Preface

This paper was written as the result of a research project conducted for the artificial intelligence laboratory of the free university of Brussels. The research project was made possible through an Erasmus exchange between Leiden University and the Free University of Brussels. The project took four months and counted as part of my curriculum at Leiden University. Together with this paper some other texts resulted from this project. Software for the experiments was written, a manual for this software was written, too and two talks about the project were delivered at the summer school on advanced information processing technologies in Varna, Bulgaria.

Without the aid of Luc Steels, Walter van de Velde, (Free University of Brussels) Ida Sprinkhuizen-Kuyper and Egbert Boers (Leiden University), the project would not have been possible. I would also like to thank all the other researchers at the AI-lab and also Maaike Valkenburg ans Jacqueline Dryter, who arranged the Erasmus exchange.

Abstract

This research is an elaboration of research conducted by Luc Steels [STEELS94a]. First Steels' genetic learning algorithm is implemented on a real robot and some basic behaviours are learned. Among these are backward movement, forward movement and halting behaviour. Also a combination of these behaviours using tendencies is tested. Examples of results are presented and some instabilities in the long term behaviour are discussed.

Then some more complex behaviour systems are tested. These are one dimensional obstacle avoidance and two dimensional obstacle avoidance. Results of the experiments are presented. The results show that the Steels' system is capable of learning rather complex behaviours, but that it is very dependent on initial parameters.

Also some theoretical arguments are presented to support the fact that the system can only learn behaviour systems in which there is a direct functional dependency between the actuator and the sensor. Some suggestions are presented on how to extend the system.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Autonomous agents, or robots, that operate in a nontrivial environment need an ability to adapt themselves to that environment. Static behaviour can be good in one particular environment, but can be disastrous in another. An environment can also be too complex to be modeled accurately by the designer of the behaviour. An agent with the ability to adapt itself to changes in the environment has the advantage. Hence the need for learning algorithms.

1.1 Learning Algorithms

Learning algorithms exist in about as many variations as there are people researching them. But it is quite possible to find some criteria that a learning algorithm has to fulfill if it has to be successfully applied to adapting behaviour of autonomous agents.

First of all, the learning algorithm should be able to learn on-line. The agent is learning as it operates, so off-line learning algorithms that must be trained without directly using the learned material are out of the question. Next, the algorithm should be immune to noise. The robot's sensors will not always give reliable results and an action will not always give the same response. Learning algorithms that cannot cope with inconsistent training data are not usable.

Third, the algorithm should be able to learn with a minimum of feedback from the environment. The robot can only communicate with the environment through its sensors and its actuators and it is usually not possible to present the robot with examples of what to do and with explanation of what it is doing right or wrong through these. Clearly, some kind of reinforcement learning is necessary.

A last and perhaps not so obvious criterium which such a learning system has to fulfill is that it has to be provably stable in the long run. If this would not be the case, it would not be safe to use it in real world applications, as one

would never know if the system could collapse suddenly.

The algorithm that was used in this research is a variation on a genetic algorithm, (for an excellent introduction into genetic algorithms, see [GB89]) but many other approaches could have been used as well, for example classifier systems [GB89, HOL86] or neural networks [HKP91]. But the genetic approach was chosen because this research is a continuation of [STEELS94a]. The reason why genetic algorithms were chosen in that paper will be summarized in the next section.

1.2 Autonomy

The goal of this research was to test a truly autonomous learning algorithm. The criteria which an autonomous learning system must fulfill have been discussed briefly in the previous section. But why would we want to use autonomous agents anyway? They have been used, because it is believed that (artificial) intelligence is linked with mobility and autonomous operation. See for example [MOR89].

But as this research is an elaboration of Luc Steels' [STEELS94a] research it is especially necessary to explain the philosophy on which he bases the learning system in his paper. Steels writes: "*We want to stay at the sub- or presymbolic level in which the dynamics of the world is directly coupled to an internal dynamics without prior segmentation or categorization.*" This effectively rules out symbolic learning methods and even classifier systems, that also operate on the basis of categorization and classification. Furthermore he finds it essential that different parts of the system cooperate on an equal basis: "*We want instead to create a 'level playing field' in which one behavioral module cannot inhibit another one. Different modules must cooperate or compete with each other in order to achieve a coherent behavior.*"

To achieve these objectives Steels uses a set of cooperating and competing parallel processes that are trained by a genetic algorithm. The system will be described in some detail in chapter 2. Steels also wants the system to be capable of open-ended evolution and thus of generating its own learning-targets.

Not many experiments have been done with really autonomous robots. A lot of robots are being used in the real world, but almost none of these has on-line learning capabilities. In other experiments learning robots are simulated in a computer after which perhaps the final solution is tried on a real robot (see for example [CHH93]). Other experiments with learning robots were concerned with learning of high level behaviours as opposed to the low level behaviours in this paper. For a collection of papers on learning robots, see for example [VELDE93].

In this paper Steels' learning system is tested on a real robot. In chapter 2 the learning system is described in some detail. In chapter 3 the results of the experiments are presented. Experiments with open-ended evolution have not

yet been performed. In the second part of this paper a theoretical discussion is presented about the limitations of the system and some suggestions are made on how to extend the repertoire of possible behaviours.

Chapter 2

Methodology

The behaviours of the robots in this paper have been implemented using a variation on the Process Description Language, or PDL, of the Artificial Intelligence Laboratory of the Free University of Brussels [STEELS94a]. The variation consisted of using a genetic algorithm to generate a population of very simple processes. This genetic algorithm was then used to find the most optimal combination of processes.

2.1 PDL

The basic objects of PDL are *quantities* and *processes*. Quantities represent the internal states of the robot and its connections to the outside world in the form of real numbers (originally they are integer numbers, but in this research real numbers were chosen by reason of analytical simplicity). If a quantity represents the value of a sensor, it is called an *input quantity*, and it can be used by the robot to obtain information about the outside world. If a quantity is connected to an actuator of the robot, it is called an *output quantity* and it can be used to perform actions on the outside world.

Processes are small pieces of terminating C or C++ code that do not have local variables. They can only work with data through quantities by either inspecting their value or by proposing to add a value (possibly a negative value) to the quantity. Processes need to terminate, because the PDL cycle is based on processes that complete their execution.

All the processes are ran in a parallel fashion. When all processes are finished, their proposed additions to a quantity are summed up and added to the former value of the quantity. Then the output-quantities are written to the actuators and the values of the sensors are written into the input-quantities and the processes are cycled again.

PDL has many more features, among which is an easy portability among

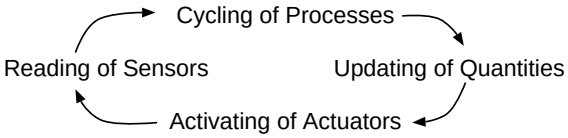

Figure 2.1: The PDL-cycle.

platforms. In this paper no in depth description of the syntax, features and philosophy of PDL will be undertaken. The interested reader is referred to [?].

2.2 The Processes used in this Paper

Although PDL-processes can be made arbitrarily complex, it is an essential part of the PDL-philosophy to keep processes as simple as possible and to let complexity emerge from their interactions. For that reason and for reasons of easy implementation of the genetic algorithm, the processes in this research have been kept extremely simple. Essentially they have the following form:

$$c_{l_p} = \alpha(\gamma_p - q_{k_p}) \tag{2.1}$$

In which c_{l_p} is the change of quantity l proposed by process p, q_{k_p} is the value of the quantity k that process p monitors and γ_p is the desired value (or the *gauge value*) of the monitored quantity. There also is a constant α with which the difference is scaled.

This process effectively tries to keep quantity k_p on the value γ_p by influencing quantity l_p. Whether process p is effective or not has to be determined by the genetic algorithm.

The total change of any quantity q_i can now be determined by the following summation:

$$\Delta q_i = \sum_{p \in \{p | p \in \mathcal{P} \cap (l_p = i)} c_{l_p}\} \tag{2.2}$$

Where $\mathcal{P}$ is the set of all processes. The new value of a quantity is determined by adding the change to the old value. Of course the situation is different for input quantities. These have values that are determined by the sensors. Processes can legally propose changes to input quantities, but these will have no effect.

2.3 The Learning System

When the robot first starts, a behaviour system is generated at random. This means that processes of the form of equation 2.1 are generated at random. The l_p is taken from the range $[0, N-1]$ where N is the number of output quantities

and k_p is taken from the range $[0 \ldots M-1]$ where M is the number of input quantities in the system. The gauge value is taken from the set $\{-\Gamma, 0, \Gamma\}$ where Γ is an arbitrary, but conveniently chosen number. For a system with N input quantities and M output quantities there are $3N \cdot M$ different procestypes.

In our system the processes are stored per processtype. For every different process that appears in the random initialization of the processes, a processtype is created in the robot and the number of instances of this processtype is set to 1. If this processtype happens to be created again, then the number of instances is increased by one. In the remainder of this paper we will sometimes use p and the word 'process' to refer to a certain processtype, instead of to a single process. The number of instances is then written as n_p and $\mathcal{P}$ will be the set of processtypes.

The PDL-system will now be cycled for a certain number of cycles, which we will call W or the *window size*. After this the processes are evaluated according to how well they correlate with the *satisfaction* of the system. This satisfaction is a real number that indicates how well the robot is performing its task. The better the task is performed, the higher the satisfaction. In the experiments the way the satisfaction is calculated is programmed by hand, but in future experiments the robot must determine its own satisfaction.

Processes are said to correlate well with the satisfaction if the average of their proposed changes (c_{l_p}) is in the same direction of the average change of the quantity they want to change (q_{l_p}) if the satisfaction has increased, or if their proposed changes were in the opposite direction of the total change when the satisfaction has decreased. The size of the proposed changes relative to the total proposed change in the same direction is also taken into account when determining the strength of the correlation.

To determine this correlation, we first need to know the contribution $b_{p,t}$ of a process p at time t. This is calculated as follows:

$$b_{p,t} = \begin{cases} \operatorname{sgn}(\Delta q_{l_p}) \frac{c_{l_p}}{\Delta^+ q_{l_p}} & \text{if } c_{l_p} \geq 0; \\ \operatorname{sgn}(\Delta q_{l_p}) \frac{c_{l_p}}{\Delta^- q_{l_p}} & \text{if } c_{l_p} < 0. \end{cases} \tag{2.3}$$

Where $\Delta^+ q_{l_p}$ is the summation over all the positive contributions to quantity q_l and $\Delta^- q_{l_p}$ is the summation over all negative contributions. All these values are taken at time t, but to simplify the notation, we have left out all the suffixes t.

The b_p are sampled at every cycle of the system. But not every cycle a genetic step is performed. This is done only every W steps. After these cycles, the numbers of the different processtypes are updated with the following formula:

$$n_{p,t+W} = n_{p,t} + \left\lfloor \epsilon_{t+W} \zeta \frac{\sum_{i=0}^{W-1} b_{p,t+i}}{W} n_{p,t} \right\rfloor \tag{2.4}$$

Where ζ is a constant that determines the speed of growth of processes and ϵ_{t+W} is -1 if the satisfaction has decreased between time t and $t+W$ and 1 if

the satisfaction has increased. If the satisfaction has stayed equal, then ϵ_{t+W} is set to a random number in the range $\langle -1, 1 \rangle$

2.4 Maintenance of Diversity

Some rudimentary precautions were taken to preserve diversity in the population so that the system will not end up in a local maximum. One of the measures was the setting of ϵ to a random value if no change in satisfaction happened. Another measure was to insert a random new process (not a processtype!) if the satisfaction stayed constant or the number of processtypes got less than a certain limit.

The details of the experiments will be discussed in chapter 3 with the results. A remark that can be made is that the measures to maintain diversity seem a bit ad hoc. They are copied from the work by Steels [STEELS94b], but it appears that some improvements are in order. Actually the whole genetic algorithm seems to be liable to improvement. But this will be discussed in chapter 8.

The reader must be aware that between different experiments minor modifications were sometimes made to the software. These modifications will be described and explained in chapter 3 about the experiments.

Chapter 3

First Practical Results

The genetic system was implemented on a real robot. This robot, called 'Lola' is based on a standard B12 mobile robot base from Real World Interface Inc. It is enhanced with all kinds of sensors (touch, infra-red, light, speed and distance) and can ride around by using a battery-powered electric motor. It also has a PC-compatible computer on board, on which the control-software can be run. The robot is thus capable of completely autonomous behaviour.

3.1 The First Experiments

The first experiments that were performed are meant to duplicate the results that were obtained through simulation in [STEELS94b]. The experiments in this paper involved forward movement, backward movement and halting behaviour. These very simple behaviours cause the robot to move forward, backward and to stop, respectively.

The robot has disposal of one sensor and one actuator. The sensor returns the distance the robot has covered in the previous cycle. As the average cycle time is fixed this is a measure of the robot's speed. The actuator sets the velocity of the robot. However the sensor measures the velocity independently from the setting of the actuator. The robot now has to learn the connection between these two quantities.

The results are presented in figures 3.1 and 3.2 for forward and backward movement.The behaviours take place convincingly.

The results for halting behaviour are presented in figure 3.3. To test if the halting behaviour was real, an artificial disturbance was introduced by accelerating the robot through a touch-sensor reflex. These simple reflexes were maintained in the robot to prevent it from damaging itself by trying to ride through a wall. The disturbances caused the peaks in the graph at cycles 800 and 900. If it had come to a halt because there were no processes left to drive

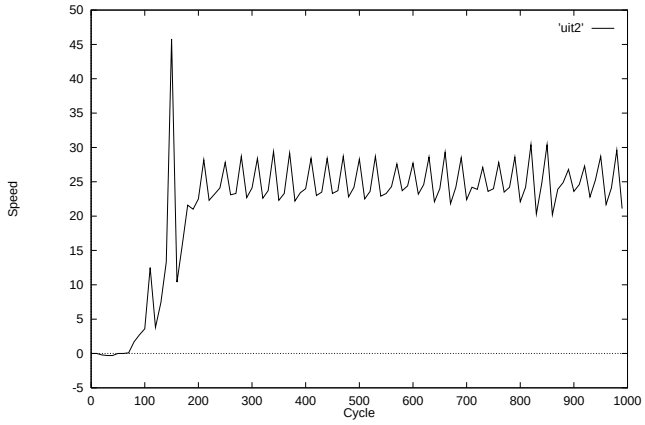

Figure 3.1: Forward movement.

description	value
Number of processes at start	50
Value of α (see equation 2.1)	0.01
Values of γ_p (see equation 2.1)	-25, 0, 25
Speed to be learnt	-25, 0, 25
Value of ζ (see equation 2.4)	10

Table 3.1: Parameters of the learning system.

it, then it would not have come to a halt again after the disturbance. But it did reduce its speed to zero, so the halting behaviour was real.

One must keep in mind that these results are only examples. The graphs are made from single runs of the robot that showed the correct behaviour. Different runs give different graphs, which look similar, but are by no means the same. In table 3.1 the parameters of the system are given.

The periodicity shown in the graphs for forward and backward movement is an artifact. This was caused by the fact that the results were periodically written to the floppy disk drive of the robot. This caused a periodic lengthening of the process cycle and thus an apparent increase of the robot's speed (because it covered more distance). In later experiments this effect was removed by using a RAM-disk instead of a floppy disk.

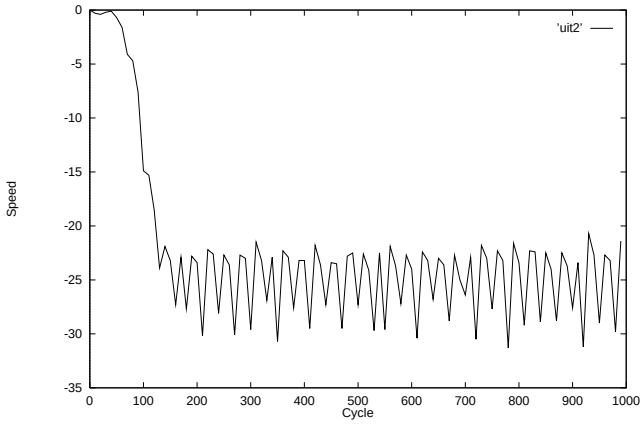

Figure 3.2: Backward movement.

3.2 Changing Tendencies

The previous experiments were undertaken with a fixed goalvalue for the quantities. But what we finally want is a robot that can operate independently using open-ended evolution. It then has to be able to choose its own behaviour systems. This can be implemented by using tendencies. A tendency is associated with a certain behaviour system. If the robot has a certain tendency, the accompanying behaviour system is active. The tendency is actually associated with a (control-) goal, which is expressed as a value that has to be achieved by a specified quantity.

The tendencies can now be turned on and off. In the beginning a random tendency was turned on. If it became active, the processes belonging to that tendency were allowed to influence the quantities. All other processes were held inactive. The processes belonging to the active quantity were also the only ones that were used in the genetic search. If the goalvalue had been attained, or if the system had ran too long without reaching the goalvalue, another tendency was generated randomly.

This approach has the result that behaviour systems are trained one after another. In the beginning behaviour systems will react slowly to a changing tendency, but later in the run when the behaviour systems have been trained sufficiently they will react much quicker. This is shown in the first part of figure 3.4.

In the second part of the graph, an undesirable effect is seen to happen. Here the behaviour systems still seem to be active, but their goalvalues are not reached anymore. An explanation can be found in the numbers of the different processtypes, shown for forward movement in figure 3.5.

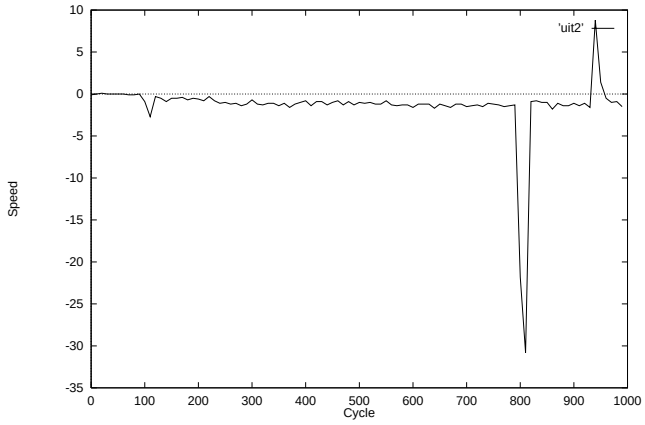

Figure 3.3: Halting behaviour.

In the beginning the process "solving" the problem $(50 - q_0)$ grows and holds the absolute majority and all other processes have zero instances. But then, around generation number 140, processes that damage the solution appear and grow in number. We are now confronted with the strange effect that processes that were first exterminated by the genetic algorithm, because they were found to be inferior, are now favoured although they still are detrimental to the system.

Apparently some processes get assigned a fitness to which they have no right. How this can happen is still very unclear. In chapter 4 some possible causes are discussed. Also some situations are presented in which it can be proven that the system will assign the correct fitness. Further drawbacks of the system are discussed in chapter 6.

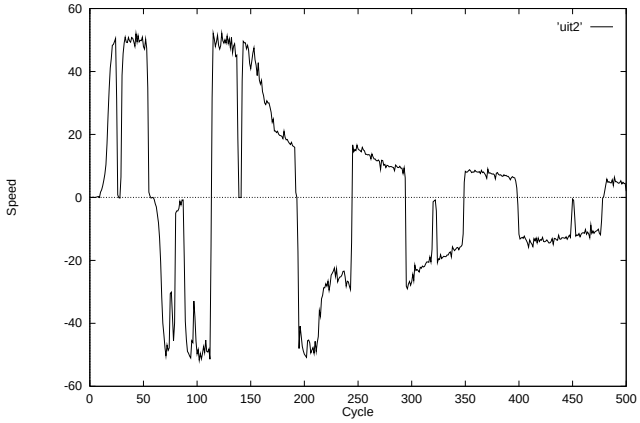

Figure 3.4: Value of the wheelcounter.

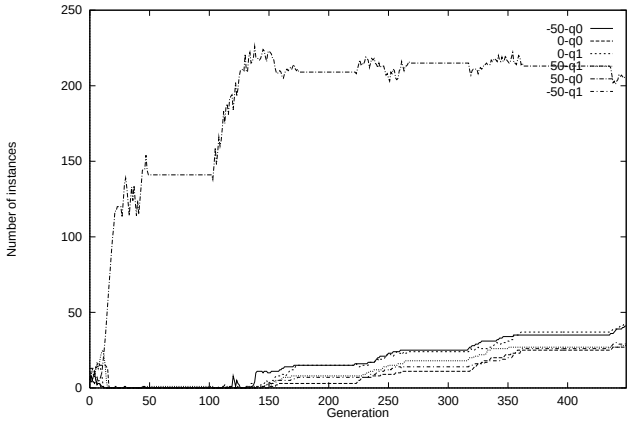

Figure 3.5: Numbers of processes for forward movement.

Chapter 4

Some Remarks about the First Results

The results of the first experiments with the genetic learning PDL system implemented on a robot showed great promise. The system most definitely learns and it learns very quickly , independently and on-line, which are all prerequisites for a learning system on an autonomous robot. Some caution is in order, though. Long term behaviour of the system does not seem to be particularly stable, the system is very dependent on its initial parameters and the system has only been used in a very simple setting with only one sensor and one actuator. We will now discuss the system and its problems in some more detail. The system discussed will, unless specified otherwise, be a system with two quantities, one sensor and one actuator that react to each other and that have a more or less linear relation. Systems with multiple sensors and actuators are more difficult to analyze. These kinds of systems will be discussed in chapter 6.

4.1 Initial Behaviour

An unpleasant behaviour of the system that was observed was the impossibility to learn if the number of processes at the start was chosen too high. The system then remained in an oscillatory state with the quantities' values very near zero. The oscillations were observed when the parameters of the system were as in table 3.1 and the number of processes (per behaviour system) was increased to 100 or more.

A possible explanation of this effect is that if the number of processes gets too high, their influences tend to average each other out. The relative differences between the numbers of instances of the different processtypes gets smaller when the number of processes increases. If the relative influence per processtype gets too small, then the number of instances of this process will never change, so

there will never be any evolution.

If this is the right explanation or not has not been investigated. The problem was solved very effectively by limiting the number of processes per behaviour system to about 50 divided among six different processtypes at initialization. Another possible solution (if more processes are needed) could be to use real numbers for the number of instances and thus effectively allowing non integer numbers of instances or to use extra noise in the beginning in order to "shake" the system out of local maxima.

4.2 Limit on the Number of Processes

There is an important reason to limit the number of processes in a behaviour system. If the number of instances of a certain processtype exceeds a certain limit, the changes these processes propose to a quantity will cause it to get a value that is further away from the gauge value than it was before the "correction". The result will be an exponentially growing oscillation around the gauge value.

This can only happen if the correction proposed by all processes together is more than two times the deviation that was to be corrected. If we have a processtype of the form:

$$q_1 \leftarrow \alpha(x - q_0)$$

in which $\leftarrow$ means that the value at the right of the arrow is added to the value of the term to the left of it. The q_1 is the output quantity, the q_0 is the input quantity, x is the gauge value and α is the sacling factor. If we also have a relation between the two quantities of the form:

$$q_1 = y \cdot q_0$$

then the maximum number of processes can be found by solving the inequality:

$$\left| \frac{M\alpha\left(x - (x + \epsilon)\right)}{y} \right| < |\epsilon|$$

which solves to:

$$M < 2\frac{y}{\alpha} \tag{4.1}$$

In the system tested in this paper ($y = 20, \alpha = 0.01$) this would result in a maximum number of 4000 instances of processtypes monitoring q_0 and 200 instances of processtypes monitoring q_1. These are upper limits that have to be observed in any case; if these are exceeded the system will always break down.

4.3 Instabilities

The system showed an instability in the long run (see chapter 3). The value of the goalquantity reached by the system got closer and closer to zero, because the

number of instances of wrong processes, which we will call *parasitic* processes grew. An exact mathematical explanation of this phenomenon has not been found yet, but some observations can be made.

The first observation is that, if the quantities change in only one direction, the system will increase the number of good processes and decrease the number of bad processes. This can be supported by the following argument.

Suppose the goalquantity changes in such a direction that the satisfaction increases. Suppose furthermore that the quantity does not cross one of the gauge values of the processes. This is a reasonable assumption, because the breakdown of the system takes place far away from the gauge values of the processes, anyway. As the changes are in only one direction, all the δq will have the same sign. For all processes the $b_{p,t}$ (see equation 2.3) will also have the same sign (as no gauge values are crossed). Then obviously for all processes the summation of the $b_{p,t}$ will have the same sign as the $b_{p,t}$ themselves. Processes that have proposed changes in the same direction of the quantity will have a positive sign, the others will have a negative sign. The value of ϵ (see equation 2.4) will be positive, as the satisfaction has increased. The number of instances of processes working in the direction of the satisfaction will increase, of processes that work in the opposite direction will decrease.

If the quantities have changed in a direction decreasing the satisfaction, then the argument goes analogously, but with inverted sign for ϵ. The observation can probably even be extended for unbiased systems whose quantities change in both directions in one genetic window.

The second observation that can be made is that for reasonable numbers of processes (that is less than y/α, see equation 4.1) the changes of the quantities in a completely linear system will be in only one direction. A completely linear system is a system, in which all the quantities react to each other without delay and in a completely linear way. In such a system, the change proposed by the processes will always be less than the distance of the value of the goalquantity to the average gauge value of the processes. This distance will become smaller and smaller by the action of the processes, but it will never become zero and it will also never change sign.

The conclusion that can be drawn from these two observations is that the breakdown of the system is caused by non-linearities in the outside world to which the system is not prepared. This looks like a huge problem for a real-world learning system, but fortunately, the breakdown can be prevented by limiting the number of instances of processes allowed. Also a mechanism should be added that removes processes that are not useful. This can simply be done by multiplying the number of instances of each processtype by a value $\beta, 0 \leq \beta < 1$.

So although the long-term breakdown of the learning system can be prevented by applying a few simple measures, the exact mechanics of the how and why are not understood. Why the system starts to break down when a certain number of processes is reached nor what the connection is between the initial inability to learn and the long-term breakdown. These gaps in our understand-

ing of the behaviour of the system severely limits the applicability of it. If we cannot state the condition under which the system will remain to function, it is not safe to use it in real-world applications.

On the other hand, we have now got some measures under which a system exhibits a reasonable long term stability. We can now do some more complex experiments with the assurance that the system will not suddenly break down.

Chapter 5

Further Experiments with the Robot

The experiments with the basic behaviours and with the tendencies were only a starting point for the experiments with this learning system. The behaviours that were learnt by the robot in these experiments were very simple indeed. In this chapter some experiments are described in which the robot is taught more complex behaviours. The first of these is something which could be called "obstacle avoidance". In fact it is a one dimensional reaction to a touch sensor. The important thing of this experiment is, that the robot needs to learn a discontinuous behaviour, something which at first sight might seem impossible.

For real obstacle avoidance, the robot has to be able to turn away from an obstacle, as well as to back away from it. This behaviour is found to be too complex for a single behaviour system, but an experiment is conducted to check if a robot can learn to drive straight, when using both a translation and rotation actuator. This will also be useful for phototaxis behaviour.

The last experiment is indeed concerned with phototaxis. In this experiment a robot has to orient itself towards a light source by using its rotation actuator. The robot will not be able to learn this.

5.1 One Dimensional Obstacle Avoidance

In this experiment the repertoire of sensors of the robot was extended with a touch sensor. When touched, this sensor returned a value that was a function of the time it was pressed and the pressure with which it was pressed. In fact the behaviour of the touch sensor is a bit more complex. A typical touch sensor activation is illustrated in figure 5.1

The behaviour is caused by the fact that the physical touch sensors of the robot Lola are 1-bit sensors. They are not sensitive to pressure. Therefore the

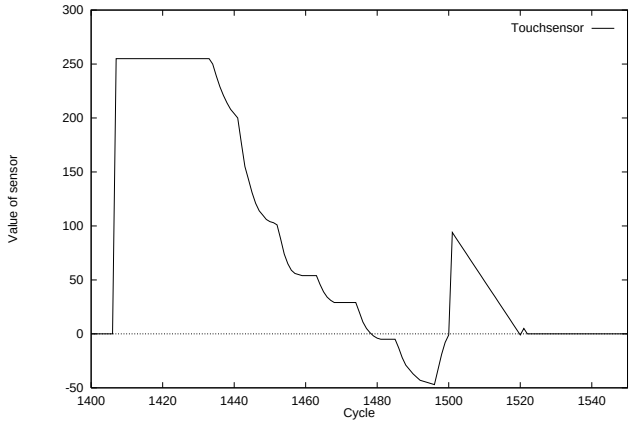

Figure 5.1: Typical touch sensor behaviour.

translation velocity that is set (but possibly not realized) is added to the value of the touch sensors. Then for every timestep that it is activated a counter is increased, to simulate the fact that the longer the touch sensor remains depressed, the worse. Lastly, the maximum value of the touch sensor is limited to (an arbitrary) 255.

Now when a touch sensor gets depressed, it quickly jumps to 255, as the speed added is usually higher than 255. The counter now begins to increase. When the robot finally goes backwards, the speed decreases rapidly and eventually becomes negative, hence the drop below zero of the touch sensor. When the touch sensor is released, the value jumps to a positive value again, as the counter was not yet zeroed. The value of the counter then goes to zero in a linear way.

This might not be the ideal behaviour of a touch sensor, but in the author's opinion it is a reasonable realistic way of behaviour. The value is dependent of time and (indirectly) pressure, and a little oscillation at the release is also not unrealistic.

The robot was quite capable of learning how to back up from obstacles. In the experiment it was equipped with a forward movement tendency, so it would bump into an obstacle several times. It did not remember that there was an obstacle. Once it had backed up a little, its "urge" to move forward became too strong so that it would ride into the obstacle again. This was hoped to be solved in the next section in the two dimensional obstacle avoidance.

The results of a sample run of the learning system are shown in figure 5.2. In this figure the values of the three quantities of the robot are shown. These are the touch sensor, the wheelcounter and the translation velocity. Only the last is an actuator, the first two are sensors. It can be seen that when the robot runs into an obstacle, it pushes against the obstacle for awhile and then goes

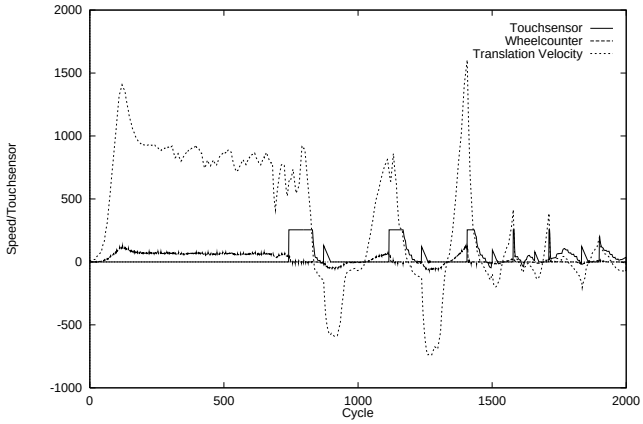

Figure 5.2: One dimensional obstacle avoidance.

backwards to get free from it. This is repeated several times, with a less and less strong reaction. The most important processes that are involved in this are shown in figure 5.3.

In this figure the explanation for the discontinuous behaviour of the robot can be found. An unchanging system of PDL-processes would have great difficulties implementing the obstacle avoidance without being dependent on the values the sensors return. The backward movement processes of the touch sensors would have to compete with the forward movement processes of the tendency. The number of processes needed for the backward movement would have to depend on the number of forward movement processes, the value of the touch sensor and the gauge values of the processes. But this is not the case in our system.

Every time the robot bumps into an obstacle, it "learns" how to avoid the obstacle by increasing the number of backward movement processes and decreasing the number of forward movement processes. When it has cleared the obstacle and the touch sensor has become silent again, it "forgets" the backward movement and starts to move forward again.

Unfortunately, every time the robot bumps into an obstacle, the number of noise processes is also increased. This could be the cause of the weakening reaction of the robot to the touch sensor. But that could also be caused by its ever decreasing forward speed, because it gets less and less time to recover from its contacts with the obstacle.

The parameters of the system in this experiment are shown in table 5.1. The satisfaction function was the sum of the absolute value of the goalvalue minus the wheelcounter and the absolute value of the touch sensor.

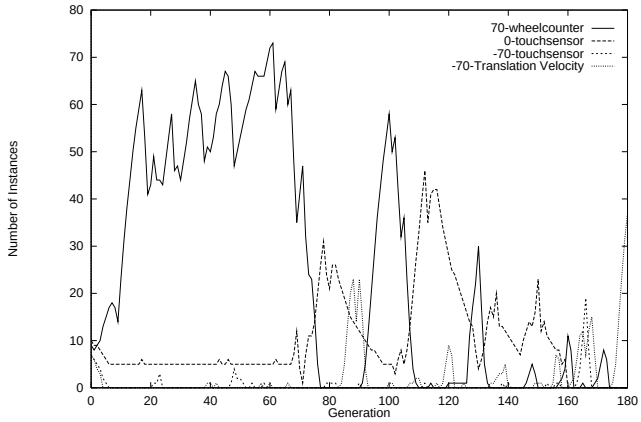

Figure 5.3: One dimensional obstacle avoidance.

description	value
Number of processes at start	75
Value of α (see equation 2.1)	0.01
Values of γ_p (see equation 2.1)	-70, 0, 70
Speed to be learnt	70
Value of ζ (see equation 2.4)	10

Table 5.1: Parameters of the learning system.

5.2 Driving Straight

The one dimensional obstacle avoidance is not a very practical behaviour system. After the robot has "avoided" the obstacle by backing up from it, it completely forgets that it was there and will run into it again. This shows clearly from figure 5.2. In practical obstacle avoidance, the robot usually moves in a two dimensional environment and avoids obstacles by backing up *and* changing the direction in which it moves. The robot will then move forward again, but as its direction of movement has changed a little bit, it will probably miss the obstacle. This can be considered true obstacle avoidance. Unfortunately, preliminary experiments showed that this was a bit too complex for a single behaviour system. Thus we concentrate on teaching a robot to learn to walk straight in a two dimensional world.

In order for Lola to be able to change direction and thus to become two dimensional, we need to add a new sensor and a new actuator. The actuator will allow her to start a rotation and the sensor will provide her with information about how quickly she is rotating. We also have to change the satisfaction

function a little bit. We want the robot to move in straight lines, so we lower the satisfaction if it is rotating.

That the addition of an extra sensor and an extra actuator is not without penalty is shown in figure 5.4. In this figure the system is shown while learning to move forward in a straight line. In contrast with the very fast learning of the system that had only one sensor and one actuator, the system with multiple sensors learns much slower.

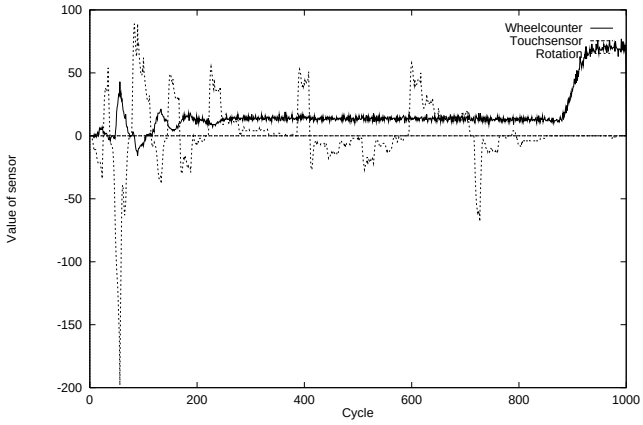

Figure 5.4: Slow learning with multiple sensors.

This can be explained by two facts. The first is that the possible relations between the sensors and actuators are much more complex. The second is that the number of different processtypes is much higher, while there are still only a few that give correct behaviour in the given situation. The extra processes cause a lot of "noise". There is also a greater risk of reducing the number of instances of one of the correct processtypes to zero.

These factors slow down the learning process, but as can be seen from the figure, does not prevent the robot from learning how to ride in a straight line.

5.3 Phototaxis Failure

The last experiment that was conducted appears to be extremely simple. The robot has to orient itself towards a light source using two light sensors and a rotation actuator. The outputs of the two light sensors are preprocessed into one input quantity representing the difference between the two sensors and in one input quantity representing the average of the two sensors. This is a somewhat more useful form for the simple PDL processes.

Unfortunately, the learning task only appears to be simple. In reality the robot is not able to direct itself towards the light source in any stable way. It does seem to react on its sensors, turning in the right way when one of the sensors gets more light than the other, but contrary to what was hoped and expected, the robot is not able to reach a stable, unchanging position directed towards the light source.

So although the problem seems at first sight to be similar to forward or backward movement, in that it has to stabilize the value of an input quantity using output quantities, it is in fact quite different. In the movement experiments, there was a rather direct functional relation between the sensor and the actuator. In the phototaxis experiment on the other hand, there is no such functional relation. There is only a functional relation between the integral over the actuator and the value of the sensor. In such a situation, the PDL-learning system is usually not able to learn or to perform. This will be discussed in more detail in chapter 6.

Chapter 6

Limitations to the PDL Processes

The processes used in the genetic learning PDL system are very simple indeed. It is therefore to be expected that they cannot solve all problems. In this chapter some theoretic arguments are presented as to the limitations of the learnable functions and some suggestions are made to improve the range of learnable functions.

6.1 Functional Dependency between Actuator and Sensor

In order to be able to understand the behaviour of PDL-functions, we have to get an idea of how the connection between the different quantities in the system is. For this we assume that there is a function f converting an activator into a sensor value.

$$q_0 = f(q_1) \tag{6.1}$$

in which q_1 is the actuator- and q_0 is the sensor quantity. This function will not have to be completely invertible, although we will assume that it is locally invertible in a later proof. It is also possible that there is no functional relation between the actuator and the sensor. This will also be discussed later.

We can now give a formula that links the value of quantity q_1 at a certain timestep with its value at the next timestep. This formula can be derived using equation 2.1 and combining it with equation 6.1. for convenience we will assume that γ_p is zero. The resulting formula will then be:

$$q_{1,t+1} = q_{1,t} - \alpha f(q_{1,t}) \tag{6.2}$$

Now if we have a functional relation, there are three possible behaviours of the system. These are: convergent, divergent and periodical or chaotic. This is shown in figures 6.1, 6.2 and 6.3. In these figures the equation 6.2 is shown and a trajectory of q_1 when the system is iterated.

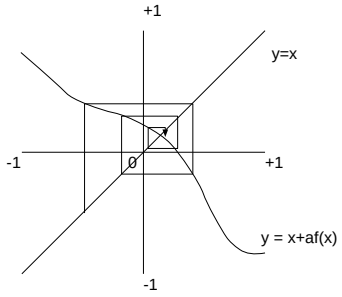

Figure 6.1: Convergence of the quantities

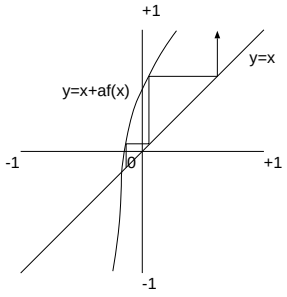

Figure 6.2: Divergence of the quantities

It is obvious that only the first form of behaviour is desirable. But what kinds of functions show exactly this behaviour? They have to avoid chaotic or periodic behaviour and they have to converge (at least locally) to a solution.

Chaotic and periodic behaviour can be avoided by disallowing functions f so that $x + f(x)$ does not have local maxima or minima. There are, of course functions that do have local maxima and local minima and that do not exhibit chaotic or periodic behaviour, so the condition is sufficient, but not necessary. If we do have a function with local maxima and minima, we should be aware of periodic or chaotic behaviour.

Divergence can be avoided by taking care that $0 < f'(0) < \frac{2}{\alpha}$. Divergence away from the solution does not occur if we can find a limit d so that for every $|\epsilon| < |d|$ it is true that:

$$|f(f^{-1}(\epsilon) - \alpha\epsilon)| < |\epsilon| \tag{6.3}$$

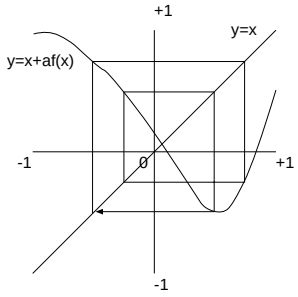

Figure 6.3: Periodic behaviour of the quantities

This expression was derived by assuming that the sensor quantity q_0 has a value of ϵ. The actuator quantity must then have a value of $f^{-1}(\epsilon)$ (hence the aforementioned local invertibility of the function). This was then substituted in 6.2 after which equation 6.1 was used to calculate the new value of the sensor quantity. The condition 6.3 is now derived by making sure that the new deviation is smaller than the old one.

The inequality can be solved by assuming that ϵ is so small that $f(\epsilon)$ can be approximated by $f'(0)\epsilon$ and $f^{-1}(\epsilon)$ by $\frac{\epsilon}{f'(0)}$. Substituting these and solving for $f'(0)$ (assuming $\alpha > 0$) yields the aforementioned limits on $f'(0)$:

$$0 < f'(0) < \frac{2}{\alpha} \tag{6.4}$$

Note that in the case that the goal value of the sensor is not zero, the derivate can be taken at this goal value and the formula remains valid. Also compare this result with that from equation 4.1.

6.2 No Functional Relation

Other interesting problems occur if there is no functional relation between the value of the actuator quantity and that of the sensor quantity. If this is the case then a process of the form of equation 2.1 cannot directly control the value of a sensor. Extra constraints need to be fulfilled for a system to be controllable in such a case.

An example of such a case was the phototaxis experiment from section 5.3. An other case where there is no functional relation is when one tries to learn forward or backward movement to a certain position instead of to a certain speed. One does not monitor the measured speed in such a case, but the distance covered by the robot. Preliminary experiments showed that this can also not be learnt.

In both these cases there is a functional dependency between the integral (or the summation in the discrete case) over the actuator and the value of the sensor. In a situation like that the changes proposed by the PDL-processes will not necessarily lead to a stable value of the actuator. This is even so in preprogrammed systems. The PDL-processes will then not only not be able to *learn* the desired behaviour, there is not even a way to *implement* it.

We will attempt some mathematical analysis of the situation where there is a functional dependency between the integral or summation over the actuator and the sensor. Cases in which there is another kind of relation or in which there is no relation whatsoever between the sensor and the actuator can also exist. But in the former situation, the analysis will be similar and in the latter no analysis is necessary as it is clear that nothing at all can be learnt.

For the analysis we need two quantities, the sensor at time t $q_{0,t}$ and the actuator at time t, $q_{1,t}$. We will assume that there is one process of the form:

$$q_{1,t+1} = q_{1,t} - \alpha(0 - q_{0,t})$$

which is, in fact an ordinary PDL-system with zero as the goal value.

Now we can write $q_{1,t}$ in terms of $q_{1,0}$ and the series of $q_{0,\tau}$ where $0 \leq \tau < t$. In other words:

$$q_{1,t} = q_{1,0} - \alpha \sum_{\tau=0}^{t-1} q_{0,\tau} \tag{6.5}$$

The summation over all the actuator values, S_t, on which the value of the sensor is functionally dependent can now also be written in terms of $q_{1,0}$ and the $q_{0,\tau}$:

$$S_t = \sum_{\tau=0}^{t} q_{1,\tau} = (t+1)q_{1,0} - \alpha \sum_{\tau=0}^{t-1} (t-\tau)q_{0,t} = (t+1)q_{1,0} - \alpha \sum_{\tau=0}^{t-1} (t-\tau)f(S_\tau) \tag{6.6}$$

This might look like a recurrence, but it is in fact not, because S_t in this formula is only dependent on S_τ's that have been calculated before. It is true, however that S_t can now be written completely in terms of $q_{1,0}$ and possibly an S_0, which would have to be added separately.

In a real system this extreme determinism will not be possible, as there will always be some noise. However it is a good starting point for a mathematical analysis. We can for example investigate what conditions have to be fulfilled for the system to reach an unchanging state. In such a state $q_{0,t}$ will have to be zero otherwise the actuator will be changed. But it must also be true that $S_t = S_{t+1}$ because otherwise $q_{0,t+1}$ will change as well.

So for the system to be stable at time t it must both be true that $q_{0,t} = f(S_t) = 0$ and the difference between S_t and S_{t+1}, $\sum_{\tau=0}^{t-1} f(S_t) = 0$. Not many functions fulfill this criterium, so it is unlikely that a stable state will be reached in a certain system.

The limited repertoire of relations between sensors and actuators that can be learnt by the PDL-system is a disadvantage. In chapter 7 we will discuss a possibility to extend the repertoire.

Chapter 7

How to Learn more Complex Things

The genetic learning PDL system has shown some remarkable learning of simple behaviour systems like forward movement, backward movement, halting behaviour and even a rudimentary form of obstacle avoidance. Problems occurred, however, when the behaviour systems that had to be learnt were more complex or were based on multiple sensors and actuators or consisted of a more complex relation between sensor and actuator.

Another problem was, that in most cases, except in the case where tendencies were used, all processes had an equally big influence at all times. This was especially clear in the obstacle avoidance experiment. Here all the processes were active (and were under the influence of the genetic algorithm) at the same time. This resulted in a system that learned how to drive backwards every time when it encountered an obstacle, but which forgot this once it was clear of the obstacle.

The tendencies as implemented in the experiment in chapter 3 were of not much use, as they were controlled with no purpose in mind and without a learning algorithm. Clearly a more intelligent solution is needed.

7.1 Using a Finite State Machine

Using tendencies as a means to activate and deactivate behaviour systems is not a bad idea. Only a more flexible method of turning them on and off is needed. Ideally we would want a system that has one basic element that both implements the tendencies as well as builds the behaviour systems. However tendencies seem at the moment to require a rather different mechanism than the behaviour systems.

Behaviour systems monitor a sensor and influence an actuator accordingly

in a dynamical and linear way. Tendencies on the other hand, are switched in a non linear way when a certain condition has been met. These kinds of behaviour suggest a finite state machine that turns on and off tendencies.

A finite state machine has an input and an internal state and a set of rules that determine the next internal state when it receives a certain input when it is in a certain state. In the case of a finite state machine controlling behaviour systems the internal state consists of the active tendencies. The inputs are the events that can happen to the different active behaviour systems. These events are for example: achieving maximum satisfaction or not achieving maximum satisfaction after a certain time limit.

The finite state machine can now switch between different groups of active behaviour systems according to the total behaviour of the system. The system is illustrated in figure 7.1

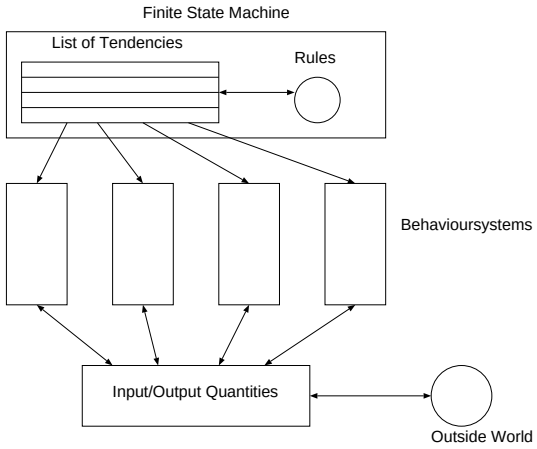

Figure 7.1: Finite state machine controlling behaviour systems.

7.2 Adding Learning Capabilities

The finite state machine is capable of controlling the behaviour systems in a coherent way, but it is not very flexible. Furthermore it must be determined beforehand which behaviour systems have to be used at which moments. This is an undesirable situation, as this might be impossible in certain cases and because we are researching learning behaviour in the first place.

The behaviour systems themselves can learn according to the same rules as those that were used in the earlier parts of this research. As long as they are only trained when their tendencies are active. They should also not become too complex. The system will then have random behaviour in the beginning, but

will be able to perform rather complex tasks in the end. This is comparable to a system that has a high level instinct, but that has to learn the basic workings of its sensors. A system of this kind is in fact very useful, as it is able to adapt to changes in its sensors and actuators.

However, we would want the system to be able to learn more complex things. Therefore the finite state machine must be made flexible as well. In other words, we want the system to learn the rules that govern the transitions between the different states (that is behaviour systems).

These rules can be modeled as condition action pairs. The condition is the input and the present state and the action is the next state. We now want a system that can find out which rules are useful and which are not. For this we need to solve a temporal credit assignment problem, because we do not know when rules that perform a certain action will receive payoff.

Notice that the system is very similar to a classifier system. Classifier systems also use rules in the form of condition/action pairs and we could consider the list of behaviour systems and events as the message list. Usually classifier system use a system like bucket brigade or Q learning, but we will use a slightly different scheme.

The credit assignment system for the classifiers is based on a running average of the temporal correlation between actions and satisfaction. For every rule a running average is maintained which will be high (close to 1) if the rule being active is usually followed by a high satisfaction. The average will be low (close to -1) if the activation of the rule is usually followed by decreasing satisfaction.

With a_t the 0 if a rule is not active at time t and 1 if it is active and with s_t 1 if the satisfaction has increased at time t and -1 if it has decreased, the following formula can make a reasonable estimate of this correlation:

$$(-\gamma)(1-\beta)\begin{pmatrix} a_t s_t + \gamma a_{t-1} s_{t-1} + \ldots + \gamma^t a_0 s_0 + \\ \beta(a_{t-1} s_t + \gamma a_{t-2} s_{t-1} + \ldots + \gamma^{t+1} a_0 s_1) + \\ \vdots \\ \beta^t a_0 s_t \end{pmatrix} \tag{7.1}$$

Here γ and β are constants that determine how quickly the past is forgotten.

This might seem an extraordinarily complex formula, but it can in fact be calculated by keeping track of only two variables per timestep: the summation $A_{tot,t}$ over the activations and the total of the running average, $F_{tot,t}$. They can be calculated by:

$$A_{tot,t} = (1-\beta)a_t + \beta A_{tot,t-1} \tag{7.2}$$

and

$$F_{tot,t} = (1-\gamma)A_{tot,t} * s_t + \gamma F_{tot,t} \tag{7.3}$$

We have now got an estimate over all time of how well a rule behaves.

Note that the satisfaction function is something different from the satisfaction functions of the basic behaviour systems. The basic behaviour systems have

very simple satisfaction functions consisting of one goalvalue and one goalquantity. The satisfaction function of the whole system, on the other hand, can be arbitrarily complex.

We can now use a genetic algorithm to find a good population of rules. This genetic algorithm can for example copy the rules with fitness that is less than zero with a certain (small) probability and rules with higher than zero fitness always or with a very high probability. To maintain diversity we will have to insert new random rules into the population. This can be considered a form of mutation. The resulting system is shown in figure 7.2.

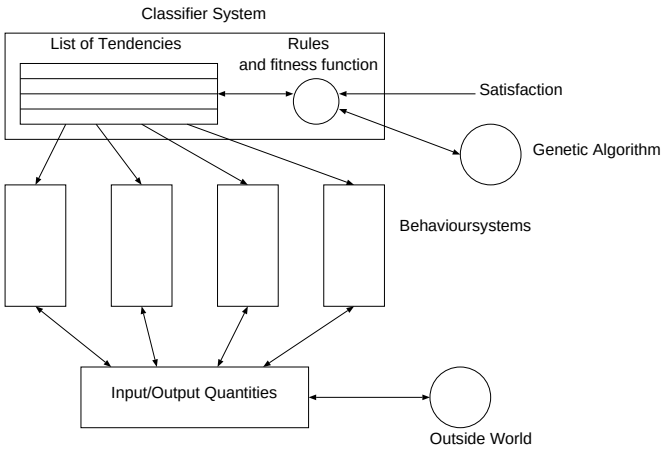

Figure 7.2: Learning PDL system with classifier system.

Many variations on this basic theme can be tried out. The genetic algorithm proposed is still very primitive and will probably not work very well if we get too many rules with high fitness. ALso the classifier system can be extended as not to allow every active rule to perform its action. Rules will then be chosen with a probability that is proportional to its fitness.

7.3 Augmenting the Abilities of the Behaviour Systems

In chapter 6 some arguments were put forward to show that the basic behaviour systems consisting of simple PDL-processes are a bit too simple for real world learning tasks. In this section we will show that these are in fact equivalent to a very simple form of neural network and make a suggestion as to how the capabilities can be extended.

The simple PDL processes used in this research are very similar to simple neural networks. This becomes clear if we write the system in a matrix notation.

Using equation 2.2 we can write:

$$\vec{Q}' = M\vec{Q} + \vec{G} \tag{7.4}$$

Where $\vec{Q}$ and $\vec{Q}'$ are the vectors $q_0 \ldots q_n$ of quantities. Vector $\vec{Q}$ contains the old values and $\vec{Q}'$ contains the new values.

The elements of matrix M can be calculated by the following formula:

$$m_{ij} = \alpha \sum_{p|p\in\mathcal{P}\cap l_p=i\cap k_p=j} -1 \tag{7.5}$$

In other words, the matrix elements represent the number of times a certain quantity is added to another quantity when cycling the system. The scaling factor α was necessary, because it is also present in the processes themselves.

The vector $\vec{G}$ consists of the values that are added to quantities in a PDL-cycle. These are given by equation:

$$g_i = \alpha \sum_{p|p\in\mathcal{P}\cap l_p=i} \gamma_p \tag{7.6}$$

The definitions of all the symbols can be found in section 2.2.

In the matrix form of equation 7.4 the PDL system looks remarkably like the description of a recurrent network with asymmetric inhibiting connections and linear activation functions. The elements of matrix M are the weights of the connections and the vector $\vec{G}$ represents the thresholds of the network nodes. The quantities themselves map on the nodes. The values of the quantities are the activations of the nodes.

From neural network theory (see for example [HKP91]) we know that networks that have only linear threshold functions are not very powerful. Also in chapter 6 it was shown that the only type of relation that can be learnt is a direct functional dependency between actuator and sensor which also has to fulfill certain constraints if one wants to avoid chaos or divergence. A somewhat more powerful basic process could be based on a device from control theory, known as the PID * (for an introduction into control theory, see for example [DSW90]).

A PID is a device that bases its output on a weighted sum of terms that are proportional to the derivate of the measured function, to the measured function itself and to the integral over the measured function. By using the extra terms one can implement controls between sensors and actuators that do not have a direct functional dependency.

A genetic algorithm can be used to search for the right weights of the three terms and for the right combinations of sensors and actuators. In such a genetic search one could use crossover between individuals having the same sensor and

*Suggested by drs E.J.W. Boers in a personal communication.

actuator. One should also take measures to maintain diversity, as it could be necessary to have multiple sensors influencing multiple actuators for the behaviour system to function.

The processes based on PID's are a direct extension of the simple PDL processes from this thesis if we add a small offset to their outputs. If we choose a negative weight for the proportional term of the PID and zero weights for the other terms they are in fact equivalent. We thus have a real extension to the old system.

This system has not been implemented yet. For the PID's to work, some extra information about the quantities has to be maintained and choices have to be made with what accuracy that is done. Also the genetic algorithm has to be modified considerably. It cannot be kept a simple as it was in this research. The basic determination of fitness, however, can still be kept the same. Examples that could be tested with this new system are the phototaxis from sectionsect:phototaxis and possibly the experiment in which a certain point has to be reached with translation.

Chapter 8

Conclusions

The simple PDL processes learning with a genetic algorithm seem to be a promising learning system. They learn in real time and on line without supervision from a teacher. Furthermore the system learns very quickly.

The system has been implemented on a real robot, which learned forward movement, halting behaviour and backward movement and also a combination of these three behaviours. Also a very simple one dimensional obstacle avoidance was learned, but this was found to require constant action of the genetic algorithm, instead of produce a stable behaviour system.

In the long term behaviour instabilities were found, but these could be avoided by limiting the number of instances of each processtype and by eliminating unused processes. Also some problems with the extreme dependency on good staring parameters were found, but these too could be solved by choosing the right size of the starting population. Unfortunately a sound theoretical basis for stable behaviour was not found.

Theoretical analysis showed severe limitations to the kinds of behaviours that can be implemented using the simplest form of PDL-processes. It was suggested in this paper to use PID's instead. The use of a simple classifier system was suggested to get different behaviour systems to cooperate. Both these suggestions were not tested in an implementation.

Learning PDL processes also appear to be similar to simple neural networks. It has to be investigated if results from the very rich field of artificial neural network research (see e.g. [HKP91]) can be applied to PDL-processes.

Genetic learning PDL processes fulfilled many requirements necessary for use in a real robot. They learn on-line, without supervision and very quickly in simple situations. This shows great promise, but unfortunately a very important requirement for use in real world applications, the provability of stable behaviour, was not fulfilled. Also the system as tested is not able to implement more complex or discontinuous behaviours. It is hoped that the suggested improvements can change this.

The simple system showed great performance in simple situations. It is hoped that with the results and suggestions of this paper a more complex system can be built that performs as well in more complex situations.

Bibliography

[CHH93] Dave Cliff, Inman Harvey, Phil Husbands, *Explorations in Evolutionary Robotics*, Adaptive behavior Vol 2, No 1, MIT Press 1993, pp73-110

[DSW90] Joseph J. DiStefano, Allen R. Stubberud, Ivan J. Williams, *Feedback and Control Systems 2nd ed.*, Schaum's outline series, McGraw-Hill inc. New York, 1990.

[GB89] D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison Wesley, Reading MS, 1989.

[HKP91] J. Hertz, A. Krogh and R. G. Palmer, *Introduction to the Theory of neural Computation*, Addison Wesley, Reading MS, 1991.

[HOL86] J. H. Holland, *Escaping Brittleness, The possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems*, in [MCM86] pp. 593–623.

[LAN89] Christopher G. Langton (ed.), *Artificial Life*, the proceedings of an interdisciplinary workshop on the synthesis and simulation of living systems held september 1987, in Los Alamos, New Mexico, Addison-Wesley, Redwood City, CA, 1989

[MOR89] Hans Moravec, *Human Culture: A Genetic Takeover Underway*, in [LAN89] pp. 167-199

[STEELS94a] Luc Steels, *The artificial life roots of artificial intelligence*, Artificial Life journal, Vol 1,1, MIT Press, Cambridge, 1994

[STEELS94b] Luc Steels, *Emergent functionality in robotic agents through online evolution*, Proceedings of alife IV, MIT Press, Cambridge, 1994.

[VELDE93] Walter Van de Velde (ed.), *Toward Learning Robots*, MIT Press, Cambridge, Massachusetts, 1993.