Evolutionary Blimp & i

Matthijs van Leeuwen

22nd August 2002

E-mail: mleeuwen@cs.uu.nl

Internship report

May 11th, 2002 - July 5th, 2002
Professor: Dario Floreano
Advisor: Jean-Christophe Zufferey
Internal advisor: Marco Wiering

Autonomous Systems Laboratory (LSA2)
Institute of Systems Engineering (I2S)
Faculty of Eng. Science and Technology (STI)
Swiss Federal Institute of Technology in Lausanne (EPFL)

Institute of Information and Computing Sciences
Utrecht University
P.O. Box 80.089
3508 TB Utrecht
The Netherlands

Abstract

The first part describes evolutionary experiments with an indoor airship in which spiking circuits capable of performing vision-based navigational behaviour are evolved within twenty generations. The strategies of the best evolved individuals exploit the characteristics of the robot and (textured) environment to perform well. Although only visual information is used as input, the resulting neural controllers are able to react when the flying robot collides with a wall.

The second part presents the objectives and design choices of an application, named *i*, designed and developed for evolutionary robotics experiments with neural controllers. The design of the application facilitates easy implementation and usage of different types of robots, sensor morphologies and neural networks. Implementation of *i* is currently well under way and a few preliminary experiments have already been done.

Contents

1 Introduction **2**

2 Evolutionary Blimp **3**
2.1 Bio-inspired Robotics . . . 3
2.2 Cheap Vision for Flying Robots . . . 3
2.3 Artificial Evolution . . . 4
2.4 Spiking Neural Circuits . . . 5
2.5 Indoor Airship . . . 7
2.6 Experimental Setup . . . 9
2.7 Results . . . 10
2.8 Discussion . . . 12

3 i **15**
3.1 Objectives . . . 15
3.2 Design Choices . . . 16
3.3 Implementation Choices . . . 16
3.4 Status . . . 17
3.5 Further Development . . . 17

4 Conclusions **19**

Chapter 1

Introduction

Evolutionary robotics [14] has become a very popular field of research in the last few years. A bottom-up approach and techniques inspired on biology (used to develop real robots that show intelligent and life-like behaviours) make that the discipline fits very well within both 'Nouvelle A.I.' and 'Artificial Life' [1, 13], two fast growing fields combining numerous sciences, such as micro-engineering, computer sciences, biology, neurosciences and psychology.

Probably the best-known work on the subject is the pioneering book written by Nolfi and Floreano [14], in which the authors explain and motivate the concepts of (their view of) evolutionary robotics and present many promising results. Dario Floreano is professor and head of the Autonomous Systems Lab 2 (LSA2) at EPFL in Lausanne and his lab focuses primarily on research in the field of evolutionary robotics. In this report, I will present the results of my two months work at this lab.

My primary goal during my stay at EPFL was to achieve good results with experiments with an indoor airship (see also Zufferey et al. [18]). The blimp and its environment were almost ready for doing experiments, mainly the software needed to be modified before experiments could be done. It took some time to get fully ready and resolve some last practical issues, but from that moment on, the blimp has been flying around almost permanently during my stay.

A large benefit of evolutionary experiments is that, once running, few human interventions are needed. In the case of the blimp, only battery changing and adjusting the weight are necessary on a quite regular basis. The rest of the time, though, can be spend on other matters. Since the original evolutionary software, written by Prof. Dario Floreano, wasn't designed to be easily used with multiple robots and protocols, I decided to re-design and re-develop this application.

After preparing the blimp and the (old) evolutionary software for doing experiments, I started together with Brice Legrand (another visiting student) and Jean-Christophe Zufferey to develop a new application based on the concepts of the old one. Meanwhile, the blimp experiments continued non-stop and results had to be evaluated now and then.

The overall result is that my research project consisted mainly of two rather different tasks: on one hand doing experiments with the blimp, on the other hand developing a new application for evolutionary robotics, named *i*. In the next chapter, I will describe the experimental setup, the experiments I did and their results. In the third chapter, I will discuss the objectives of the new application and the current status. After that, I will end with some conclusions in chapter four.

Matthijs van Leeuwen

Chapter 2

Evolutionary Blimp

This chapter deals with the evolutionary experiments that were performed with an indoor airship, with the objective to evolve vision-based navigational behaviour using spiking circuits as control systems. The first two sections will introduce the context of bio-inspired robotics and cheap vision in which these experiments have been carried out. Sections 2.3 and 2.4 explain, respectively, the evolutionary method and neural network model, after which section 2.5 will deal with the robotic testbed and section 2.6 will put all this together. Section 2.7 describes the achieved results and the last section gives place to discussion.

2.1 Bio-inspired Robotics

Tasks like moving and navigating in an environment are very trivial for us human beings. There are exceptions, but generally seen, it's not hard for us to go from one place to another and, for example, to avoid obstacles while moving. But the same counts for almost every single species on earth: even organisms that we see as 'very stupid' are often capable of finding their way around. But then, all living organisms on earth are the result of millions of years of evolution.

Ever since the first robots were invented, humanity has been intrigued by robots and the possible consequences they may have for the future. Especially autonomy is considered something very special, but when we look at the robots that have been developed in the past decennia we see that we are yet far away from truly autonomous robots. It's not obvious how life-like autonomous robots could be hand-designed. It is for this reason that more and more researchers use biology as inspiration, observing carefully and applying adapted similar concepts to robotics.

Evolutionary robotics is based on such an adapted concept: artificial evolution. Genetic algorithms [8] are already used for searching and optimising for quite a long time, but using it to evolve controllers for real robots is relatively new. It's certain that artificial evolution is inspired on natural evolution, but Schwardy even argued that it is a correct phenomenological reduction [16]. Section 2.3 explains more on artificial evolution.

A neural network is a model of the nervous system and therefore also inspired on biology. Neural networks have also been used for a long time, but mainly for pattern matching and other static tasks. Using it to control a real robot in real-time is something of the last decade. Although these models were rather simplistic at first, they are becoming more sophisticated now and as a result of this more capable of accomplishing difficult tasks. Section 2.4 will explain the model used in the experiments.

2.2 Cheap Vision for Flying Robots

Most existing robots are terrestial, which is not so surprising, since the (often wheeled) terrestial robots already offer many possibilities for research: many different sensors and actuators can be

used, it is fairly easy to let these robots show interesting behaviour and there are many real-life applications. When developing such a robot, the only restriction is the available technology: suitable sensors are often hard to find.

Developing aerial robots brings many more restrictions, of which the most important is the weight constraint. A robot that is too heavy won't be able to fly, it's as simple as that. A consequence is that many conventional sensors, such as laser and ultrasonic distance sensors, cannot be used. Also, since a flying robot has to be wireless, the power supply has to be onboard and this can't weigh too much either.

Using vision for sensory perception is a very interesting option, since vision sensors can be very lightweight and don't need much power. However, the mainstream approach to machine vision is not feasible, since segmentation, feature extraction and pattern recognition demand more computing power than a small flying robot is able to provide with its very limited load capacity.

In the previous section, it was mentioned that it's often useful to use concepts we know from biology. Since simple but effective visuomotor mechanisms have evolved in many living organisms, particularly in insects, we can use these as inspiration and use analog mechanisms for our robots. Pioneering work in this direction was done by Franceschini et al. [6], who developed a wheeled robot with a vision system inspired upon the visual system of the fly. A 10 kg wheeled robot featured an artificial compound eye with 100 discrete photoreceptors and was able to navigate at about 50 cm/s towards a light source, avoiding randomly arranged obstacles. Contrary to later similar realisations, this robot didn't require many computing power, since computation was done onboard by analog electronics.

An example of a bio-inspired flying robot is Melissa [10, 11], an indoor airship able to show goal-directed navigation behaviour using Elementary Motion Detectors (EMDs). The experiments were rather simple though, since a fixed route was used. Also, as was the case with the robot of Franceschini et al., the robot controller was hand-designed.

Huber evolved vision-based navigation using simulated 2D agents [3]. The agents were equipped with four photoreceptors making up two EMDs, symmetrically places on each side of the agent. Both the parameters of the EMDs and the field of view (FOV) of the receptors were evolved and the best individuals were able to navigate in a simulated corridor with textured walls and obstacles. The simulation was rather simple though, especially because inertial forces were not taken into consideration.

Floreano and Mattiussi evolved vision-based navigation for the Khepera [4], a wheeled robot, using a spiking neural network as controller. A Khepera robot with a linear camera got the task to navigate in a rectangular arena with textured walls (figure 2.1, left). The best controllers were capable of moving forward and avoiding walls very reliably. However, the complexity of the dynamics of this terrestial robot is much simpler than that of flying robots.

The flying robot in this report has a task similar to the Khepera: it is asked to navigate in a textured room using only visual information. Artificial evolution is used to evolve the architecture of a spiking neural circuit, which connects the visual input to the motors of the small indoor airship (figure 2.1, right).

2.3 Artificial Evolution

Genetic algorithms are widely used for solving optimise and search problems, particularly when the search space is extremely large. When applied to search problems, a large set of possible solutions is evaluated and generation of the next set of possible solutions is based on this evaluation. Artificial evolution, though, resembles natural evolution even more. At the start of an experiment, a random initialised population is created, consisting of a number of individuals. In evolutionary robotics, each individual is a robot controller (in most experiments) and the performance of each individual can be measured by trying the controller on a robot.

An individual consists of a genotype and a phenotype. The genotype is the genome, in our case a simple binary string of fixed length. When an individual's performance is tested, the phenotype

Figure 2.1: Evolutionary vision-based robots. *Left:* The Khepera equipped with a linear camera (16 pixels, 36° FOV) was positioned in an arena with randomly sized black and white stripes. Random size was used to prevent development of trivial solutions whereby the controller would use the size of the stripes to measure distance from walls and self-motion. *Right:* The blimp-like flying robot, provided with a similar linear camera (16 pixels, 150° FOV), is closed in a 5x5x3 m room with randomly sized black and white stripes on the walls.

corresponding to the genotype is created: a robot controller. In the experiments we did, this controller was a spiking neural network, see next section.

After the performance of each individual of a population has been tested on the robot, reproduction of the population can be done based on the measured fitness values. As selection method we used truncation selection: the best individuals were selected for reproduction, the rest was truncated. Elitism was also used: the best individual was always retained (without modifying the genome). Another important characteristic of evolution are the genetic operators: the crossover operator we used cuts two bitstrings in two parts, at the same positions, and glues one part of one genome to the second part of the other genome and vice versa; the mutation operator toggles every single bit of the genome with a certain chance.

The evolution parameters used for the experiments are similar to those of the Khepera experiment already mentioned. A population has 60 individuals, of which the 15 best individuals are each copied 4 times to the new population. Each individual is tested on the robot for 2 epochs of 40 seconds each and the crossover and mutations rates are 0.1 and 0.05 respectively.

2.4 Spiking Neural Circuits

Conventional connectionist neural networks [2] are very simplistic models of the nervous systems of living organisms. In standard networks, membrane potentials are propagated using (real-valued) activation values and decay due to axon/dendrite length and synapse strength are represented by a single weight. Moreover, most networks have no 'memory' (states are not preserved between time steps) and are therefore not suitable for applications in which temporal changes play an important role.

The spiking neural circuits used in the experiments described in this report use a more sophisticated model, the spike response model [7], that resembles better the nervous systems of living beings. Membrane potentials are not simply propagated to other neurons using only a weight, but a neuron emits a spike to all outgoing connections when its membrane potential exceeds a certain threshold. Neurons are considered to be interconnected by synapses with strength 1.

A spiking circuit consists of sensory receptors and a set of neurons, of which a subset is used to control the actuators of the robot (figure 2.4, left). Sensory receptors can be connected to any or all of the neurons, the same counts for the neurons (which means that they may be fully interconnected and self-connections are also possible). The sensory system is based on a

stochastic mechanism: depending on sensory perception, each receptor has a certain chance of emitting a spike once every 100 ms. The average firing rate during a longer time is therefore directly proportional to the average perception. When a receptor or neuron emits a spike, the spike is duplicated for each separate outgoing connection. A neuron can be either excitatory or inhibitory: a spike coming from an excitatory neuron adds to the membrane potential, vice versa for inhibitory neurons.

Each neuron is updated every millisecond: contributions of all incoming spikes of the last 20 ms are added to the membrane potential and when the potential exceeds a threshold θ, a spike is emitted. After emitting a spike, the potential is set to a very low value in order to prevent the neuron from spiking the next time step and the potential subsequently gradually increases back to its resting potential; this is to model the refractory period. As a result of this, a neuron can emit a spike every second millisecond and the maximum spike rate of a neuron is 500 Hz.

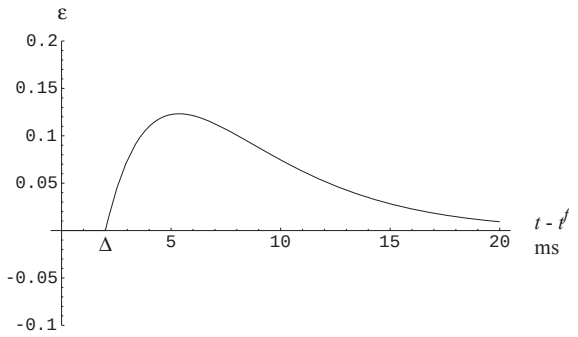

Figure 2.2: The function describing the contribution ε of a spike from a presynaptic neuron, emitted at time t^f. The contribution of the spike begins after a delay Δ (2 ms) and decreases after a strong initial increase. The synapse time constant τ_s is set to 10 ms, the membrane time constant τ_m to 4 ms.

Spikes in natural neurons have two more important characteristics that are taken into consideration: firstly, there is a certain delay Δ between the emission of a spike and its arrival at another neuron (because of the length of axons and dendrites and the time needed for neurotransmitters to cross a synapse), secondly the strength of the spike is not constant. Although many different types of spikes can be found in biology [9], the contribution ε of an incoming spike to the membrane potential is modelled with an exponential function of the difference s between the current timestep t and the time of spike emission t^f (see figure 2.2):

$$\begin{aligned} \varepsilon(s) = \quad & \exp[-(s-\Delta)/\tau_m](1-\exp[-(s-\Delta)/\tau_{s]}) & s \geq \Delta \\ & \varepsilon(s) = 0 & s < \Delta \end{aligned}$$

The function has several parameters that influence the precise form of the function: Δ is the delay, τ_m is the synapse time constant and τ_s is the membrane time constant. The refractory period is also modelled as a function of s, the time between the last spike and the current time; the speed of recovery depends on the membrane time constant τ_s (see figure 2.3):

$$\eta(s) = -\exp[-s/\tau_m]$$

Now that we have functions to calculate contributions of individuals spikes to the potential and the refractory period of a neuron, we can define a function to calculate the membrane potential v of a neuron i at any given timestep t (when the calculated membrane potential exceeds the threshold θ, a spike is emitted):

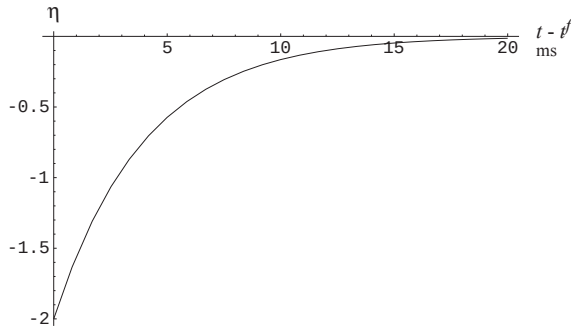

Figure 2.3: The function describing the refractory period of the neuron after emission of a a spike.

$$v_i(t) = \sum w_j^t \sum \varepsilon_j(s_j) + \sum \eta_i(s_i)$$

A full sensorimotor cycle consists of 100 (discrete) time steps of 1 ms each. At the first time step, each sensory receptor may or may not emit a spike. The neuron membrane potentials are updated and may (or may not) emit a spike every millisecond, 100 times a cycle. At the end of the cycle, average fire rates (average of the last 20 ms of the 100 ms cycle) of the selected output neurons are used to control actuators. Thus, sensory receptors get the opportunity to spike ten times a second and actuator values are also set ten times a second.

In the experiments, the architecture of the spiking circuit is genetically encoded and evolved (figure 2.4, right), as described in the previous section. The genome, a binary string, is composed of a series of blocks, each block representing one of the neurons of the circuit. The first bit of a block determines whether the corresponding neuron is excitatory or inhibitory, the remaining $n + s$ bits encode the presence (yes/no) of incoming connections from all n neurons and s sensory receptors. All other parameters are set to the given default values; no tuning has been done on these.

2.5 Indoor Airship

The practical issues of evolving wheeled robots are not really a problem nowadays, but evolving aerial robots brings new challenges to be conquered. The major issues of developing a control system for an airship, with respect to a wheeled robot, are (1) the extension to three dimensions, (2) the more complex dynamics, (3) the impossibility to communicate with a computer through cables, and (4) the difficulty of defining and measuring performance.

While a control system for a Khepera only needs to steer the Khepera in two dimensions, a control system for an indoor airship has to be able to gain control over all three dimensions[1]. Moreover, a Khepera is controlled in speed and has three degrees of freedom and two degrees of motion, but an airship is controlled in thrust and has 6 degrees of freedom and 4 degrees of motion (if we assume that pitch and roll are passively stabilised). The third, uncontrolled degree of freedom for a Khepera is always zero, since a Khepera can't move sidewards. This is not the case for the airship though: sideward movement is possible, but not controlled directly. Also, inertial and aerodynamic forces play a major role.

Communication between the airship and the desktop computer is done by a wireless Bluetooth connection with a range of more than 15 m. A connection can be established within seconds and subsequent data transmission is reasonable fast and reliable. Since the airship has to be wireless,

[1] However, the experiments described in this report were limited to two dimensions by the use of a hand-designed altitude controller.

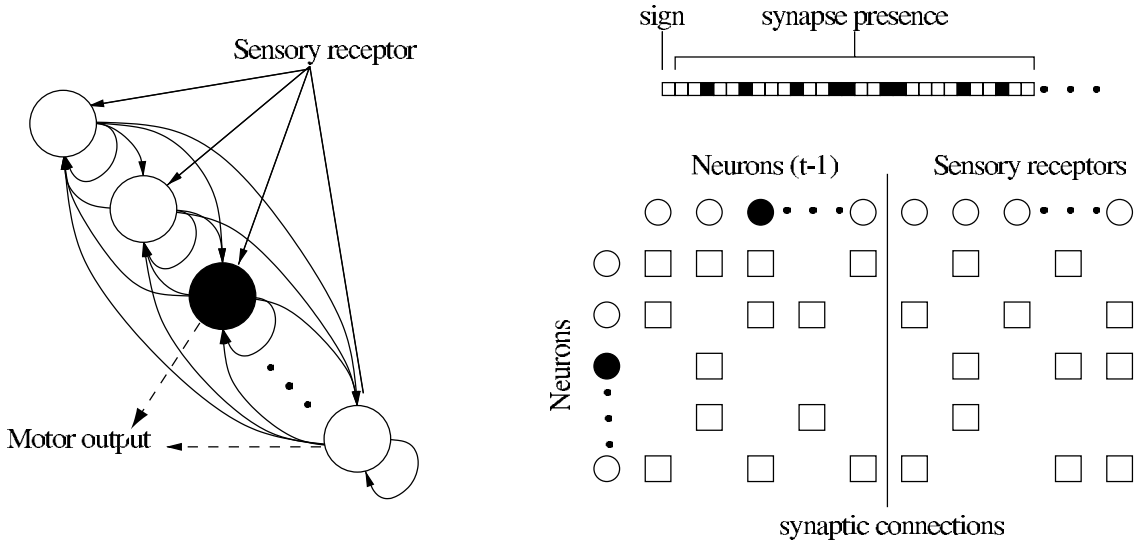

Figure 2.4: Network architecture (only a few neurons and connections are shown) and genetic representation of one neuron. *Left:* A representation showing the network architecture. *Right:* The same network unfolded in time (neurons as circles, synaptic connections as squares). The neurons on the column receive spikes from connected neurons and receptors on the top row. The first part of the row includes the same neurons at the previous time step to show the connections among neurons. Sensory neurons do not have interconnections. The signs of the neurons (white = excitatory, black = inhibitory) and their connectivity pattern are encoded in the genetic string and evolved.

power supply has to be onboard; a lightweight 700 mAh Lithium-Ion battery is enough for three hours of normal operation, during evolution.

Measuring performance of a flying robot is another difficult issue, since performance is often a function of the (forward) motion of a robot. For wheeled robots, it's often possible to read the actual wheel speeds using optical encoders, but it's not obvious how to measure speed of an aerial robot. Computed motor speeds don't tell much, since the robot is thrust controlled. In order to be able to use relative airspeed for performance measurement, an anemometer has been mounted on the airship (see figure 2.5).

Apart from the challenges described above, there are some other issues specific for articial evolution that deserve attention. Evolutionary runs typically last 2 to 3 days and it is therefore required that the robot is suitable for long periods of operation with as few human interventions as possible. Also, it has to withstand shocks and a way has to be found to start each epoch with a more or less random initial situation.

All these requirements led to the development of the airship shown in figure 2.5. All onboard electronics are connected to a Microchip PIC^{TM} microcontroller, an easily programmable controller for which code can be written in C. This microcontroller features a serial port that is used to establish a bidirectionial connection with the onboard Bluetooth radio module. The microcontroller waits for an incoming Bluetooth connection from the desktop computer and starts processing remote commands (like set motor speeds and read sensor data) when a connection is accepted. For purpose of analysis and easy control, the evolutionary algorithm and spiking circuits run on the desktop computer, which exchanges commands and data with the blimp. An adapted form of the evolutionary algorithm and spiking circuits could also be run on the onboard microcontroller [5], but this would limit the amount of data analysis possible and would make it less obvious for the human supervisor when batteries should be changed.

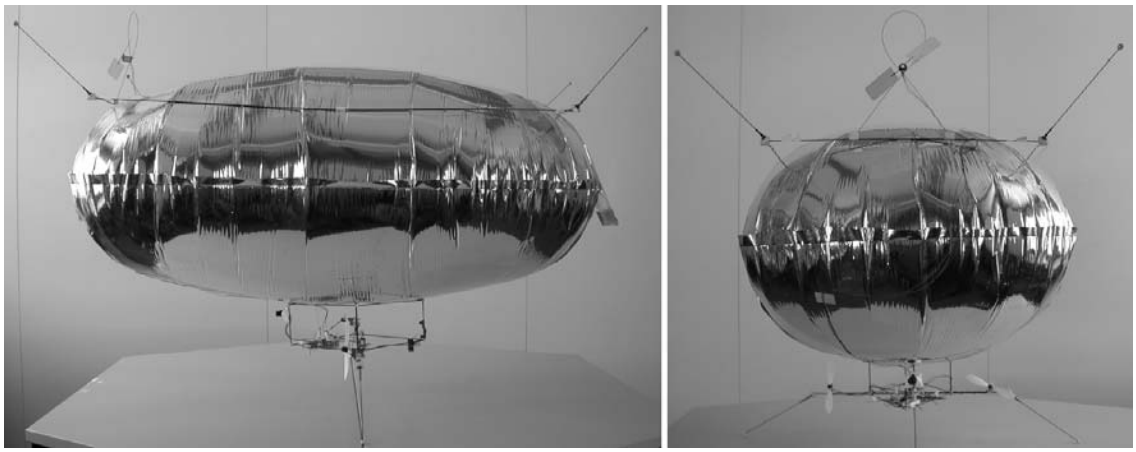

Figure 2.5: The blimp features an elliptic envelope (100x60x45 cm) filled with helium for a lift capacity of approximately 250 g. Attached above and below the envelope are two frames made of carbon fiber rods that support six bumpers (4 on top and 2 below) for collision detection and protection of other hardware parts (especially the propellers). It is equipped with three engines (miniature DC motor, gear and propeller): two for horizontal movement (forward, backward and rotation around jaw axis) and one for vertical movement. In order to qualitively measure the relative forward airspeed, an anemometer (free rotating balsa-wood propeller) has been mounted on top of the envelope. A distance sensor has been mounted below the gondola and directed towards the floor for altitude control. A linear camera is mounted horizontally in front of the gondola, the field of view directed straight ahead. Data transmission is handled by a Bluetooth wireless connection, power supply by an onboard battery lasting approximately three hours.

2.6 Experimental Setup

Now that we have defined a way of developing a control system (artificial evolution), a type of control system (spiking circuits) and a testbed (an indoor airship), it's about time to put these separate parts together to form a whole.

During evolution, a hand-designed altitude controller is used to maintain a reasonable altitude (approximately between 50 and 100 cm above ground level). This altitude controller is based on the dynamics of the distance measured by the infrared sensor and the rotation speed of the vertical propeller: a decreasing distance makes the propeller accelerate (thrusting upward), an increasing distance makes the propeller decelerate. The airship has to be balanced in such a way that it goes down very slowly when the vertical propeller is off.

As shown in figure 2.6, the 16 vision values (8-bit greylevels) measured by the forward pointing linear camera are filtered using a one-dimensional Laplace filter spanning three adjacent values in order to detect contrast. After this, absolute values are taken and scaled in the range [0,1]. At the start of a 100 ms cycle, the resulting values are used as input values for the 16 sensory receptors of the spiking circuit; each receptor has a chance of emitting a spike equal to the input value at this time step.

The total spiking circuit consists of 16 photoreceptors and 10 neurons, of which 4 neurons are dedicated as output neurons. At the end of each cycle, the motor speeds of the two horizontal propellers are set according to the output neurons in a push-pull mode, two neurons for each motor. The firing rate (measured during a 20 ms window at the end of the cycle) of one of the two neurons is substracted from the firing rate of the other, resulting in a value in the range [-1,1].

I already described the genome encoding of the circuit architecture in section 2.4. Each population of 60 individuals encodes 60 spiking circuits and each of these spiking circuits is subsequently tested on the real blimp. Each individual is tested for two epochs of 40 seconds, during which fitness is measured. Since one sensory-motor cycle takes 100 ms, a full test consists of $T = 800$

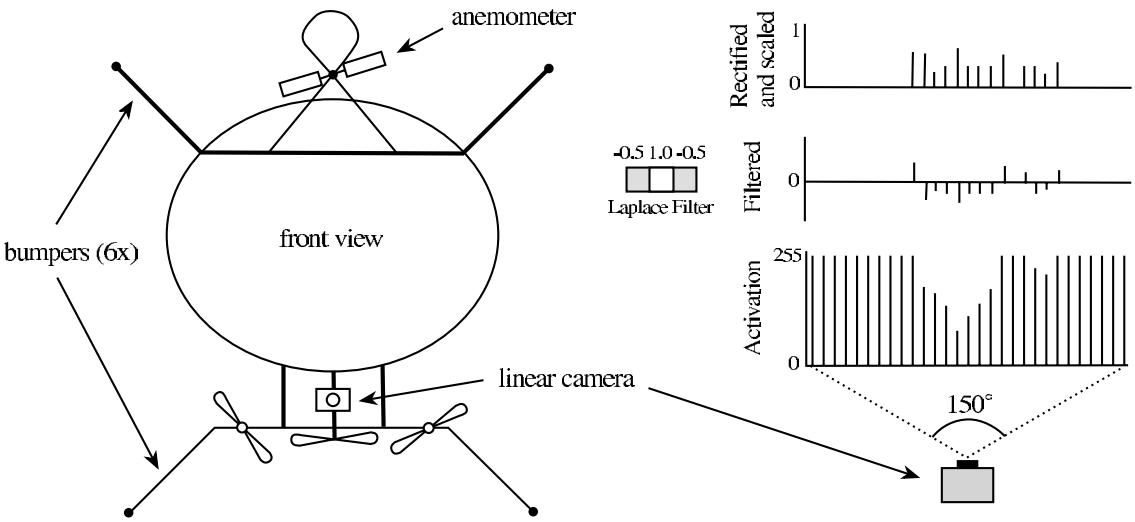

Figure 2.6: *Left:* The blimp and its main components: the anemometer on top of the envelope, the linear camera pointing forward with 150° FOV giving an horizontal image of the vertical stripes, the bumpers and propellers. *Right:* Contrast detection is performed by selecting 16 equally spaced photoreceptors and filtering them using a Laplace filter spanning three adjacent photoreceptors. Absolute values are taken and scaled in the range [0,1] to serve as probability of emitting a spike.

time steps. The fitness is a function of the approximate forward motion $\hat{v}$ as measured by the anemometer every time step t: $\Phi = \frac{1}{T}\sum_t^T \hat{v}^t$.

After each epoch, a preprogrammed semi-random movement is executed to create a more or less random intial position for the next epoch. First, both motors thrust backward to free the robot from any walls or corners. After that, both motors thrust forward, but one (randomly chosen) motors thrusts with double the power of the other one to rotate away and stop the backward movement. Finally, there is a short period with both motors turned off to let the blimp lose (almost) all its motion.

2.7 Results

After some preliminary experiments to test the setup and resolve some last practical issues, we started experimenting with a form of killing to quicken evolution. Whenever an individual collided with the wall longer than one second, the epoch was immediately ended and the next epoch was started. This strategy can be motivated with the assumption that individuals that hit the walls are not good enough: good navigation should include obstacle avoidance.

The results of the two runs are shown in figure 2.7. The first run seemed very promising, since both average and maximum fitness values were still increasing after 20 generations. However, the results of the second run are not so good: the maximum changes a lot over time.

Since killing seemed to introduce too much randomness, all subsequent runs were done without killing: all individuals were tested for two full epochs, independent of collisions. The results of these runs are much more stable than the killing runs; the average of five performed runs is shown in figure 2.8 (top left).

The best individuals of the five runs are able to navigate around the room using only the processed visual information as input for the spiking circuits. Within 20 generations (2-3 days), well-adapted strategies are evolved that use the characteristics of both the environment and the robot to gain forward speed and thus achieve a good fitness value. Typical best individuals actively use the walls to stabilise the course, which is not surprising if we keep in mind that the fitness function only includes the forward speed measured with the anemometer. Obstacle avoidance is

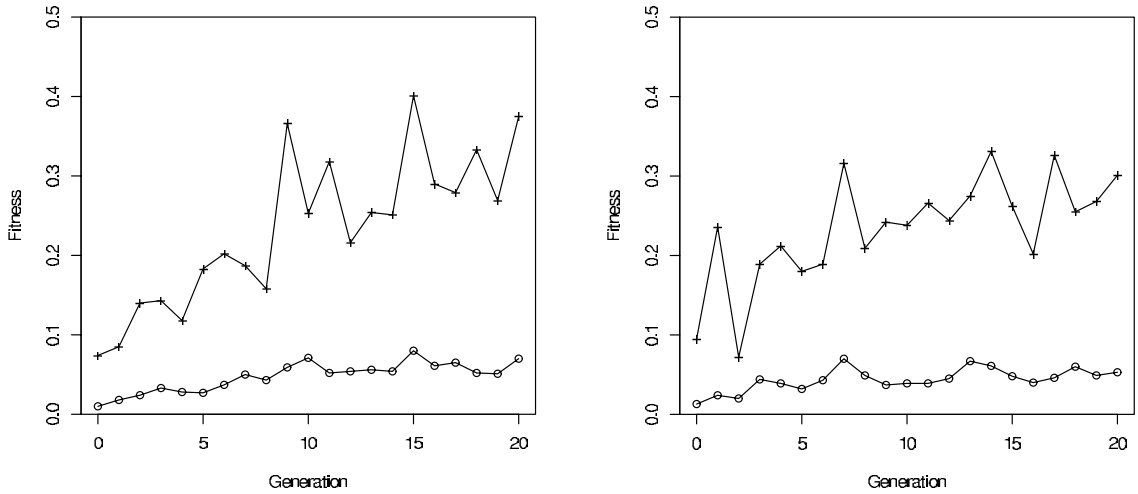

Figure 2.7: Two evolutionary runs with killing (circle = average individual; cross = best individual). An individual is killed when more than 10 collisions occur; the collision counter is updated once every sensory-motor cycle, 10 times a second, based on the bumper information. The fitness is calculated exactly the same as for individuals that are not killed; the sum of measured speeds is divided by the maximum number of time steps.

not required by this fitness function, neither explicit nor implicit.

The behaviour of a typical best individual is shown in figure 2.8 (top right). The individual starts with a rotational velocity and keeps rotating (a) until the course is stabilised by a collision with a wall (b). (As soon as the blimp rotates, it's easy to keep it rotating: thrusting both motors forward is enough. It is not so easy, though, to stabilise the course of the rotating blimp without using the walls.) The blimp then moves forward until it hits another wall (c), at which point the motors are turned off until the robot has bounced back and is free from the wall again. The motors thrust forward, once more stabilises its course using the wall and moves forward again. This strategy is repeated over and over again.

The performance of one of the best spiking circuits can also be seen in figure 2.8 (bottom). It clearly reacts when it bumps into a wall; there is a strong correlation between collisions and the motor speeds. Whenever the image on the retina doesn't move, the motor speeds are gradually decreased and kept on zero until the percepted image starts moving again. This situation occurs when the blimp is right in front of a wall, right after a collision. Note that a collision can be a collision with any one of the six bumpers on the blimp and therefore doesn't necessarily imply that the blimp is frontal against the wall.

The three bottom graphs also show that there is a correlation between the left and right motor speeds, the average of the two is almost equal to the two separate speeds. This indicates that individuals that turn off both motors after a collision have an evolutionary advantage compared to those that don't, since evolution selected those individuals that have spiking circuit architectures turning off both motors.

To get an idea of how good the performance of the best evolved spiking circuits is, seven test persons were asked to control the blimp using a joystick (figure 2.9, left). They were told how the performance would be measured in advance (exactly the same as with the robot experiments) and were shown only the same visual information as the spiking circuits during the tests (but without applying the Laplace filter). As with the evolutionary experiments, altitude was controlled to limit the degrees of motion to two. After six epochs of training, where no feedback was given apart from the visual information, performance was measured during two epochs. The fitness of the best evolved spiking circuit was 0.413, but the fitness of the best human was only 0.257.

It is clear that evolved spiking circuits actively use the walls to perform better, an interesting question is whether people use the same strategy. Figure 2.9 (right) shows the correlation between

the achieved fitness values and the number of collisions belonging to the corresponding epoch. If collisions with the wall in this experiment would stabilise the course of the blimp, we would expect higher fitness values to correspond with higher collision numbers. This seems not to be the case though: whatever strategies the test persons may have used, they didn't use the walls to achieve a higher fitness value.

2.8 Discussion

Although the results of the first run with killing were good, the results of the second run were disappointing and it looks like killing introduces too much randomness in evolution. Initialisation of an epoch is always very important to make sure that the robot starts in a (more or less) random situation, but becomes even more important when individuals hitting the walls are killed. If the starting position is near the wall, the chance of colliding (and therefore getting killed) is rather large even when the individual would have performed well if it wouldn't have been killed.

Evolution without killing individuals is better able to select those individuals that are capable of navigating around the room, but this has other disadvantages: individuals are no longer required to avoid obstacles, since there is no penalty whatsoever on bumping against the wall. As a result of this, the best individuals actively use the walls to stabilise the course and gain forward motion.

It may be possible to make individuals avoid walls though, by changing the fitness function. For example, the number of collisions during an epoch could be counted and used as penalty, although the fitness function would become more explicit this way and a scaling factor would have to be hand-designed.

It is very difficult to start each epoch with a randomly chosen position and reality learns us that the current random movement initialising each epoch is able to free the airship from the walls on most occasions, but doesn't displace it much from the ending position of the previous epoch. Since the initial situation has a major influence on the performance of most individuals, it may happen that a very good individual starts twice in an unlucky situation and performs rather bad. With truncation selection, this means that a very good individual may simply be truncated. Therefore, roulette based selection, a less harsh selection method, may be better for this application, since bad performing individuals also have a certain chance of reproducing. Testing epochs in a different order could also be a solution: if all individuals are first tested for one epoch and after that all individuals are again tested for the second epoch, chances that the two epochs have different starting positions are much larger.

Another important issue is the input for the spiking circuits. In the experiments performed, 16 greylevel values were processed using a Laplace filter and this turned out to be enough to make it possible to evolve efficient navigation strategies. However, these strategies are rather simple and use the walls actively. In order to make it possible to evolve better strategies, it may be necessary to provide more and/or other visual information.

Using more photoreceptors is the simplest possibility; up to 50 vision values could be used with the current camera. It might also be worth investigating feeding in other types of visual information. For example, when the airship is far from the walls, the linear camera percepts a more or less grey image without much contrast and the average greylevel of (parts of) the image might be helpful for the spiking circuits to distinguish such a situation from a situation in which it is motionless near the wall. Processing the image with other filters could also be helpful. Elementary Motion Detectors could be used, possibly co-evolving the parameters of these simple but powerful elements.

The measured human performance was not as good as the best evolved spiking circuits, but this experiment could be improved by providing more feedback during the learning epochs. Since human learning is based on feedback, it would be helpful to give more real-time feedback during the training epochs to give the test person the opportunity to see when he is performing well. This could be achieved by displaying the measured forward speed, also used for fitness computation during evolution. In the experiment done, the only feedback was the visual image, but most test persons found it very difficult to see what happened and make out when the robot moved forward.

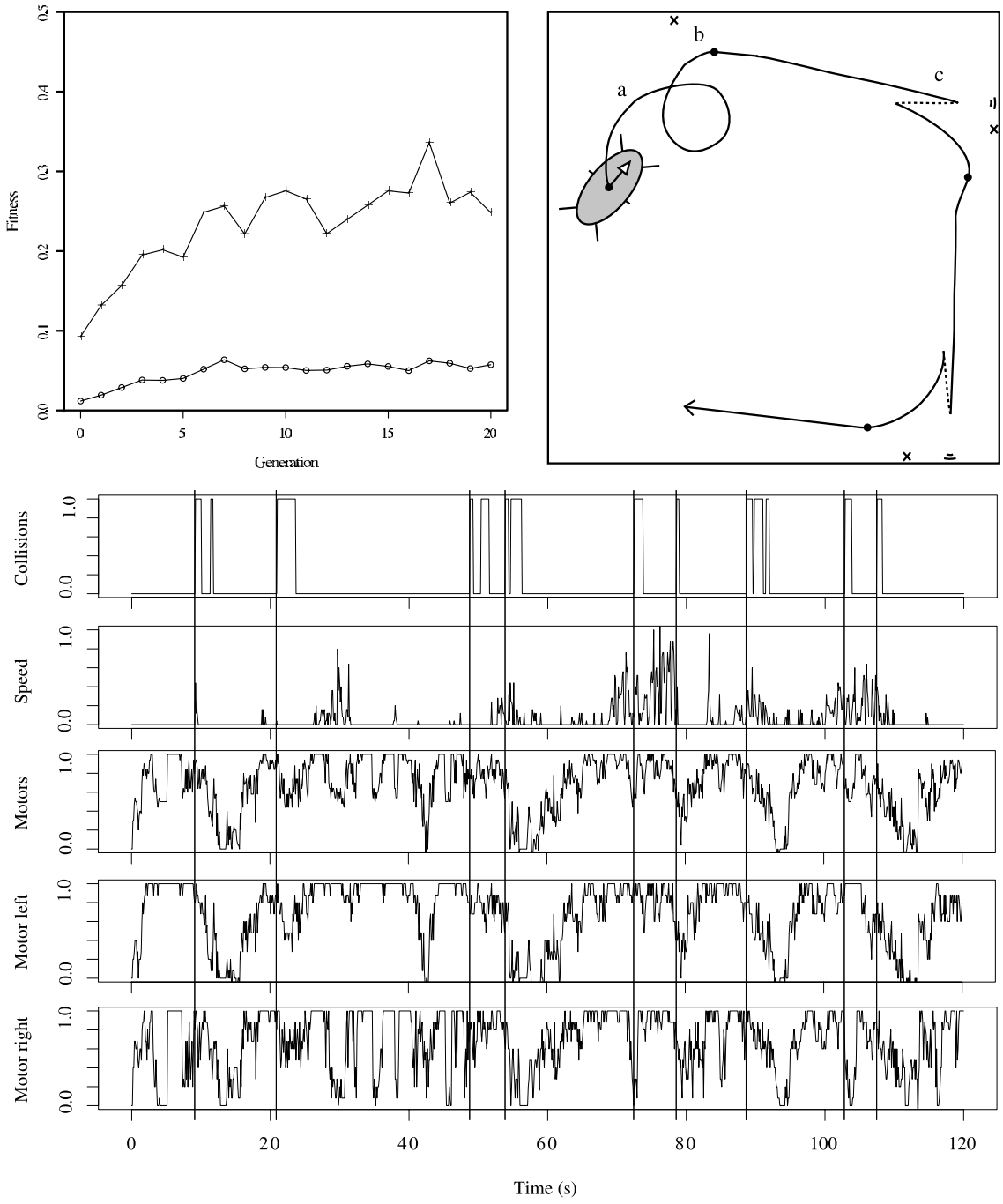

Figure 2.8: *Top left:* Average fitness values of five evolutionary runs without killing (best fitness = crosses, average fitness = circles). Each data point is the average of five evolutionary runs starting with different random initialization of the chromosomes. *Top right:* Hand-drawn estimation of the typical path of the selected best individual (solid line = forward movement; dashed line = backward movement; small curves = frontal collision; cross and circle = place of colission with left back bumper). *Bottom:* Performance of the selected best individual during two minutes. The upper graph shows collisions, as detected by the bumpers. The start of a series of collisions, occuring as the bumpers remain switched when touching the walls, is marked with a long vertical line. The second graph shows an approximation of the forward speed, as measured by the anemometer. The motor graphs show the forward thrust of the engines, which is given by the spiking circuit output ('Motors' is the average of both motors).

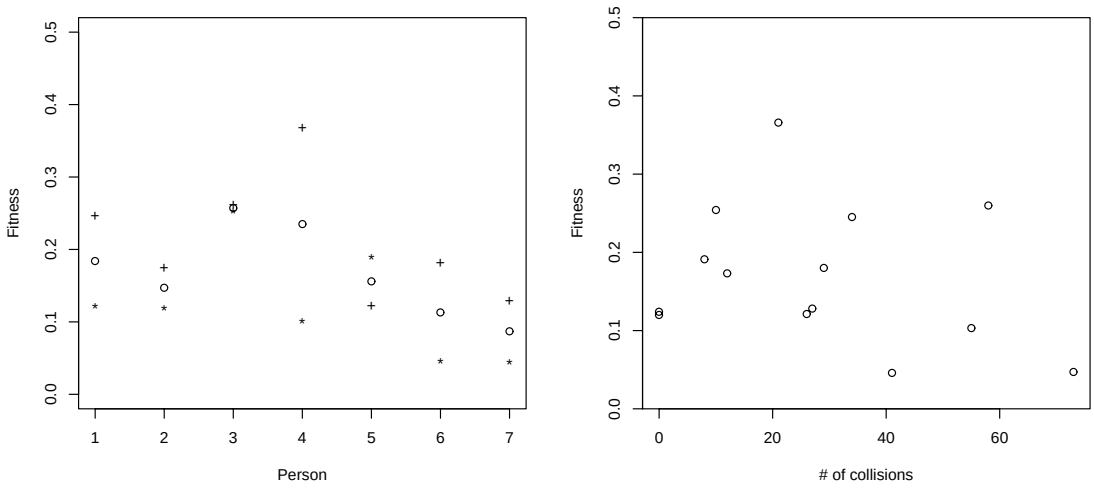

Figure 2.9: Human performance. *Left:* Fitness values achieved by seven test persons (star = first epoch; plus = second epoch; circle = average). The test persons were asked to control the blimp with a joystick, given only the 16 pixels also used by the spiking circuits. The greylevel pixels were drawn on the screen in the correct order, from left to right. Each person first got the opportunity to train during 6 epochs of 40 seconds. Fitness was measured during the following two epochs in exactly the same way as with the neural circuits. *Right:* Correlation between the number of collisions and performance. Each data point represents one of the fourteen epochs performed by the seven test persons.

Chapter 3

i

i (pronounce in Dutch or French, similar to 'ee') is the name of the newly developed application that can be used to evolve neural controllers for (real or simulated) robots. Although both the design and implementation are completely new, the concept is based on the software originally written by Dario Floreano for the experiments with spiking neural networks and the Khepera [4].

Since *i* is a fairly large program, it is impossible to go into too much detail concerning design, implementation and current status in a couple of pages. I will try to describe the major topics though, in order to give a good impression of the possibilities (and restrictions). In the first section, I present the objectives we had in mind when designing the application. After that, I will discuss some design and implementation choices in sections 3.2 and 3.3, respectively. In sections 3.4 and 3.5, the current status and issues for further development will be described.

3.1 Objectives

The major objectives we had in mind when designing *i* were:

Modular Modularity was by far the most important objective. It was required that it would be easy to work with different robots, different communication media and protocols, different neural controllers and different experiment parameters. Also, new robot types, communication types, etc. may have to be added later - this should be very easy to do. Another important point is that a good modular design should make it possible to implement other interesting features, such as manual control and distributed parallel evolution.

Multiple platform MS Windows^{TM} and Linux both have advantages and disadvantages and it would be very convenient to have an application that runs (at least) on these two operating systems.

Robust Evolutionary runs typically last a few days. This means that an application performing such runs should be very robust: it has to be able to recover from all sorts of failures (most probable due to the communication channel between host computer and robot) and keep going for a long time.

User-friendly A graphical user interface, combined with an appropiate structure of menus and dialogs of settings should make the application as easy to use as possible. Loading and saving (e.g., of settings) should also contribute to this. Since the application will be used by researchers who will always want to add/modify code, it's very important to have well-documented code.

Real & simulated A last objective was to make it possible to do experiments with both real and simulated robots. Simulation makes it possible to speed up evolution with large factors, while real-time experiments with real robots make sure that the resulting controllers actually work in reality.

3.2 Design Choices

i was designed as a set of UML diagrams. We started with one large diagram containing all major classes, which we separated into seven packages. Each of these seven packages was further designed in more detail in separate UML diagrams, at this point we first included the most important methods to the diagrams. The seven packages (and two subpackages) are:

main The package where the most important classes reside: the main class, MVC classes (see below) and threads.

comm Communication classes used to establish a connection with the robot: SerialPort, Bluetooth, TcpIp, ...

comm/tcpip Classes used by the ServerModel (see below) to provide TCP/IP server functionality.

exp Experiments, provide the measurement of performance and the coupling between the neural network and robot.

ga All classes related to the genetic algorithm: Population, Individual, Genome, Truncation, Crossover, Mutation, ...

nn Neural network base classes and implementations: SimpleNN, SigmoidNN, SpikingNN, ...

robot The robot base classes and available robots can be found in this package: Blimp, Khepera, KheperaVLSI, Koala, ...

utils Utility classes like Observer/Observable, LaplaceFilter, Loggable/Logger, RandomTools, several thread-safe methods, ...

utils/xml Classes used for loading and saving of XML (setting) files.

Several design patterns [12] have been used, of which the Model-View-Controller and Observer-Observable patterns are the most important. The MVC pattern has been used to design the main classes of the application: there is only one Controller (the main window) controlling several models and views. EvoModel and EvoView are the main classes used for evolution. ManualModel is used in combination with ManualView to manual control a robot with a joystick and display the sensor data. ServerModel makes it possible to remotely control a robot.

The Factory pattern has been used on several occasions (in some cases an adapted form has been applied). Robots, communication media, experiments and neural networks are all created by factories. A number of interfaces were defined to make it easy to add additional implementations later. Good examples of this are the reproduction and genetic operator interfaces (resp. Reproduce and GenOperator).

A last pattern that deserves some attention is the self-designed Loggable-Logger pattern. It can be used in combination with the Observer-Observable pattern to easily log data (such as sensor values) to disk.

3.3 Implementation Choices

Since the previous application was written in C++ and the lab didn't want to switch to a different programming language, it was immediately clear that the implementation would have to be done in C++. Since C++ provides everything required for good object-oriented programming and also results in very fast code (which is required for real-time applications like this), this didn't seem a bad choice and we didn't think about it longer than necessary.

The next important step was to decide on libraries and coding conventions. We needed a GUI library for C++ that would make it possible to compile under both Linux and MS Windows.

Also, we needed a thread library for both platforms. After a little investigation, two possibilities were available: wxWindows [23] and GTK+ [21]. We chose for wxWindows, since it's available for many platforms and uses the native API's of the different platforms (contrary to GTK+). wxWindows is a very extensive framework that makes it very easy to develop applications with a graphical user interface (not quite as easy as Java though). Apart from GUI functionality, it offers many other features among which are also threads.

After our choice for wxWindows, it was no longer necessary to look for other libraries. Coding conventions were composed and a CVS server was set up to host the source code of the application and facilitate easy collaboration with multiple developers. Although wxWindows makes it possible to write platform-independent code, classes that rely heavily on platform-dependent features had to be implemented using preprocessor directives and defines to distinguish between Linux and MS Windows. A good example of this is the SerialPort class.

Standard makefiles are used to compile the application. Under Linux, the GNU C++ compiler is used; on MS Windows, the GNU port MinGW [22]. doxygen [20] (a documentation generator for C++, similar to Javadoc) is used to maintain up-to-date documentation of all classes and files.

3.4 Status

Development still continues at the moment of writing and since *i* is a very extendible tool used for research, development will probably continue for a long time. The application has been tested under both MS Windows and Linux and, after resolving some platform-dependent problems, all basic functions (including graphical user interface) are operative (figure 3.1, left).

Apart from the classes in the experiment and XML packages, all classes are at least partially implemented. All robots and neural networks are operative, the most important communication links are as well. Manual control of all robots is already possible, during which computed speeds and sensor data are displayed. Remote manual control over TCP/IP is also possible (and has been tested - a Khepera in Zurich has been controlled from Lausanne). Evolution with a real robot is also possible; populations and best individuals are saved to disk, fitness values and statistics are logged.

A simple preliminary experiment has been done with the Khepera. The robot was put in a small rectangular environment with equal-coloured walls (a cardboard box). The weights of a simple perceptron with 8 input neurons, 4 output neurons and a standard sigmoid function were evolved, where the performance was measured by the sum of the forward speed of the two separate wheels. Due to time constraints, only 7 generations have been done, but the results so far seem good (figure 3.1, right).

3.5 Further Development

As I already mentioned, *i* is used for research and will therefore never be 'finished'. On other hand, there is currently a list with important features that haven't been implemented yet. More data has to be displayed during evolution, loading and saving of settings has to be made possible. (For this, a good XML library is needed. Expat could be used, but it has the disadvantage that it's written in C and can only be used for reading XML files.) Also, simulation mode and a link with the Webots simulator [19] have to be implemented.

Another feature that will be implemented is the possibility to paralellize evolution, where one *i* server distributes the population over multiple *i* clients that measure the fitness of the assigned individuals. This feature will make many new, very interesting experiments possible, such as massive parallel evolution with (almost) equal robots and environments, but possibly also evolution of a controller for two similar, but unequal robots (such as the Koala and Khepera).

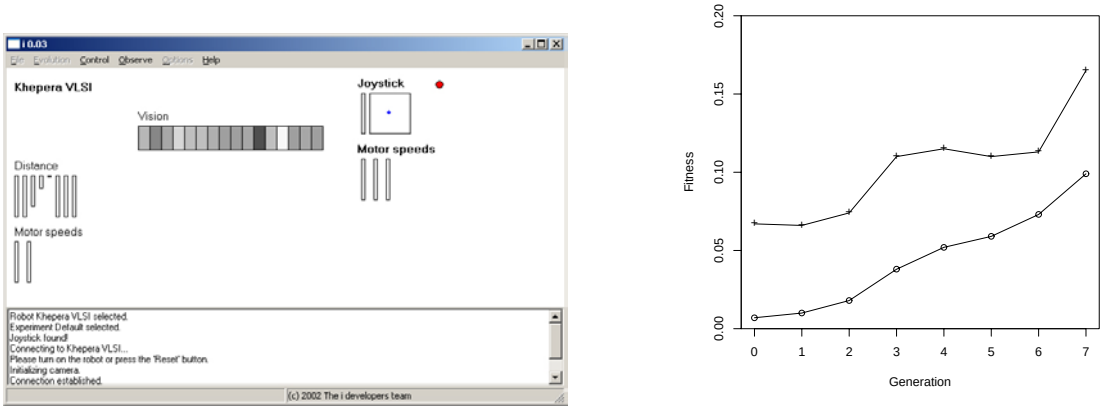

Figure 3.1: *i*. *Left:* Screenshot of *i*, taken during manual control of a Khepera with a VLSI turret and aVLSI camera. The main part of the screen displays the measured sensor data, the joystick values and computed speeds; below is the status log. *Right:* Result of the preliminary experiment with the Khepera (see text).

Chapter 4

Conclusions

Running robotic experiments and developing software are rather different occupations, but it was no problem to combine these tasks during my stay at LSA2, not in the last place because evolutionary robotics was the large common divider. The practical experience with real-life experiments turned out to be very useful in designing an application for evolutionary robotics.

The results of the experiments with the blimp, described in chapter 2, show that vision-based navigational behaviour on a flying robot is evolvable with the current experimental setup; the best evolved individuals use only visual information to effectively navigate around the room, exploiting the properties of the environment.

It is not obvious how trivial tasks like obstacle avoidance could be achieved though, but I already did some suggestions in section 2.8 on what modifications on the experiment could make better results possible. A 3D simulation (implemented on top of Webots [19]) will help to find out what experimental setup is most likely to give good results. A large benefit of simulation is that experiments can be done in far less time, but the disadvantage is that a virtual world is different from the real world - the so-called reality gap. Forms of Hebbian learning will be used in the spiking circuits to make adaptation to a different environment [17] possible (i.e., spiking circuits using learning that are evolved in a simulation should be better able to cope with the reality gap when applied on a real robot than spiking circuits without a form of learning).

Designing and implementing an application like *i* requires quite a lot of time and it is therefore not surprising that it isn't finished yet. Development is still proceeding though and its ease-of-use and modularity make it very pleasant to work with.

Acknowledgements

My internship at EPFL in Lausanne has been a tremendous experience for me and I'd like to thank everyone - both in Lausanne and at home - who has helped to make this possible. Of course, there are a few people I'd like to thank in particular: Dario, for giving me the opportunity to stay at LSA2 for two months; Jean-Christophe, for guiding me and giving me freedom at the same time, for the cool flight and the battery changing during weeks and weekends; Marco, for being my internal advisor and giving me all the freedom I could've wished for.

Bibliography

[1] Bedau, M.A.: 'Philosophical aspects of Artificial Life'. In Varela, F. and Bourgine, P., eds., *Towards A Practice of Autonomous Systems.* Cambridge, MA: Bradford Books/MIT Press, pp. 494-503 (1992)

[2] Dayhoff, J.E.: *Neural Network Architectures.* New York: Van Nostrand Reinhold (1990)

[3] Huber, S.A., Mallot, H.A., Bülthoff, H.H.: 'Evolution of the sensorimotor control in an autonomous agent'. In: *Proceedings of the Fourth International Conference on Simulation of Adaptive behaviour*, MIT Press, pp. 449-457 (1996)

[4] Floreano, D., Mattiussi, C.: 'Evolution of Spiking Neural Controllers for Autonomous Vision-Based Robots'. In Gomi, T., ed., *Evolutionary Robotics. From Intelligent Robotics to Artificial Life.* Tokyo: Springer Verlag (2001)

[5] Floreano, D., Schoeni, N., Caprari, G., Blynel, J.: *Evolutionary Bits'n'Spikes.* Technical report (2002)

[6] Franceschini, N., Pichon, J.M., Blanes, C.: *From insect vision to robot vision.* Phil. Trans. R. Soc. Lond. B, 337, pp. 283-294 (1992)

[7] Gerstner, W.: 'Associative memory in a network of biological neurons'. In Lippmann, R.P.; Moody, J.E.; and Touretzky, D.S., eds., *Advances in Neural Information processing Systems 3.* San Mateo,CA: Morgan Kaufmann. 84-90 (1991)

[8] Goldberg, D.E.: *Genetic Algorithms in Search, Optimization and Machine Learning.* Reading, MA: Addison-Wesley (1989)

[9] Hall, Z.W.: *An Introduction to Molecular Neurobiology.* Sunderland, MA: Sinauer Associates. (1992)

[10] Iida, F., Lambrinos, D.: 'Navigation in an autonomous flying robot by using a biologically inspired visual odometer'. In: *Sensor Fusion and Decentralized Control in Robotic System III, Photonics East, Proceedings of SPIE*, vol. 4196, pp. 86-97 (2000)

[11] Iida, F.: 'Goal-Directed Navigation of an Autonomous Flying Robot Using Biologically Inspired Cheap Vision'. In: *Proceedings of the 32nd International Symposium on Robotics* (2001)

[12] Jia, X.: *Object-Oriented Software Development Using Java.* Reading, MA: Addison-Wesley (2000)

[13] Langton, C.G.: 'Artificial life'. In: Kelemen, J. and Sosik, P., eds., *Advances in Artificial Life: Proceedings of the 6th European Conference on Artificial Life.* Berlin, Heidelberg, New York: Springer-Verlag, pp. 519-528 (2001)

[14] Nolfi, S., Floreano, D.: *Evolutionary Robotics: Biology, Intelligence, and Technology of Self-Organizing Machines.* Cambridge, MA: MIT Press. 2nd print (2001)

[15] Pfeifer, R., Lambrinos, D.: 'Cheap Vision - Exploiting Ecological Niche and Morphology'. In: *Theory and practice of informatics: SOFSEM 2000, 27th Conference on Current Trends in Theory and Practice of Informatics*, pp. 202-226 (2000)

[16] Schwardy, E.: 'Genetic Algorithm as a Result of Phenomenological Reduction of Natural Evolution'. In: Kelemen, J. and Sosik, P., eds., *Advances in Artificial Life: Proceedings of the 6th European Conference on Artificial Life*. Berlin, Heidelberg, New York: Springer-Verlag, pp. 454-457 (2001)

[17] Urzelai, J., Floreano, D.: 'Evolutionary Robotics: Coping with Environmental Change'. In: *Proceedings of the Genetic and Evolutionary Computation Conference* (2000)

[18] Zufferey, J.C., Floreano, D., Van Leeuwen, M., Merenda, T.: 'Evolving Vision-Based Flying Robots'. To appear in proceedings of BMCV2002.

[19] cyberbotics, http://www.cyberbotics.com

[20] doxygen, http://www.doxygen.org

[21] GTK+, http://www.gtk.org

[22] MinGW, http://www.mingw.org

[23] wxWindows, http://www.wxwindows.org