Why Is My Hoover So Stupid? A Platform for Evolving Intelligent Mobile Robots

Richard Towers
Supervisor: Professor V. Vitanov

Abstract— **The importance of developing intelligent robotic agents is well known, as is the difficulty of the task. The evolutionary method offers a solution to the difficulty of manually designing complex control systems. This paper presents a new platform for conducting evolutionary experiments on mobile Khepera robots and the strengths and limitations of the platform are investigated. The results of some preliminary experiments evolving control systems for robotic vacuum cleaners are presented. It is concluded that the platform shows excellent promise in its ability to synthesise control systems and recommendations are made for future work.**

Index Terms— **Evolutionary Computation, Cognitive Robotics, Mobile Robots**

I. Introduction

"Machines will be capable, within twenty years, of doing any work a man can do."
[16, 1965]

FORTY YEARS after Simon made this statement, robotics is just beginning to edge into the consumer domain with products such as the iRobot® Roomba™ Robotic Floorvac developed by Brooks, Jones *et al.* [7]. Whilst the commercial success of this product is impressive, it can hardly be said to be *intelligent*, and it certainly cannot "do any work a man can do". The reason for this failure to live up to expectations is simple, designing systems that exhibit intelligent behaviour is enormously more difficult than researchers first assumed.

The situation is not all gloomy though, natural evolution provides us with countless examples of intelligent systems, from the amazingly simple[1] to the astoundingly complex[2]. By using artificial evolution we may hope to emulate nature's success.

The feasibility of using artificial evolution to develop the control systems and morphology of robots has been demonstrated by many authors [12], [15], [11], [9], however, as Brooks argues, "To compete with hand coding techniques it will be necessary to automatically evolve programs that are one to two orders of magnitude more complex than those previously reported in any domain"[3]. Control systems of this level of complexity have never been evolved, although work such as [6] suggests that, for some problems at least, the evolutionary method is producing control systems that are "better ... than those reported in any other domain", although perhaps not by orders of magnitude.

[1]The roundworm C. Elegans has only 302 neurons
[2]The primate Homo sapiens has over 80,000,000,000 neurons

This paper aims to further demonstrate the feasibility of the evolutionary method, specifically using embodied evolution conducted on real robots. To demonstrate that the method can be more effective than manual design a control system for a device similar to the commercially successful Roomba vacuum cleaner is evolved. For a number of reasons the control system of choice is an artificial neural network (ANN). The evolvabilities of two common types of ANN, the simple feed-forward architecture (FFNN) and the continuous time recurrent neural network (CTRNN), are investigated.

The paper is organised as follows. The relevant theory relating to evolutionary algorithms and neural networks is presented in Section II. A description of the platform designed to run the embodied experiments is given in Section III and the strengths and limitations of the platform are discussed. The results of evolving controllers for vacuum cleaners are presented in Section IV and the quality of the best controllers is evaluated. Finally, in Section V, conclusions about the efficacy of the method are drawn and recommendations for future work are made.

II. Theory

The platform described in Section III relies heavily on two computational methods. Neural networks are used as control systems, and evolutionary algorithms are used to determine the network parameters. Both methods are appealing for their biological plausibility, since we are trying to emulate nature's success it seems apt that we should mimic its methods closely.

A. Neural Networks

Artificial neural networks are all, to a greater or lesser extent, designed to mimic nature's chosen method of computation. This section covers the theory behind artificial neural networks, the most complex are presented first with each subsequent architecture making some simplifying assumptions to allow for faster computation and easier analysis, usually at the cost of computational power and evolvability.

Firstly let us consider biological networks. A biological network has a number of interconnected neurons which communicate either through chemical synapses or electrical gap junctions. At a synapse there will be a number of dendrites from incoming *presynaptic* neurons and (usually) one soma from a *postsynaptic* neuron. A neuron will only fire if the polarisation across its membrane (produced by the summation of the presynaptic potentials) exceeds some threshold, in which case the neuron will produce an action potential (sometimes

$$\begin{tikzcd} A \arrow[r] & B \arrow[r] & C \arrow[ll, bend left] \end{tikzcd}$$

(a) A directed loop, the arrows can be followed in a complete cycle

$$\begin{tikzcd} A \arrow[r] & B & C \arrow[l] \arrow[ll, bend left] \end{tikzcd}$$

(b) Not a directed loop, the arrows cannot be followed in a complete cycle

Fig. 1: Directed loops: A network can be said to be recurrent if and only if it contains at least one directed loop

called a *spike*). Synaptic transmission can be excitatory or inhibitory depending on the type of neurotransmitter. The modern view is that information is not just encoded by the rate of spikes, but also by the precise timing of the spikes[8]: a pulse-code as well as a rate-code.

By far the most biologically plausible class of ANNs are the so called *third generation*: spiking neural networks (SNNs). In SNNs neurons are modeled as leaky integrate-and-fire systems with each spike being modeled separately. These networks have been shown to have significant advantages over simpler networks in terms of both computational power and evolvability, however they are much more computationally expensive to simulate and significantly more difficult to build and analyse. For these reasons spiking neural networks are not covered in this study, although further work could be done in this area.

Continuous Time Recurrent NNs can be arrived at from SNNs by making some simplifying assumptions. Assuming that information is transmitted across synapses only in terms of spike rate (as mentioned, biologically speaking this is not the case) there is no need to model every spike. Neurons need only concern themselves with the rate of incoming spikes, which can be assumed to be the sum of the activation states of all presynaptic neurons, adjusted by synaptic weights. Once this assumption is made neurons can be modeled as simple dynamic systems, where the change in activation is given by equation (1).

$$\dot{y_i} = \frac{-y_i + \sum_{j=1}^{n} w_{ji}\sigma(y_j - \theta_j) + I_i(t)}{\tau_i} \tag{1}$$

Where $\dot{y_i}$ is the rate of change of activation, y_i the activation of the postsynaptic node, y_j the activation of the presynaptic node, w_{ji} is the weight of the connection from pre to postsynaptic node, $\sigma(x)$ is the sigmoid of x (in this case $\frac{1}{1+e^{-x}}$), Θ_j is the neuron's bias, τ_i is the time constant of the neuron and $I_i(t)$ is the input (if any) to the node.

A network can be said to be recurrent if and only if it contains at least on directed loop (Figure 1a), CTRNNs make use of the ability of these loops to sustain patterns of activation, allowing a sort of short term memory. Because of this ability to capture and store time dependent phenomena they are capable of displaying a very wide range of behaviour, including limit-cycles and chaotic effects such as strange attractors. CTRNNs with an arbitrary number of hidden neurons are known to be able to approximate any smooth dynamic system[5].

Strictly speaking the term "feed forward" refers to any network with no directed loops (Figure 1b). The neuron and synapse models are irrelevant, so a SNN or a CTRNN is feed forward so long as it doesn't have any loops. Forbidding these loops makes the analysis of networks significantly easier, but much interesting dynamic behaviour is lost. On the face of it this seems like a bad thing, however, in practice, robots that have these recurrent connections (especially neurons with self connections) often exhibit unstable behaviour that results in very low fitness scores, so removing them simplifies the fitness space significantly. When only reactive behaviour is required feed forward networks are usually adequate.

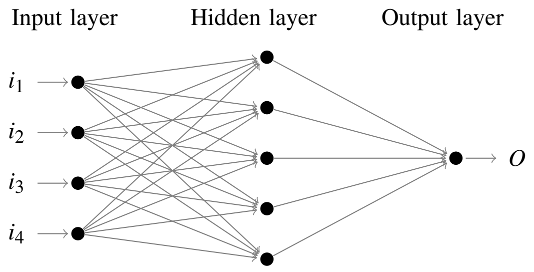


Fig. 2: A diagram of a simple Feed Forward Neural Network with 4 input, 5 hidden and 1 output nodes. Circles denote neurons and arrows denote connections.

The behaviour of the neurons described so far is heavily dependent on their parameters (the bias Θ and the time constant τ), these increase the parameter space of possible networks significantly and this makes genetic search more difficult. If a network is recurrent it is necessary to have neuron models with time constants to prevent the instability caused by feedback, however if networks are feed forward the time constants can safely be dropped. In this study the term FFNN is used to refer to a network of this type, that is:

- There are no recurrent connections
- Neurons respond immediately to their input

Figure 2 shows a typical FFNN. Using the assumptions above then, for each neuron, the output at the next time step is given by equation (2).

$$y_{i(t+1)} = \sum_{j=1}^{n} w_{ji}\sigma(y_{j(t)} + \theta_j) + I(t) \tag{2}$$

B. Evolutionary Algorithms

EAs are stochastic search algorithms that are intended to mimic the process of biological evolution. The general principal is the same: through the ability of random mutations to occasionally produce improvements, and through the ability of selection to preserve these improvements in future generations, over a period of many generations the population will improve. The general form of EAs is shown in Figure 3. EAs are by no means the only appropriate search algorithm for tuning network parameters, other processes such as simulated annealing or tabu search could equally be applied. EAs are used in this study primarily for similarity to the biological process being emulated, further work could be done comparing their performance to other search algorithms.

When thinking about any multi-parameter optimisation process it is useful to consider a *fitness landscape* (figure 4).

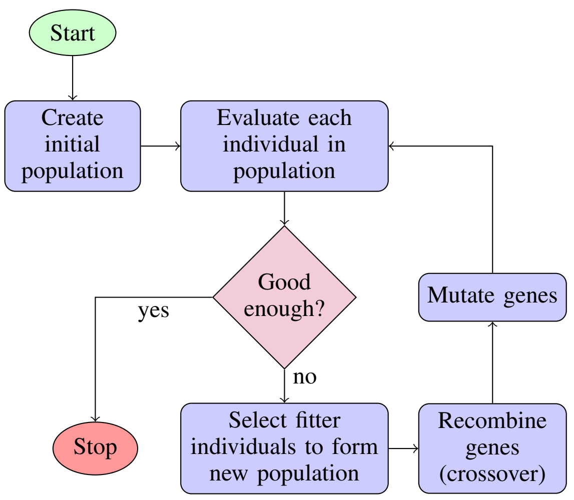

Start
Create initial population
Evaluate each individual in population
Good enough?
yes
no
Stop
Select fitter individuals to form new population
Recombine genes (crossover)
Mutate genes

Fig. 3: A flow diagram showing the general form of evolutionary algorithms

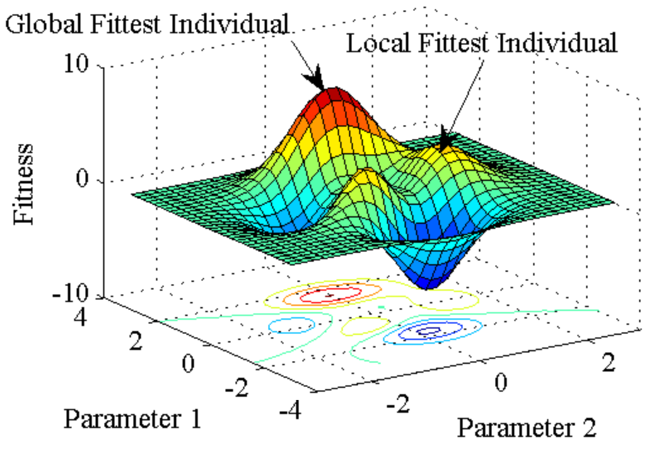
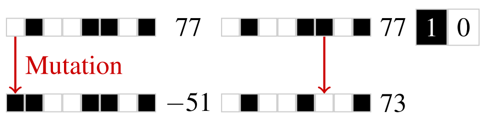
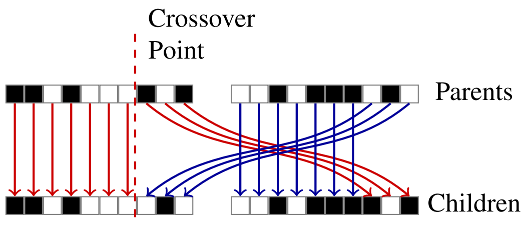


Fig. 4: A cartoon example of a fitness landscape. The dimensionality of the fitness landscape is always $n+1$, where n is the number of parameters. In this case there are two parameters to be tuned and one fitness value, resulting in a three dimensional surface.



(a) The effect of point mutation on signed, 8-bit binary strings



(b) The crossover operation

Fig. 5: Diagrams showing the genetic operations. Chains of boxes represent genomes encoded as binary strings, numbers show the magnitude of the string in decimal.

Where an explicit fitness function is available there will be single fitness value for every combination of parameters. The fitness landscape refers to the multi-dimensional surface traced by these fitness values. In robotics the picture is slightly less clear since robots with the same genotype can score different fitness values depending on things like initial position, initial heading and sensor noise, but the principal is the same. When optimising ANN parameters directly, fitness landscapes often have *thousands* of dimensions, and so, unlike in figure 4, direct visualisation is obviously impossible, this illustrates the need for heuristics.

One of the biggest difficulties in evolving neural networks is the problem of genotype-phenotype (G-P) mapping. A robot's phenotype is the expression of its genotype under the influence of the environment, it is this that determines its behaviour and fitness. The genetic operators of the EA act upon the genotype, not the phenotype, as a result the way that the genotype encodes the control system has a huge impact on the shape of the fitness landscape and thus the performance of the evolutionary search. In this study a direct *one-to-one* mapping is used, wherein the value of every network parameter is directly encoded into the genotype by means of a binary string. This mapping is the simplest from the point of view of platform development, however it becomes unwieldy for large networks. In future work a more biologically plausible scheme such as the growth scheme proposed in [12] is likely to demonstrate much better evolvability and more compact genotypes.

In natural evolution the selection process is taken care of by Darwin's "survival of the fittest", wherein stronger individuals are more likely to survive to the point of reproductive success. In artificial evolution the reproductive process is completely artificial and as a result there needs to be some artificial method of selection. When deciding how to select individuals for reproduction there are two conflicting criteria that need to be considered, average fitness should be maximised by giving fitter individuals better reproductive chances (selection intensity), but, at the same time, loss of diversity in the population must be avoided. This is known as the *exploration-exploitation* trade off.

In this study tournament selection is used as it provides higher selection intensities for the same loss of diversity compared to other methods[1]. Tournament selection involves selecting a random sample from the population of size t and choosing the fittest individual for the next generation, this is repeated N times where N is the size of the population. [2] provides a comprehensive analysis of tournament selection, the loss of diversity is given by (3) and selection intensity is given by (4).

$$p_d = t^{-1/t-1} - t^{-t/t-1} \tag{3}$$

$$I(t) = \int_{-\infty}^{\infty} \frac{t\,x}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \left(\int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{y^2}{2}}\, dy \right)^{t-1} dx \tag{4}$$

Figure 5 shows the operation of mutation and crossover. Without these operators the average fitness of the population would still improve, tending towards the fitness of the best individual, but there would be no exploration of the fitness landscape - no new solutions evaluated.

Mutation is usually the primary force driving exploration. So long as the mutation rate is low children produced will be near to the parents in terms of the parameter space. A thorough search of the fitness landscape is accomplished by having a large population of individuals with initially random genotypes and using mutation to explore the local regions.

Crossover is a more intrusive operation, many more bits are flipped and several parameters can change at once. Children are generally much further from their parents in parameter space than they would be after a point mutation and this can often mean that they score very low fitnesses, although more of the space is explored. With a naïve G-P mapping scheme this problem can be significant [12], for this reason the crossover rates used in this study are extremely low.

III. Development of the Platform

A. *Aims*

This study aims to investigate the design of control systems for commercial robots, specifically focusing on the iRobot Roomba vacuum cleaner. The specific goal is to demonstrate that the evolutionary method can produce control systems of comparable, if not superior, quality to those hand designed control algorithms used on the Roomba.

More generally, as a platform for running embodied evolutionary experiments, several issues need to be addressed:

- The system should be able to run (after setup) with no human interaction for periods of up to several days
  - Fitnesses must be evaluated automatically (i.e. with no human interaction)
  - Damaging collisions must be avoided
  - There should be some way of continuously recharging robots during experiments
- The evolutionary process should be as time efficient as possible, since the time scales involved are so long
  - Genotype encoding must be compact and evolvable
  - Fitness functions must be carefully designed to give high evolvability without compromising evolved behaviour
  - Fitness evaluation, as the most time costly element, needs to be done efficiently
- The platform should be extensible for use in future work, for example it should be simple to switch between modules such as control systems and genetic algorithms to investigate their effect.

B. *The Robots*

This study uses version III Kheperas which are small, cylindrical, two-wheel differential drive robots with a large array of sensors - six ultrasound and nine active and passive infrared (IR). For simplicity only the motors and active IR sensors are used. The IR sensors provide a voltage which is related (in a non-linear fashion) to the distance from the sensor to the nearest reflective object. These voltages are converted

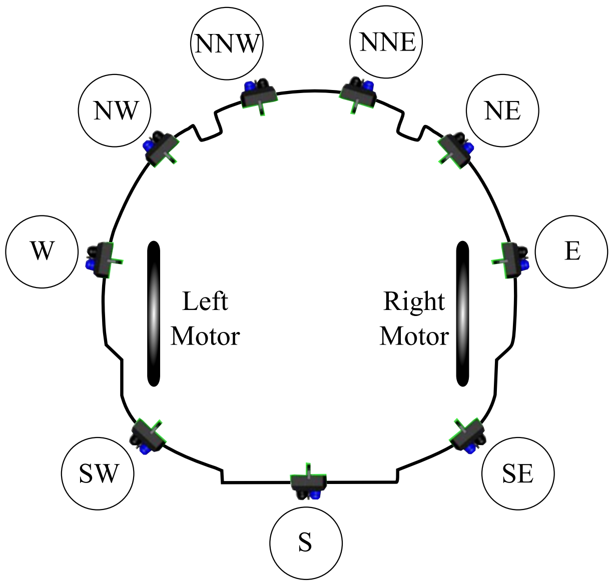


Fig. 6: A plan view of the Khepera III robot showing IR sensor positions and the arrangement of the wheels.

into binary values by analogue-to-digital converters, fed as inputs to the neural network which produces output values which are converted to analogue, boosted and fed to the two motors. Because the robot is differential drive the outputs of the motors can specify any curvilinear path, including on the spot rotation. Figure 6 shows Khepera's sensory layout.

C. *Platform Structure*

Figure 7 shows the logical flow of the platform and figure 8 shows its physical arrangement. All of the genetic processes, including fitness evaluation, are executed on a single supervising computer which communicates with the robots over a wireless network. In theory up to thirty robots could be controlled over such a network[13]. The robots themselves need only be concerned with receiving instructions and running the neural networks - they have no knowledge of their surroundings aside from that gathered from their sensors. The supervising computer on the other hand can obtain much more data from the environment, using a camera it can track the absolute positions of robots and obstacles and can therefore intervene to prevent collisions.

D. *Fitness Monitoring*

Fitness evaluation is an essential part of any evolutionary algorithm. To evaluate the fitness function it is necessary to have some knowledge of the robot's behaviour. Nolfi and Floreano use a laser tracking device to monitor robot position[12], this method provides very precise readings which can be used to follow behaviour and calculate fitness. Laser tracking is very expensive however, and does not scale even to moderately large simultaneous monitoring. A cheaper and more scalable method is proposed by Cupertino *et al.*[4] who use a webcam and a computer vision application to track robots. A similar method is used in this study. Using a small

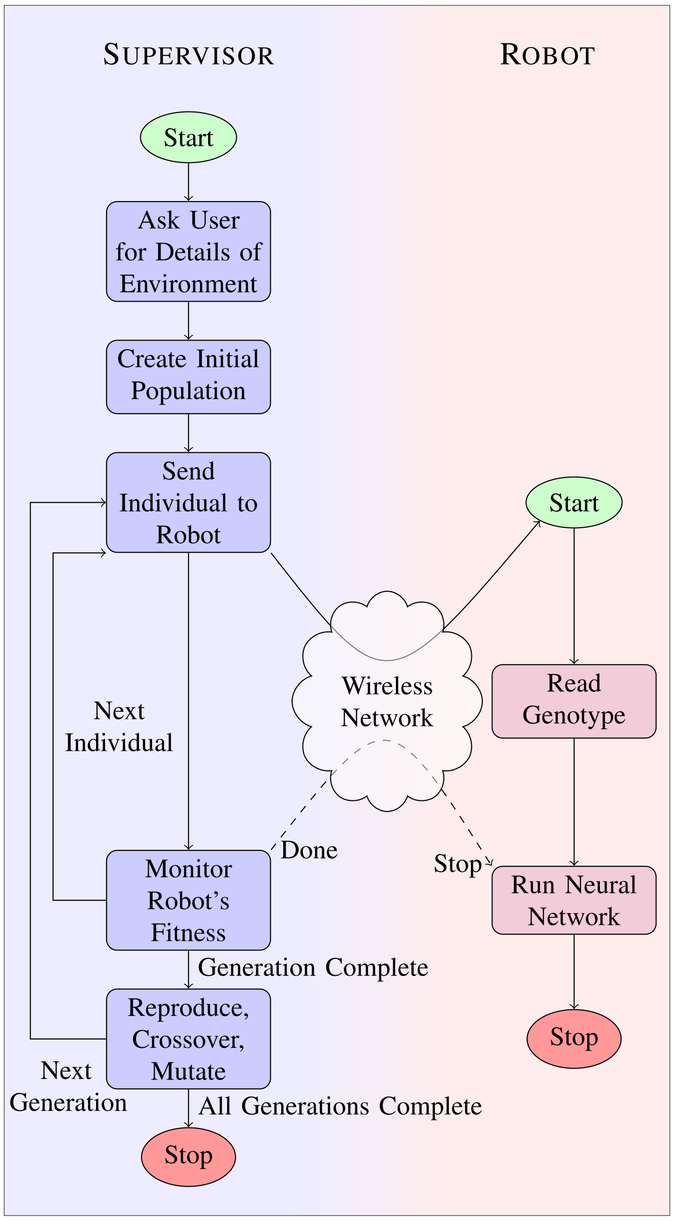


Fig. 7: Flow diagram showing the program outline.

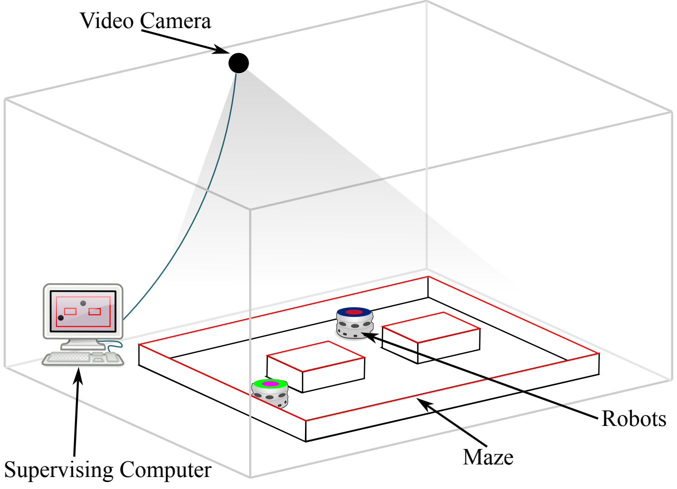


Fig. 8: The experimental apparatus (not to scale). The maze can be of arbitrary shape and complexity. Computer icon modified from [14].

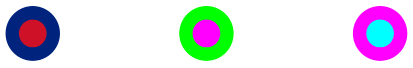


Fig. 9: Various roundels used for robot tracking.

Logitech™ webcam and the OpenCV libraries a lightweight, low cost, scalable solution to robot tracking is realised.

In the "Ask User for Details of Environment" section of figure 7 several pieces of data are acquired, figure 16 (in the appendix) shows screenshots of each step of this process.

Firstly the boundaries of the environment are specified by the user. This aids fitness evaluation and enables the supervisor to intervene if the robot is about to crash.

Next a snapshot of the background (the maze *without* the robot) is taken. When the robot is within the area this snapshot is subtracted from the full image so that the location of the robot (the only difference between the two images) is easy to determine (this is similar to the blue-screen technique used by weather presenters).

Finally the robot is placed in the environment and the user is required to select the area it occupies in the image. The sub-image representing the robot is stored and later correlated with the full image. The area of highest correlation corresponds to the robot's position. Since correlation is only performed in a linear fashion (the image will not be rotated but the robot can be) it is preferable for the images of the robots to be infinitely rotationally symmetrical, because of this the robots are marked with roundels such as those shown in figure 9. By using different roundels several robots can be distinguished, or if heading information is required robots can be fitted with a roundel at either end.

As it stands the application is capable of tracking up to three robots simultaneously in real time before the processing required between frames causes an unacceptable drop in frame rate. Significant improvements in speed (and hence maximum number of robots) should be possible by restricting the search area for each robot to a small region around its previous position, perhaps using a Kalman filter, or by reducing the resolution of the feed.

The maximum area that can be monitored by one camera at ceiling height is around 4m^2, to increase this either a higher ceiling or a wider angle lens would be required. For very large areas it should be possible to sew together images from an array of cameras.

E. Evolutionary Algorithm

The difficulty of genotype to phenotype mapping has already been covered in section II-B. In this study direct encoding of all network parameters has been used because developing a more compact, biologically plausible and evolvable scheme was infeasible in the time frame. Further work in this area could significantly increase the time efficiency of the evolutionary process.

Table I shows the structure of genotype files. The first row determines the architecture of the network (number of neu-

Structure:	2	3	1			
Biases:	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6
Time Constants:	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6
Weights:	w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{16}
	w_{21}	w_{22}	w_{23}	w_{24}	w_{25}	w_{26}
	w_{31}	w_{32}	w_{33}	w_{34}	w_{35}	w_{36}
	w_{41}	w_{42}	w_{43}	w_{44}	w_{45}	w_{46}
	w_{51}	w_{52}	w_{53}	w_{54}	w_{55}	w_{56}
	w_{61}	w_{62}	w_{63}	w_{64}	w_{65}	w_{66}

TABLE I: The structure of a genotype file. Shown with 2 input, 3 hidden and 1 output neurons

rons) and the remainder encodes network parameters. Because the architecture is specified in this way the genotypes can be variable length, allowing the genetic algorithm to add or remove neurons as well as simply modifying weights, biases and time constants.

Each value in table I is the decimal representation of an 8-bit signed integer, so can range from -128 to $+127$. Once these files have been read into memory the genetic operators can act on these numbers in a bitwise fashion (figure 5 on page 3).

Files are sent from the supervising computer to the robots across the wireless network using the UNIX secure copy (scp) utility. On arrival values are read into memory as floating point numbers in the range required by the neural network ($-1 \lesssim x \lesssim +1$). The robots make no alterations to the genotype files - all genetic processes are performed on the supervising machine.

Evolutionary parameters are currently set at compile time, for this study the parameters are set as follows. Population size is set to 10 as a compromise between search speed and thoroughness. Tournament size is set to 4 as a compromise between loss of diversity ($p_d = 0.47$, from eqn. 3) and selection intensity ($I = 1.05$, from eqn. 4). The probability of point mutation is set to 0.05 (too high a rate results in random genetic drift, too low and evolution is slowed down). Finally, the probability of crossover is set to zero for the reasons described in section II-B.

F. Neural Control System

Two neural architectures are implemented, feed forward and continuous-time recurrent. The feed forward architecture is simpler to implement since neuron states explicitly depend only on current state and input. Neuron states in a FFNN can be calculated directly from equation 2.

The implementation of a continuous time network on digital architecture is slightly more complex. Since the dynamic system given by equation 1 does not directly relate future states to present states, but instead gives the rate of change of state some integration is necessary. Analytical integration is infeasible, and so numerical methods are required. Currently two fixed step methods are implemented, the first order Euler method and the fourth order Runge Kutta method.

In the Euler method the neuron state at $t+\Delta t$ is given by equation 5.

$$y_{i(t+\Delta t)} = y_{it} + \Delta t \cdot \dot{y}_i \tag{5}$$

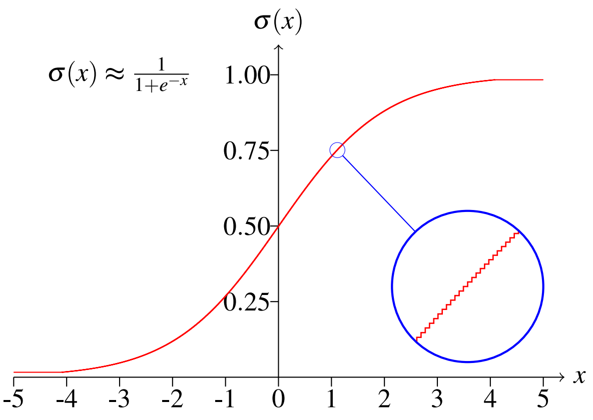


Fig. 10: The lookup table powered logistic function.

The Euler method is appealing for its simplicity. Since the calculations are being done in real time, simple, fast integration allows the use of smaller time steps which should increase numerical accuracy. However, the Euler method is known to converge slowly as time steps shrink and therefore, even though it is faster to compute, it has lower accuracy and stability than higher order methods.

In the Runge Kutta method the neuron state at $t+\Delta t$ is given by equation 6.

$$y_{i(t+\Delta t)} = y_{it} + {}^1\!/_6 \cdot \Delta t \cdot (k_1 + 2k_2 + 2k_3 + k_4) \tag{6}$$

where

$$\begin{aligned} k_1 &= \dot{y}(y_{it}) \\ k_2 &= \dot{y}(y_{it} + {}^1\!/_2 \cdot k_1 \cdot \Delta t) \\ k_3 &= \dot{y}(y_{it} + {}^1\!/_2 \cdot k_2 \cdot \Delta t) \\ k_4 &= \dot{y}(y_{it} + k_3 \cdot \Delta t) \end{aligned}$$

This method gives much higher numerical accuracy and is therefore used in all experiments in this study.

The resolution in time of both types of neural network depends on the time taken to calculate each new state. Since the sigmoid function must be evaluated n^2 times (where n is the number of neurons) its performance is critical to the overall speed of the network. To maximise speed a stepwise lookup table is created for the sigmoid function, since a memory read is much faster than evaluating the logistic function this causes a significant improvement. Figure 10 shows the sigmoid lookup table.

G. Output

The primary output of an evolutionary run is the gene file of the fittest individual, this allows the control system to be implemented where it is needed in the real world. To aid the analysis of the platform's performance several secondary outputs are also produced.

The supervising computer stores the gene file of every individual from every generation, each file is given a descriptive name of the form `GenXIndY.txt` where `X` is the generation number and `Y` is the individual number. Having the complete genotypic history allows the evolutionary process

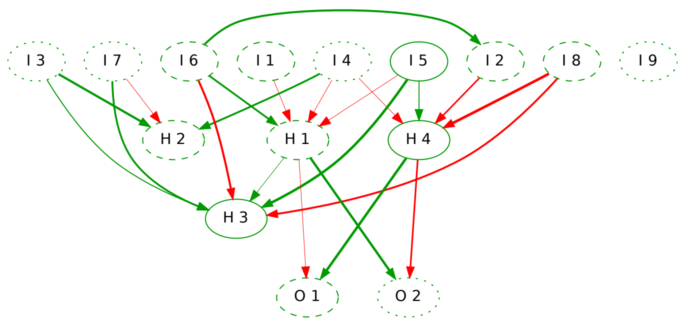


Fig. 11: A neural network diagram. Circles represent neurons, arrows connections. Green arrows are excitatory connections, red arrows are inhibitory. Arrow width is proportional to connection weight. Dotted node boundaries indicate slow time constants, solid boundaries fast time constants.

to be stopped and restarted from any point in its history, and it allows the proliferation of improvements to be tracked. The program also produces diagrams of the neural networks (using Graphviz) which allow changes to be viewed in a human readable way. This hugely aids analysis of emergent behaviours since the effect (on the network) of a mutation that caused a change (in behaviour) can be seen at a glance. Figure 11 shows an example network diagram.

The supervising machine also stores log files that record the behaviour seen by the fitness monitor. These are stored directly as scalar vector graphics files which allows human readability without any loss of information. Opening the file as an image shows the path taken by the robot and the obstacles at a glance, while opening the file as text shows the point data recorded directly. The files also store the names of the individual and its parent (`GenXIndY`), along with its fitness. This allows plotting of fitness against time and for the tracking of the ancestry of fit individuals. Figure 12 shows an example log file.

H. Further Work

Looking back to III-A its clear that some but not all of the aims of the platform have been met. The two key aims which have not been met yet are compact genotype encoding and continuous robot recharging.

The system is currently only able to run for the battery life of the khepera, about 40 minutes, after which it must be recharged, which takes around 70 minutes. The evolutionary process can be restarted from the point immediately before battery death, but it still takes 110 minutes to evaluate 40 minutes worth of controllers. A solution to the continuous charging problem would speed up evaluation for long experiments by a factor of $110 \div 40 = 2.75$.

Martinoli *et al.*[10] propose a conductive charger for mobile robots consisting of metallic floor strips and a Y shaped array of contacts. The contacts are arranged in such a way that, irrespective of position and rotation, they are always in contact with two strips of opposite polarity ($\rightarrow$). This has been shown to provide a safe and reliable source of power for robots similar to Kheperas, and implementation should be relatively low cost.

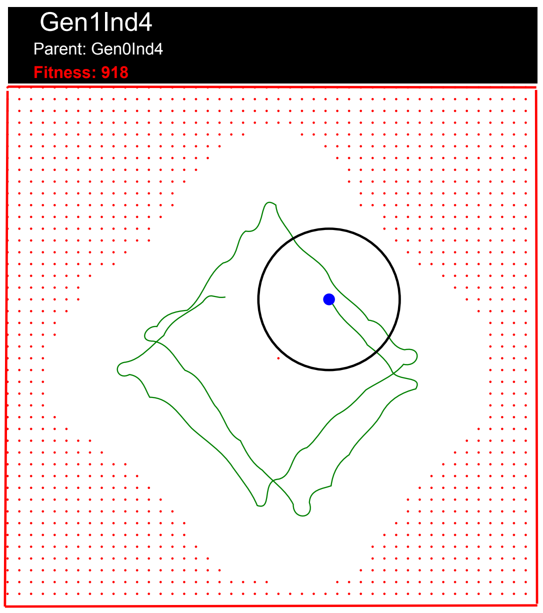


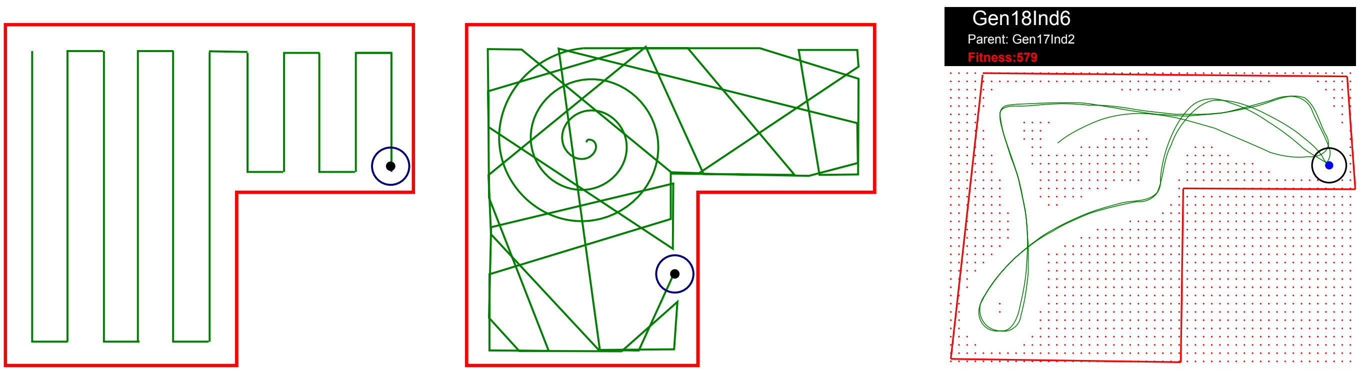
Fig. 12: A log file. The large circle represents the robot (its size is to scale, this was a small maze), the red lines represent boundaries (here the maze is an empty box), the green line represents the path traced by the center of the robot and the red dots represent area that the robot has not covered. Note also the data in the black box, this is useful for tracing improvements back through the robot's ancestry.

Genotype encoding is more of an open area of research. Many methods have been proposed that offer compact yet evolvable genomes, each with its own merit. Nolfi and Floreano[12] propose a method based on tree growth rules wherein neurons are connected if there is a route between them consisting of crossing branches.

I. Experiment

As a demonstration of the platform an experiment into evolving exploratory controllers was conducted. Robotic vacuum cleaners are a real world example of a use for such a controller. The question asked in the title of this report points to some interesting questions: Why does common wisdom say that even mundane repetitive tasks are best solved by humans? Is it possible to design robotic controllers for simple tasks that perform as well as, or even outperform, humans?

Robots such as the Roomba have made a big step forward, but they are far from a serious commercial threat to conventional vacuum cleaners. If it were possible to produce robotic agents that could cover an area as efficiently as a human, at a cost low enough to be competitive with conventional devices, then perhaps they could move out of the realm of gadgetry and into the mainstream. The Roomba's exploration algorithm uses random walk to explore an area as described in algorithm 1. Given enough time, a Roomba is very likely to cover all of the available area in a room of moderate complexity, however it will almost certainly cover certain areas much more thoroughly than others. Figures 13a and 13b show a comparison of likely paths taken by a human and a Roomba.



(a) An example of a near-optimal path. Humans tend to follow paths similar to this when vacuuming.

(b) An example of the path taken by a robot following algorithm 1. Note that although all of the area would be covered eventually, many parts are covered several times, particularly near the walls.

(c) The best path taken by a FFNN controller. Note that the robot falls into a stable repeating cycle - as FFNNs have no memory of past states they are not well suited to exploration tasks.

Fig. 13: A comparison of paths exploring an L shaped area. Figures 13a and 13b are cartoon examples while figure 13c shows experimental data.

Algorithm 1 The Roomba's exploration algorithm

```
 1: repeat
 2:     move in outwards spiral
 3: until collision with wall
 4: repeat
 5:     repeat
 6:         follow the wall
 7:     until random length of time has elapsed
 8:     rotate away from the wall
 9:     repeat
10:         move in straight line
11:     until collision with wall
12: until total cleaning time elapsed
```

The Roomba's algorithm is clearly suboptimal, however the problem it aims to solve is a difficult one. A human is able to create a mental representation of the area to be covered just by inspection, this can be used to plan an optimal route. Robots have been produced that use this method, however the cost of the equipment required to scan the environment is prohibitively large and this situation is unlikely to improve significantly. The challenge for robots like the Roomba is to explore a large area with only limited, short range information about its surroundings. The experiments conducted in this report aim to answer the question of whether or not neuroevolution can produce controllers able to tackle this exploration problem.

IV. Results and Discussion

Two studies were conducted, the first into the performance of FFNNs and the second into CTRNNs. Both studies were conducted in an L shaped area as shown in figure 13. Individuals were scored acording to the amount of unique area they covered using the fitness function described in section III-D.

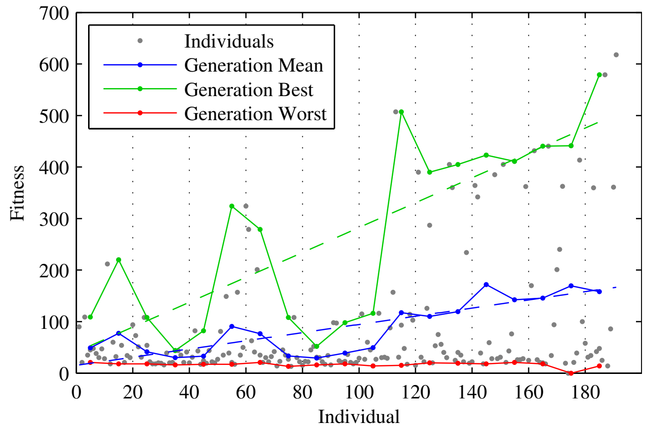
A. *Feed Forward Controllers*

Figure 14 shows the results of the experiment into the performance of feed forward controllers. The first generation of controllers had random weights and biases, and unsurprisingly recieved very low fitness scores (figure 14b).

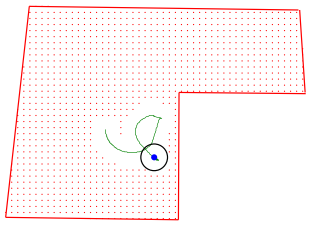
Mutations had produced controllers with the beginnings of an effective strategy by generation 6 (figure 14c). The looping strategy seen in this individual has no obstacle avoidance component yet and so its success is heavily dependant on its starting point and heading. The dip in fitness after generation 6 (evident in figure 14a) is due to this problem - in generation 7 the children of fit individuals from 6 are "unlucky" in their initial placement, score badly and are selected against - combined with the detrimental effect of mutation the strategy is quickly lost. For a strategy to be succesful it must be robust with respect to initial conditions.

The first robust controller occurred in generation 11 (figure 14d), it combines a slightly curved path in open space with wall avoidance to produce a more situation independant exploration strategy. Because the success of this strategy is situation independant it proliferates in future generations, although its fitness is often lower due to the (usually) detrimental effects of mutation. The limitations of feed forward networks start to become apparent in this individual: confronted with the same information the network will always make the same decision (irrespective of whether the situation has occured before). Thus, if the robot finds itself at a position and heading its been in before, it is forced into an infinite loop - and unable to explore new territory.

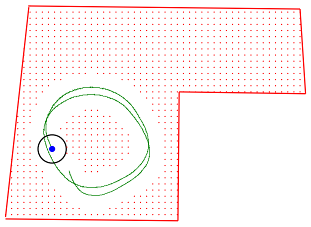
After generation 11 there was a steady improvement in average fitness as the dominant startegy became more common in the population due to selection, but improvements in maximum fitness were not seen until generation 18. There are two reasons for the long gap before an improvement, firstly, as described above, the limits of feed forward behaviour are already being reached, and secondly because mutations usually degrade not improve the controller so improvements tend to be rare. Despite this rarity the ability of selection to favour



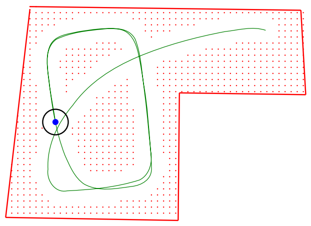
(a) Fitness against individual for evolving FFNN controllers. Each generation has 10 individuals.



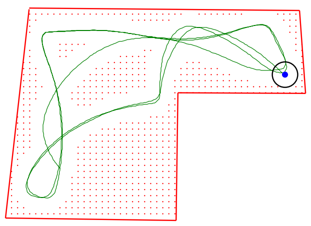
(b) Generation 0 Individual 2



(c) Generation 6 Individual 0



(d) Generation 11 Individual 2

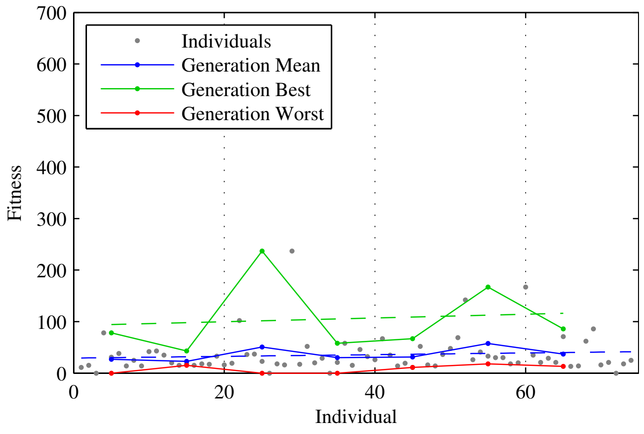


(e) Generation 19 Individual 0

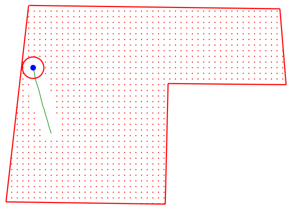
Fig. 14: Results of an evolutionary run using feed forward neural controllers.

good strategies is clear.

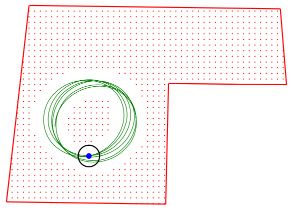
The superiority of the strategy in generation 19 (figure 14e) over generation 11 is evident, however it still suffers from the same limitation - it repeatedly follows the same path in an infinite loop. With further generations fitness would probably improve, and it could well be possible to evolve strategies that attempt to avoid looping. One method that could be used to reduce the deterministic behaviour of the network would be to feed a pseudo-random time series as an extra input to the network alongside the sensors. Inclusion of noise in networks has been shown to increase computational power and behavioural complexity. Even with these improvements it is unlikely that robots without memory or the ability to learn from past experience will be able to cover complex environments effectively. Recurrent neural networks such as CTRNNs have been shown to be able to evolve short term memory and so have the potential to use more interesting strategies.



(a) Fitness against individual for evolving CTRNN controllers illustrating the difficulty of bootstrapping.



(b) Behaviour of a typical initial CTRNN controller. Random control systems in the range used tend not to be sufficiently perturbed by sensory data to avoid obstacles.



(c) The fittest CTRNN controller. None of the controllers in this experiment were able to avoid walls.

Fig. 15: Results of a brief evolutionary run using continuous recurrent neural controllers. The run was aborted due since no individuals were able to react to their surroundings.

B. *Recurrent Controllers*

The potential advantages of recurrent networks are significant, however these advantages come at a cost from the point of view of evolution. In a feed forward network the number of parameters to be tuned is relatively small, there are less than half of the possible connections ($n^2/2 - n$ compared to n^2 in a recurrent network) and time constants are not necessary. Adding these extra parameters to the genome significantly increases the size of the haystack that needs to be searched, although it should also increase the number (and hopefully the quality) of needles. On top of this, recurrent networks exhibit much greater instability than feed forward networks, especially when neurons have self connections. The addition of one recurrent connection to an otherwise fit network can completely alter its behaviour.

The net effect is that a small initial population of controllers with random parameters is quite likely to score uniformly poor fitness, in this situation the evolutionary algorithm is reduced to mere random genetic drift. This is known as the bootstrapping problem, it can be mitigated in a number of ways. The simplest is to start with a very large population of random controllers (to be gradually reduced in future generations for speed), this allows the search to focus only on initially reasonable controllers. Another method is to start the process with controllers which are known to work for a similar problem, since the parameters are likely to be in the correct region, however this can lead to significantly suboptimal solutions.

Figure 15 shows the results of a short experiment attempting to evolve CTRNN controllers, the experiment had to be aborted because of its inability to bootstrap. The problem in this experiment was that, with parameters in the initial range, information from the sensors was too diluted by saturation in neurons and slow time constants to have a useful effect on motor outputs. Attempts to concentrate the evolutionary search in an area of parameter space with higher performance were unsuccesful. Figure 17 in the appendix shows a simulink model of a CTRNN built in an attempt to pre-tune parameters, although it was unsuccesful in this respect it does provide a good visualisation of the dilution problem.

V. Conclusions

The experiments in this study demonstrate the ability of emobodied evolution to evolve feed forward neural controllers capable of basic obstacle avoidance and exploration behaviour. Although it was not possible to evolve CTRNN controllers this is likely to be a practical problem rather than a fundamental issue with the CTRNN system, and that with further work recurrent networks could be produced with similar effectiveness to the Roomba's control system. Since the timescales involved in embodied evolution are so long it is unlikely to be possible to evolve controllers capable of human-level exploration without some use of simulation or massively parallel embodied trials.

It can be concluded that the platform itself is an effective way of designing control systems, although much further work is required. An immediate and significant improvement to the speed of the system could be achieved simply by providing some continuous charging to the robots, and massive improvements in speed would result from the use of simulation or hybrid embodied-simulated experiments. Further down the line investigations into the evolvability of non-trivial genotype-phenotype mapping schema and the use of third generation neural networks (SNNs) could further enhance the speed of evolution and the behaviour of evolved agents.

The study fails to show the superiority of evolutionary methods over conventional approaches to AI, but given its short duration and limited scope this is unsurprising.

Finally, to answer the question asked in the title: Why is my hoover so stupid? - Because its a hoover, what do you expect.

References

[1] T. Back. Selective pressure in evolutionary algorithms: a characterization of selection mechanisms. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pages 57 –62 vol.1, June 1994.

[2] T. Blickle and L. Thiele. A mathematical analysis of tournament selection. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 9–16. Citeseer, 1995.

[3] R.A. Brooks. Artificial life and real robots. In *Proceedings of the First European Conference on Artificial Life*, pages 3–10. Citeseer, 1992.

[4] F. Cupertino, V. Giordano, D. Naso, and L. Delfine. Fuzzy control of a mobile robot. *Robotics Automation Magazine, IEEE*, 13(4):74 –81, 2006.

[5] K.-I. Funahashi and Y. Nakamura. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural Networks*, 6(6):801–806., 1993.

[6] Faustino Gomez, Jurgen Schmidhuber, and Risto Miikkulainen. Efficient non-linear control through neuroevolution. In Johannes Furnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Machine Learning: ECML 2006*, volume 4212 of *Lecture Notes in Computer Science*, pages 654–662. Springer Berlin / Heidelberg, 2006.

[7] J.L. Jones. Robots at the tipping point: the road to iRobot Roomba. *Robotics Automation Magazine, IEEE*, 13(1):76 – 78, 2006.

[8] W. Maass. Networks of spiking neurons: the third generation of neural network models. *Neural Networks*, 10(9):1659–1671, 1997.

[9] Siavash Haroun Mahdavi and Peter J. Bentley. Innately adaptive robotics through embodied evolution. *Autonomous Robotics*, 20:149–163, 2006.

[10] A. Martinoli, E. Franzi, and O. Matthey. Towards a reliable set-up for bio-inspired collective experiments with real robots. In Alicia Casals and Anibal de Almeida, editors, *Experimental Robotics V*, volume 232 of *Lecture Notes in Control and Information Sciences*, pages 595–608. Springer Berlin / Heidelberg, 1998. 10.1007/BFb0112995.

[11] Geoff Nitschke. Co-evolution of cooperation in a pursuit evasion game. In *Proceedings of the 2003 IEEWRSJ Intl. Conferenceon Intelligent Robots and Systems*, 2003.

[12] Stefano Nolfi and Dario Floreano. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines (Intelligent Robotics and Autonomous Agents)*. MIT Press, March 2000.

[13] B. O'Hara. *IEEE 802.11 handbook: a designer's companion*. Institute of Electrical & Electronics Engineers (IEEE), 2004.

[14] Ulisse Perusin, Josef Vybíral, Ricardo González, et al. Gnome icon theme.

[15] R. Pfeifer. On the role of morphology and materials in adaptive behavior. *From animals to animats*, 6:23–32, 2000.

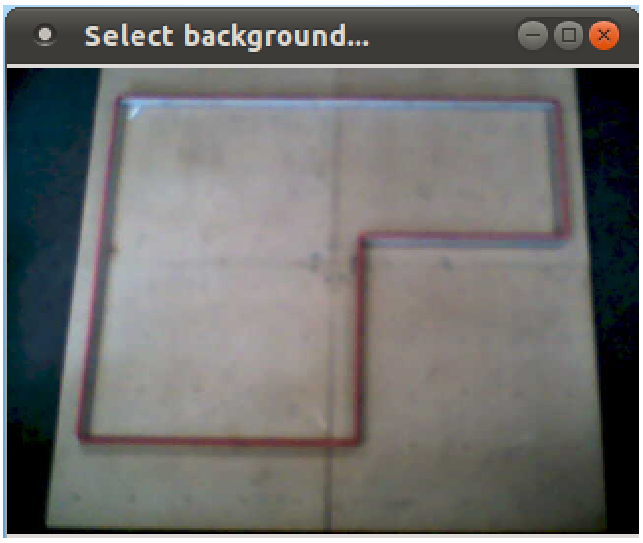
[16] H.A. Simon. *The shape of automation for men and management*. Harper & Row New York, 1965.

Acknowledgements

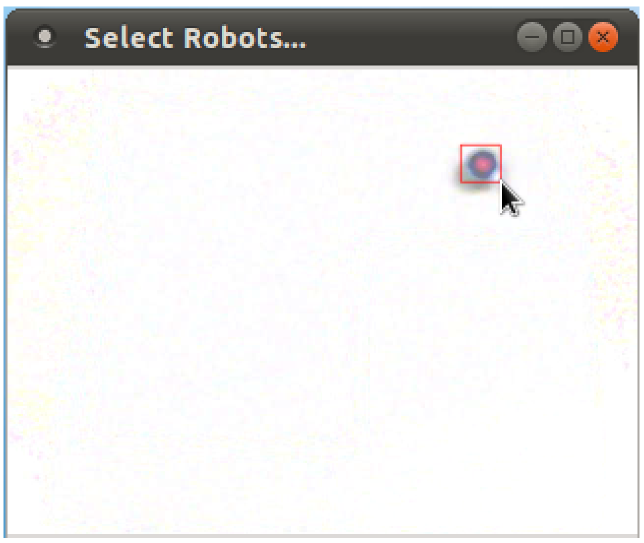
My thanks to Professor V. Vitanov and to Mr. B. Derrick for all their help and support during the project.

All code written for the platform is released under the GNU GPL v3 and can be found at *http://code.google.com/p/kh3dev/*, a copy of the licence is available at *http://www.gnu.org/licenses/gpl.html*

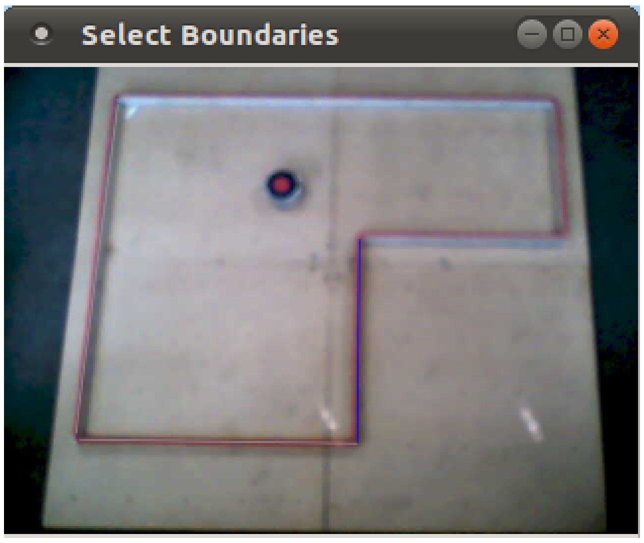
Appendix



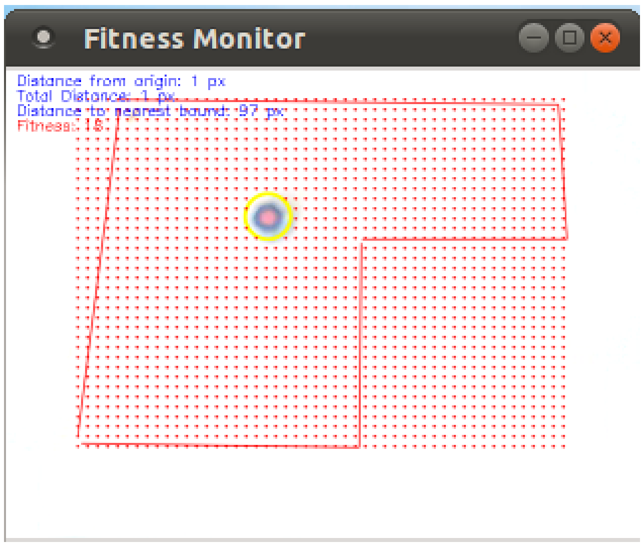
(a) Background selection. The image taken here is subtracted from future frames to make objects stand out for tracking.



(b) Robot selection. Note that the background has been subtracted from this image so that the robot is more obvious.



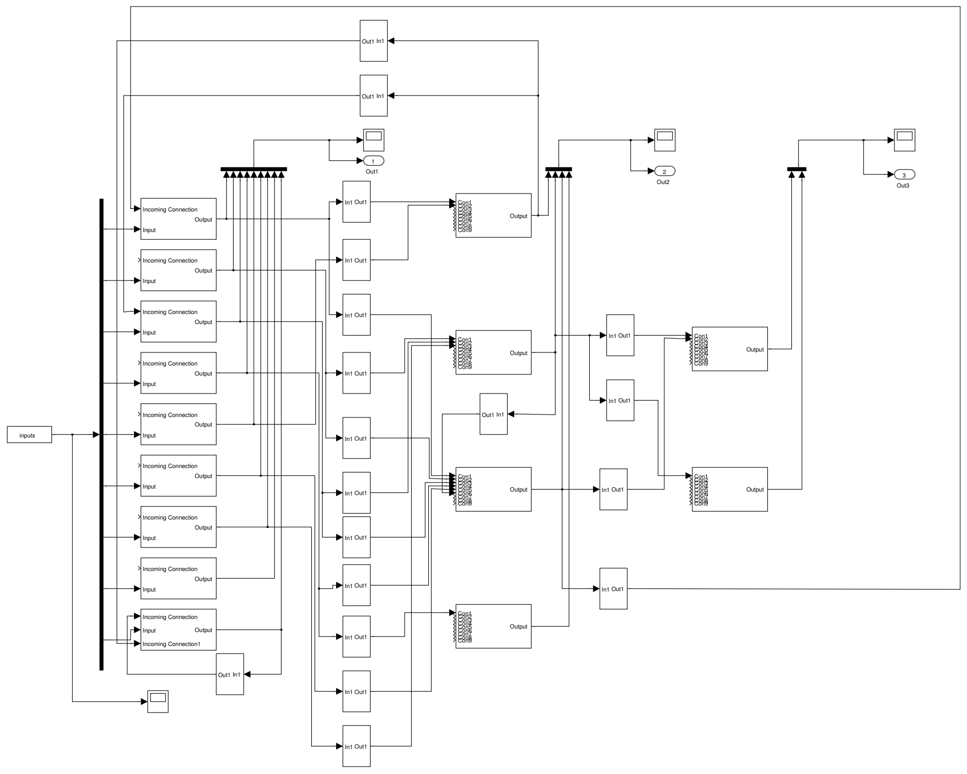
(c) Boundary selection. The user uses the mouse to drag lines delineating objects to be avoided (blue line active, gray lines already saved). The supervisor will intercede if it detects the robot has hit a boundary.



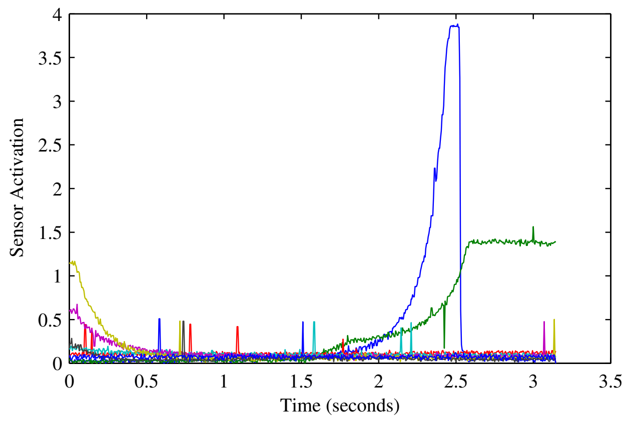
```
Distance from origin: 1 px
Total Distance: 1 px
Distance to nearest bound: 97 px
Fitness: 18
```

(d) The fitness monitor running. The program is capable of autonomously monitoring the robot and avoiding collisions for the battery life of the robot.

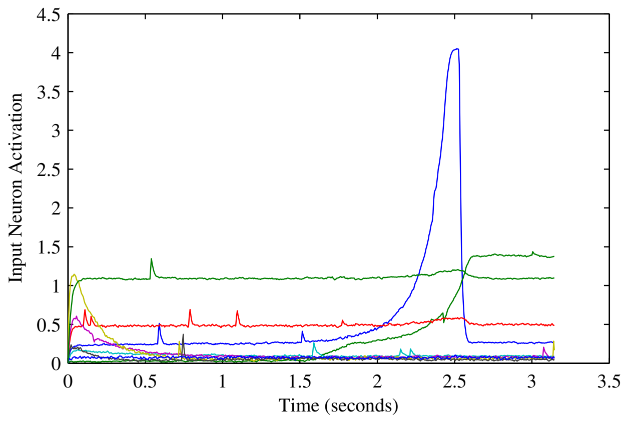
Fig. 16: Screenshots of the supervisor program



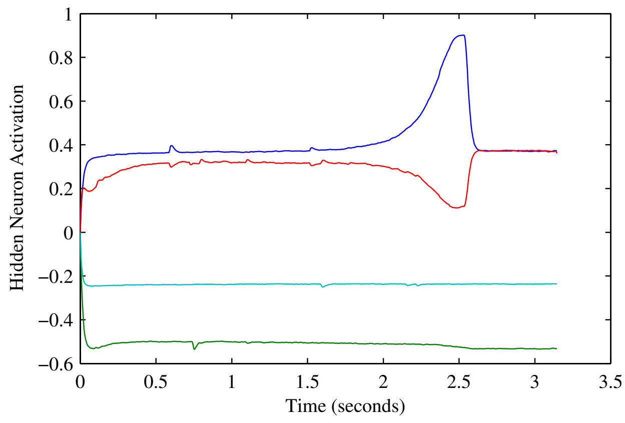
(a) The CTRNN simulink model. Inputs to the system were recorded from the robot moving in the environment.



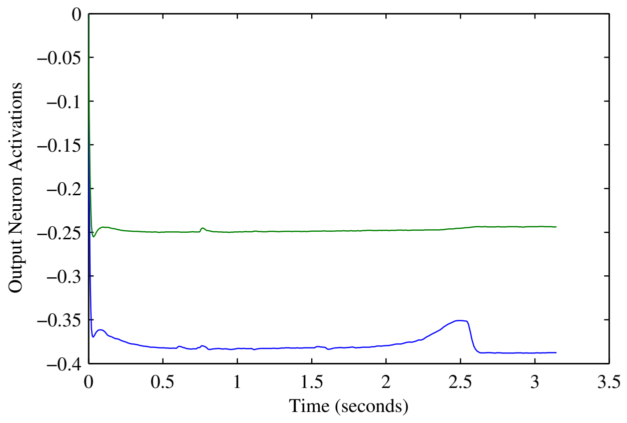
(b) Sensor ouputs before being fed into the neural network.



(c) Activations of input neurons. Note that those with recurrent connections tend to sustain activation even with no sensory input.



(d) Activations of hidden neurons. Due to saturation, damping and recurrence sensory information has become diluted.



(e) Activations of output neurons. Sensory information has been diluted to the point of irrelevance.

Fig. 17: Results of a MATLAB Simulink simulation of a CTRNN.