce the behaviour required. Phenotypes of differing complexity will also affect the time dynamics of the system - to match the update speed in simulation to that in reality is now no longer the trivial task outlined in chapter 8. The robot controller must be slowed to some preset update speed - perhaps waiting until 100ms has passed before carrying out the next update -which can be matched to the simulation.

Also of key importance was the genetic algorithm scheme; a spatially distributed population was used, but no exploration into the effects of varying the selection and breeding schemes was undertaken. The interplay between the evolutionary and network encoding/morphogenesis schemes is also crucial; more complex encoding may require a different evolutionary setup, e.g. Species Adaptive Genetic Algorithms [23].

Despite the cursory examination of many vital areas, the project has to be judged a success on three different levels. First, as an evolved controller able to play a limited game of football in the real world. Second, as a basis from which to investigate some of the areas outlined above, to produce a controller capable of playing a 'good' game of football. Third and most important, as a demonstration that the theory of minimal simulation evolution can scale up to produce controllers incorporating visual environment data, performing complex tasks in the real world.

Tom Smith
Sussex, 1997

Bibliography

[1] R. D. Beer and J. C. Gallagher. Evolving dynamical neural networks for adaptive behaviour. *Adaptive Behaviour*, 1:94–110, 1992.

[2] A. T. D. Bennett. Do animals have cognitive maps? *The Journal of Experimental Biology (special vol: Navigation)*, 199:219–224, 1996.

[3] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automaton*, RA-2:14–23, 1986.

[4] R. A. Brooks. A robot that walks: Emergent behaviour from a carefully evolved network. *Neural Computation*, 1(2):253–262, 1989.

[5] R. A. Brooks. Intelligence without reason. In *Proc. 12th International Joint Conference on Artificial Intelligence*. Morgan-Kaufman, 1991.

[6] R. A. Brooks. Intelligence without representation. *Artificial Intelligence (special vol: Foundations of Artificial Intelligence)*, 47:139–159, 1991.

[7] R. A. Brooks. Artificial life and real robots. In F. J. Varela and P. Bourgine, editors, *Towards a Practice of Autonomous Systems: Proc. 1st European Conference on Artificial Life*. MIT Press/Bradford Books, 1992.

[8] R. A. Brooks. Coherent behaviour from many adaptive processes. In D. Cliff, P. Husbands, J. A. Meyer, and S. W. Wilson, editors, *Animals to Animats 3: Proc. 3rd International Conference on Simulation of Adaptive Behaviour*. MIT Press/Bradford Books, 1994.

[9] A. Clark. Happy couplings: Emergence and explanatory interlock. In M. A. Boden, editor, *The Philosophy of Artificial Life*. Oxford University Press, 1996.

[10] A. Clark and J. Toribio. Doing without representing? Cognitive Science Technical Report 310, School of Cognitive and Computing Sciences, University of Sussex, 1994.

[11] D. T. Cliff, I. Harvey, and P. Husbands. Explorations in evolutionary robotics. *Adaptive Behaviour*, 2(1):71–104, 1993.

[12] D. T. Cliff and G. F. Miller. Tracking the Red Queen: Measurements of adaptive progress in co-evolutionary simulations. In F. Moran, A. Moreno, J. J. Merelo, and P. Cachon, editors, *Proceedings of the Third European Conference on Artificial Life*, number 929 in Lecture Notes in Artificial Intelligence, pages 200–218. Springer-Verlag, 1995.

[13] T. S. Collett. Insect navigation en route to the goal: Multiple strategies for the use of landmarks. *The Journal of Experimental Biology (special vol: Navigation)*, 199:227–235, 1996.

[14] K. Dale. First year PhD report. Unpublished, 1997.

[15] D. C. Dennett. Cognitive wheels: The frame problem of AI. In M. A. Boden, editor, *The Philosophy of AI*. Oxford University Press, 1990.

[16] D. Floreano. Re: 'adaptive behaviour in competing co-evolving species' re-creation. Personal email communication, July 1997.

[17] D. Floreano and F. Mondada. Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot. In D. Cliff, P. Husbands, J. A. Meyer, and S. W. Wilson, editors, *Animals to Animats 3: Proc. 3rd International Conference on Simulation of Adaptive Behaviour*. MIT Press/Bradford Books, 1994.

[18] D. Floreano and F. Mondada. Evolution of plastic neurocontrollers for situated agents. In P. Maes, M. J. Mataric, J. A. Meyer, and S. W. Wilson, editors, *From Animals to Animats 4; Proc. 4th International Conference on Simulation of Adaptive Behaviour*. MIT Press/Bradford Books, 1996.

[19] D. Floreano and M. Mondada. Adaptive behaviour in competing co-evolving species. In P. Husbands and I. Harvey, editors, *Fourth European Conference on Artificial Life*. MIT Press/Bradford Books, 1997.

[20] D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[21] R. C. Gonzalez and P. Wintz. *Digital Image Processing*. Addison-Wesley, 1977.

[22] F. Gruau and K. Quatramaran. Cellular encoding for interactive evolutionary robotics. In P. Husbands and I. Harvey, editors, *Fourth European Conference on Artificial Life*. MIT Press/Bradford Books, 1997. also available as Cognitive Science Research Paper 425, School of Cognitive and Computing Sciences, University of Sussex.

[23] I. Harvey. Species Adaptation Genetic Algorithms: A basis for a continuing SAGA. In F. J. Varela and P. Bourgine, editors, *Towards a*

Practice of Autonomous Systems: Proc. 1st European Conference on Artificial Life, pages 346–354. MIT Press/Bradford Books, 1992.

[24] I. Harvey. Evolving robot consciousness: The easy problems and the rest. In G. Mulhauser, editor, *Evolving Consciousness*, Advances in Consciousness Research Series. John Benjamins, 1997. In preparation.

[25] I. Harvey, P. Husbands, and D. T. Cliff. Issues in evolutionary robotics. Cognitive Science Technical Report 219, School of Cognitive and Computing Sciences, University of Sussex, 1992.

[26] I. Harvey, P. Husbands, and D. T. Cliff. Seeing the light: Artificial evolution, real vision. In D. Cliff, P. Husbands, J. A. Meyer, and S. W. Wilson, editors, *Animals to Animats 3: Proc. 3rd International Conference on Simulation of Adaptive Behaviour*. MIT Press/Bradford Books, 1994.

[27] I. Harvey, P. Husbands, D. T. Cliff, A. Thompson, and N. Jakobi. Evolutionary robotics: The Sussex approach. *Robotics and Autonomous Systems*, In preparation, 1997.

[28] J. H. Holland. *Adaptation in Natural and Artificial Systems, 2nd edition.* MIT Press/Bradford Books, 1992.

[29] P. Husbands. Genetic algorithms in optimisation and adaptation. In L. Kronsjo and D. Shumsheruddin, editors, *Advances in Parallel Algorithms*. Blackwell Scientific Publications, 1992.

[30] N. Jakobi. Evolving sensorimotor control architectures in simulation for a real robot. Master's thesis, School of Computing and Cognitive Sciences, University of Sussex, 1994.

[31] N. Jakobi. Evolutionary robotics and the radical envelope of noise hypothesis. *Journal of Adaptive Behaviour*, 6, 1997. In preparation.

[32] N. Jakobi. Half-baked, ad-hoc and noisy: Minimal simulations for evolutionary robotics. In P. Husbands and I. Harvey, editors, *Fourth European Conference on Artificial Life*. MIT Press/Bradford Books, 1997.

[33] N. Jakobi. A minimal simulation for evolving walking behaviour in an octopod robot. Cognitive Science Research Paper 464, School of Cognitive and Computing Sciences, University of Sussex, 1997.

[34] N. Jakobi, P. Husbands, and I. Harvey. Noise and the reality gap: The use of simulation in evolutionary robotics. In F. Moran, A. Moreno, J. Merelo, and P. Chacon, editors, *Advances in Artificial Life: Proc. 3rd European Conference on Artificial Life*. Springer-Verlag, 1995.

[35] S. P. D. Judd, K. Dale, and T. S. Collett. On the fine structure of view-based navigation in insects. In Preparation, 1997.

[36] K-Team. *Khepera: The Users Manual.* K-Team, LAMI-EPFL, 1993.

[37] K-Team. *Khepera: The Low Level BIOS.* K-Team, LAMI-EPFL, 1994.

[38] D. Kirsh. Today the earwig, tomorrow man? *Artificial Intelligence (special vol: Foundations of Artificial Intelligence)*, 47:161–185, 1991.

[39] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. RoboCup: The robot world cup initiative. In *Proc. First International Conference on Autonomous Agents.* The ACM Press, 1997.

[40] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press/Bradford Books, 1992.

[41] S. B. Laughlin. Form and function in retinal processing. *Trends in Neuroscience*, 10(11):478–483, 1987.

[42] H. H. Lund and O. Miglino. From simulated to real robots. Technical report, Institute of Psychology, National Research Council, Rome, 1996.

[43] M. J. Mataric. A distributed model for mobile robot environment learning and navigation. Technical Report 1228, MIT AI Lab, 1990.

[44] M. J. Mataric. Evaluation of learning performance of situated embodied agents. In F. Moran, A. Moreno, J. Merelo, and P. Chacon, editors, *Advances in Artificial Life: Proc 3rd European Conference on Artificial Life.* Springer-Verlag, 1995.

[45] M. J. Mataric and D. T. Cliff. Challenges in evolving controllers for physical robots. *Robotics and Autonomous Systems*, 19(1):67–83, 1995.

[46] F. Mondada. Re: K213 vision module iris. Personal email communication, July 1997.

[47] F. Mondada, E. Franzi, and P. Ienne. Mobile robot miniaturization: A tool for investigation in control algorithms. In *ISER '93*, Kyoto, Japan, October 1993.

[48] N. J. Nilsson. Shakey the robot. Technical Note 323, SRI International, California, 1984.

[49] S. Nolfi, D. Floreano, O. Miglino, and F. Mondada. How to evolve autonomous robots: Different approaches to evolutionary robotics. In R. A. Brooks and P. Maes, editors, *Artificial Life IV: Proc. 4th International Workshop on the Synthesis and Simulation of Living Systems.* MIT Press/Bradford Books, 1994.

[50] S. Nolfi and D. Parisi. Evolving non trivial behaviours on real robots: An autonomous robot that picks up objects. In *Proc. 4th Congress of the Italian Association for AI*. Springer-Verlag, 1995.

[51] R. J. Schalkoff. *Digital Image Processing and Computer Vision*. John Wiley and Sons, 1989.

[52] A. K. Seth. 1st year PhD report. Unpublished, 1997.

[53] M. V. Srinivasan, S. B. Laughlin, and A. Dubs. Predictive coding: A fresh view of inhibition in the retina. *Proc. Royal Societ London, Series B*, 216:427–459, 1982.

[54] A. Thompson, I. Harvey, and P. Husbands. Unconstrained evolution and hard consequences. In E. Sanchez and M. Tomassini, editors, *Towards Evolvable Hardware*. Springer-Verlag, 1996.

[55] B. Webb. Robotic experiments in Cricket phonotaxis. In D. Cliff, P. Husbands, J. A. Meyer, and S. W. Wilson, editors, *Animals to Animats 3: Proc. 3rd International Conference on Simulation of Adaptive Behaviour*. MIT Press/Bradford Books, 1994.

[56] M. Wheeler. From robots to Rothko: The bringing forth of worlds. In M. A. Boden, editor, *The Philosophy of Artificial Life*. Oxford University Press, 1996.