Latent Learning and Action Planning in Robots with Anticipatory Classifier Systems

Wolfgang Stolzmann & Martin Butz

Institute for Psychology, University of Wuerzburg, Germany
{stolzmann, butz}@psychologie.uni-wuerzburg.de

Abstract. Two applications of Anticipatory Classifier Systems (ACS) in robotics are discussed. The first one is a simulation of an experiment about latent learning in rats with a mobile robot. It shows that an ACS is able to learn latently, i.e. in the absence of environmental reward and that ACS can do action planning. The second one is about learning of the hand-eye coordination of a robot arm in conjunction with a camera. Goal-directed learning will be introduced. This combination of action planning and latent learning leads to a substantial reduction of the number of trials which are required to learn a complete model of a prototypical environment.

1 Introduction

In [10,this volume, pp. ???-???] ACS were applied to simple "woods" environments to show how they work and in which kind of environments they can be applied. In this chapter ACS will be used to control robots.

Firstly, an experiment of latent learning in rats is simulated with a mobile robot. In the field of learning classifier systems this experiment was first simulated by Riolo [6] with CFSC2. This simulation is not comparable with the robot-simulation described here, because here the internal representation of the environment is much more complicated for a robot-simulation (e.g. the robot has to deal with aliasing states).

Secondly, an experiment about learning the hand-eye coordination of a robot arm in conjunction with a camera is replicated with an ACS. This experiment was first done by Birk [1]. He used a learning algorithm called "Stimulus Response Learning".

For the simulation of the first experiment it is necessary to enable ACS to do *action planning*. In the second experiment the performance of an ACS can be improved, if *goal-directed learning* is used. Both algorithms *action planning* and *goal-directed learning* are introduced.

In this paper the same notations are used as in [10,this volume, pp. ???-???].

2 An Experiment about Latent Learning

Latent learning is defined as learning in the absence of environmental reward, i.e. $\rho(t) = 0$ for all time steps t. Therefore latent learning cannot be simulated with usual reinforcement learning techniques. In this section an experiment about latent learning in rats is introduced, that was developed by Seward [7]. This experiment is simulated with a Khepera robot that learns by using an ACS. For this simulation the ACS must be able to learn latently and to do action planning. As shown in [10, this volume, pp. ???-???], an ACS is able to learn an internal model of its environment without getting environmental reward, i.e. it can learn latently. How action planning can be done in an ACS is shown in section 2.3.

2.1 An Experiment about Latent Learning in Rats

Seward [7] developed an experiment about latent learning in rats. He used a simple T-maze with two distinguishable goal boxes F and E shown in figure 1.

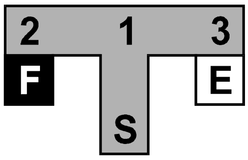

Fig. 1. A simple T-maze

During an exploration phase a rat was put into the T-maze and explored it without getting any reward. After that the rat was not fed for 24 hours. Then it was placed in one of the goal boxes and received food for one minute. Finally, the rat was placed in the box 'S'. 28 of 32 tested rats demonstrated latent learning by going to the good-goal box. A similar experiment was done by Croake [3]. During a control experiment without an exploration phase only about 50% of the rats go to the good-goal box. This experiment proves that the rats learned latently during the exploration phase, because there was no environmental reward.

2.2 A Khepera Robot

A Khepera robot is a mini mobile robot (5 cm diameter) distributed by K-Team S.A (*http://www.k-team.com*). Its features are 2 independent DC motors with encoders and 8 infra-red sensors (emitter/receiver) (cf. figure 2). The motor speeds range from -20 to +20, The infra-red reflection values range from 0 to 1023 and the infra-red ambient values range from 0 to 511.

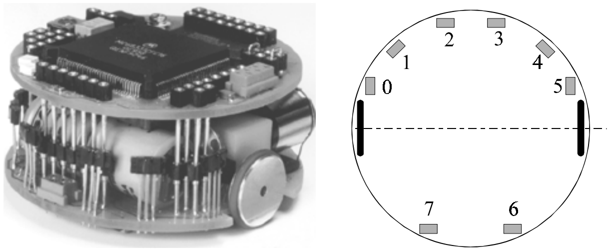

Fig. 2. A Khepera robot

This robot will be used to simulate Seward's experiment about latent learning in rats. The goal boxes of the T-maze must be distinguishable for the robot. This can be done with an infra-red light in the goal-box F. The infra-red light enables the robot to distinguish the light goal-box F from the dark goal-box E.

The Khepera robot will learn by using an ACS. So it is necessary to define an input interface and an output interface that enables the ACS to interact with the environment. It is possible to define the following 5 detectors and 3 simple actions for an ACS by using the 8 sensors and 2 motors of a Khepera-robot:

$$d_1 = \begin{cases} 0, \text{if there is nothing in front of the robot} \\ 1, \text{if there is a wall in front of the robot} \end{cases},$$

d_2 as d_1, but on the left of the robot,

d_3 as d_1, but behind the robot,

d_4 as d_1, but on the right of the robot,

$$d_5 = \begin{cases} 0, \text{if there is no light near to the robot} \\ 1, \text{if there is a ligth near to the robot} \end{cases},$$

l, to make a 90 degree left turn,
r, to make a 90 degree right turn,
f, to go forward until a wall or a choice-point is reached.

These detectors and simple actions are very robust. In all simulations a robot never got stuck in a corner of the T-maze.

Figure 3 shows the ACS's internal representation of the T-maze. This internal representation contains four important aliasing states: $2_N = 3_E$, $2_W = 3_N$, $2_S = 3_W$ and $2_E = 3_S$.

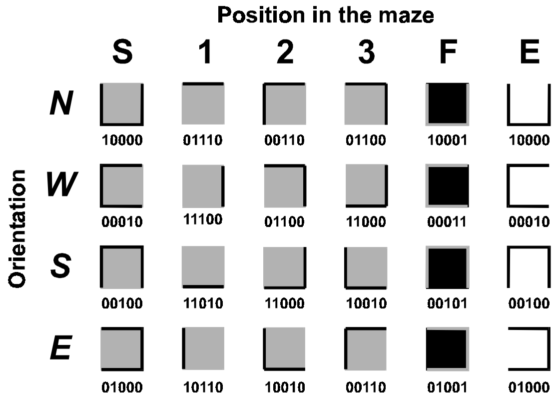

Fig. 3. The ACS's representation of the T-maze (*N=North, W=West, S=South, E=East*)

A simulation of Seward's experiment with a real Khepera robot would take a very long time. Therefore a real robot was only used for about 100 behavioral acts to show that the input interface and the output interface works. For the learning experiment we used a simulated robot.

2.3 Action Planning in ACS

An ACS can use its internal model, i.e. its reliable classifiers for action planning. When doing action planning the expectation part *E* of a reliable classifier is used to calculate an anticipation of the next environmental state S_{t+1}^{ant}. It is assumed that S_{t+1}^{ant} is equal to the next state S_{t+1} and so it is possible to calculate all new satisfied reliable classifiers. For each new satisfied reliable classifier this can be done again. The same process can take place backwards, too. That is how a bi-directional search is used to generate an optimal sequence of classifiers (at least the system "believes" it is optimal) with action parts that lead from the actual state to a desired goal state. Action planning is done without any interactions with the environment.

Let us assume that we have a reliable classifier for each possible behavior in each state shown in figure 3, i.e. an internal model of the T-maze. (This assumption is wrong because some of the states are aliasing states, but nevertheless we make this assumption to explain action planning in ACS.) And let us assume that the current state is S_N and that the goal state is F_S. Now three algorithms are described that can be used to find a sequence of classifiers in the internal model that enables the ACS to reach the goal.

It's possible to use the *forwards-directed breadth-first search* to find such a sequence:

1. Let $c_i = C_i - A_i - E_i$ for $i = 1, \ldots, k$ be all reliable classifiers with: S_N matches C_i.
2. Calculate $S_1^{ant}, \ldots, S_k^{ant}$ with $S_i^{ant} = passthrough\left(S_N, C_i^{ant}\right)$.

3. If $F_S \in \{S_1^{ant}, \ldots, S_k^{ant}\}$, then stop the calculation, else repeat 1. and 2. for $S_1^{ant}, \ldots, S_k^{ant}$ instead of S_N.
4. Repeat this recursively until F_s is found or a maximum number of repeats $s_{\max}$ is reached. $s_{\max}$ is called *maximum number of steps during action planning*.

If F_s is found, then execute all action-parts of the classifiers of the sequence. This execution of all action-parts can be regarded as goal-directed behavior. This is similar to lookahead planning for classifier systems that was introduced by Riolo (1991).

It is also possible to use the *backwards-directed breadth-first search* to find a path:

1. Let $c_i = C_i - A_i - E_i$ for $i = 1, \ldots, k$ be all reliable classifiers with: F_s matches E_i.
2. Calculate $S_1^{ant}, \ldots, S_k^{ant}$ with $S_i^{ant} = passthrough(F_S, C_i)$.
   (Note: $passthrough(F_S, E_i)$ would be wrong here!)
3. If $S_N \in \{S_1^{ant}, \ldots, S_k^{ant}\}$, then stop the calculation, else repeat 1. and 2. for $S_1^{ant}, \ldots, S_k^{ant}$ instead of F_s.
4. Repeat this recursively until S_N is found or $s_{\max}$ is reached.

If S_N is found, then execute all action-parts of the classifiers of the sequence starting with the last classifier.

A third possibility is to combine both algorithms and use *bi-directional search*.

All of these three algorithms have the problem of cycles. For example, in S_N a left turn l leads to S_W and in S_W a right turn r leads back to S_N. Cycles can be avoided if only messages S_i^{ant} are considered for the *breadth-first search* that have not been calculated before.

Note: It is absolutely necessary to use a small value for $s_{\max}$ because of the exponential memory requirements of the search-algorithms.

Figure 4 shows an example for action planning:

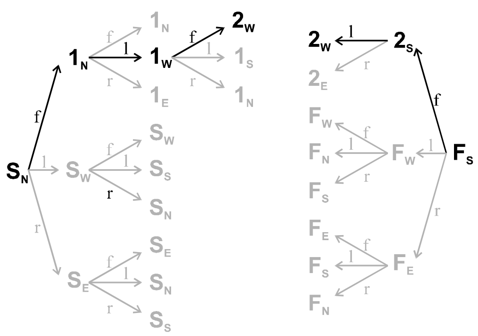

Fig. 4. An illustration of a bi-directional search while the ACS is doing" action planning

The robot is located in the position S_N. Its goal is to reach the left goal box F_S. If the *forwards-directed* search and the *backwards-directed* search reach a common state, here 2_w, then the action planning is successful and the ACS knows the actions that are necessary to reach the goal, here flflf.

2.4 Results

Two series of simulations were done, pure simulations and Webots-simulations. Webots is a open mobile robots simulator (*http://www.cyberbotics.com*). In contrast to a pure simulation the Webots-simulation has noisy detectors and effectors. Figure 5 shows two examples for behavioral and perceptual errors that happen during a Webots-simulation.

Fig. 5. Errors during a Webots-simulation

In the left example the robot has done several left and right turns without going forward in the mean time. So its orientation is south-east instead of east. If the robot wants to go forward now it does not work. In the right example the robot is very near to the lower wall and does not detect the upper wall, i.e. the robot believes that there is no wall to its left-hand side.

One simulation consists of 100 runs. One run is organized as follows: The robot is controlled by an ACS that uses *specification of unchanging components*, *action chunking* to avoid aliasing states and *action planning*. The ACS starts without any knowledge and executes 100.000 behavioral acts. After each 10.000 behavioral acts the execution is interrupted and the bi-directional search checks whether the internal model (reliable classifiers) contains a path with a minimum number of behaviors from 1_N to F_S (or E_S). If the bi-directional search finds such a path, then we assume that the T-maze is learned.

The following parameters were used for the ACS: $\rho(t) = 0$ for all *t*, $b_q = 0.05$, $b_r = 0.0$, $p_x = 0.0$, $\theta_r = 0.95$, $\theta_i = 0.2$, $u_{\max} = 2$, $a_{\max} = 3$, $s_{\max} = 5$

$p_x = 0.0$ means that only roulette-wheel selection is used to select a classifier.

In Figures 6 and 7 the abscissa contains the number of behavioral acts and the ordinate the number of robots that learned the T-maze. Figure 6 shows the results of a series of 10 pure simulations and 1 Webots-simulation with food in F, i.e. a path from 1_N to F_S has to be learned. The result of the Webots-simulation is not different from the other results. This shows that an ACS is able to deal with low behavioral and perceptual errors. In [2] ACS in non-deterministic environments, e.g. in environments with noise in the actions, are discussed in further detail.

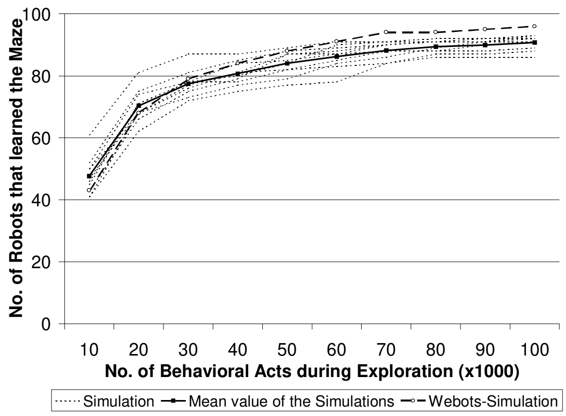

Fig. 6. Results of the simulations with food in F

In addition to the series of simulations with food in F we have done a series of 10 pure simulations with food in E, i.e. a path from 1_N to E_S has to be learned, to analyze whether there is a different learning rate in both series. Figure 7 shows the results.

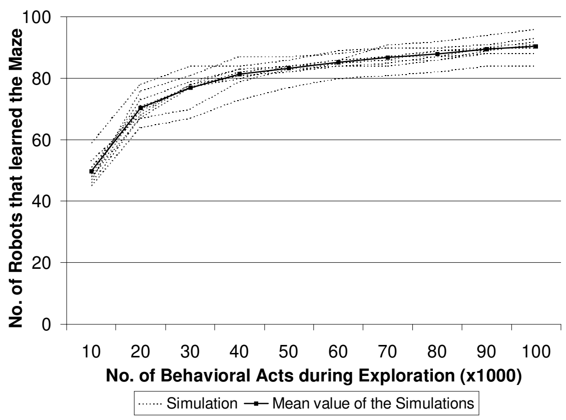

Fig. 7. Results of the simulations with food in E

A comparison of figures 6 and 7 proves that the results of the simulations do not depend on the goal during the post-eating trial because the mean values for 10 simulations with food in F and the mean values for 10 simulations with food in E are equal. A priori this is not clear because the light goal-box F is distinguishable from the position S but the dark goal-box E is not distinguishable from the position S. In other words the robot's perception of E is an aliasing state but the perception of F is not. This important difference between F and E does not have any consequences for the results of the simulation.

The performance can be improved, if also random selection is used to select a classifier. Figures 8 and 9 show the results for $p_x = 0.5$. All other parameters are the same as above.

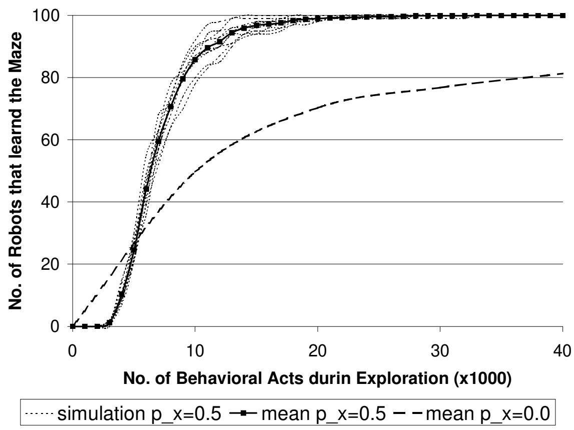

Fig. 8. Results of the simulations for $p_x = 0.5$ with food in F

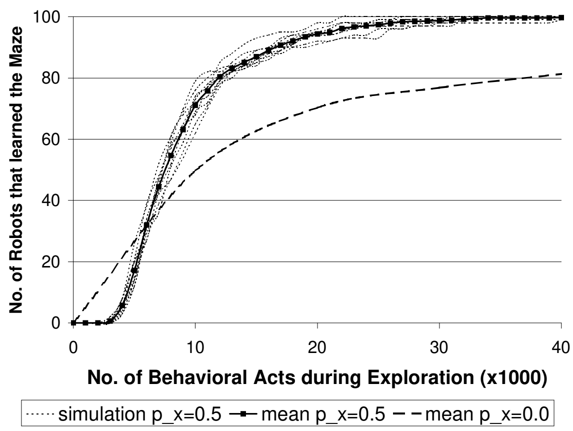

Fig. 9. Results of the simulations for $p_x = 0.5$ with food in E

A comparison of figures 8 and 9 shows that this time there is a difference between the simulations with food in F and the simulations with food in E. The performance in figure 8 is better than in figure 9.

An objection to this simulation of Seward's experiment could be that the task looks very simple and the ACS needs about 10.000 behavioral acts to reach 90% in figure 8. The task looks simple but it is not. The experiment was replicated with students: The states were presented as shown in figure 3 and the students were told to press three different keys to explore this finite state world for 15 minutes. After that only two of 11 students knew how to reach one of the goal-boxes.

3 An Experiment about the Hand-Eye Coordination of a Robot

This section focuses on the integration of action planning and latent learning in ACS, called goal-directed learning. It is merging the planning possibility with the latent learning process in ACS, building an action planning phase that will alternate with the normal exploration phase. The experiment discussed in section 2 is unsuited to discuss an integration of action planning and latent learning because first the environment is explored and then, after learning finished, action planning is used to reach a goal state. In this section an experiment about learning the hand-eye coordination of a robot in conjunction with a camera is discussed. This experiment was introduced by Birk [1]. A camera is monitoring a matrix representing a planar surface with a robot-arm on it. The robot is able to simply move in the four directions of the planar surface, grip a block, transport it to a different position and release it again. It has to learn the effects of its possible motor actions.

This scenario is especially useful to show the positive effects of goal-directed learning, because the different states do not occur with the same probability. Only on the position where the block is situated gripping will be successful, so that the system normally will move around the field without the block most of the time. Latent learning of how to grip the block and to move it to a different position seems to be impossible. Besides, it will be possible to compare the positive effects of goal-directed learning in ACS to those Birk made in his work. Birk tested this experiment also on a real robot arm in combination with a real camera, which shows the possible application in reality.

First of all, the experiment is described. After that the method of goal-directed learning is introduced. Finally, the experimental results are presented and discussed.

3.1 The Experiment

3.1.1 The Coding

The camera (the eye) is vertically watching a black planar surface. The picture is diverted in a $n \times n$-matrix. Over this surface, the robot-arm with a red gripper is installed. It can move in the four directions North, South, East and West over the plane. Besides there is a blue block that can be gripped by the robot-arm and be moved around the planar surface. The robot-arm has got a feeling sensor that switches from 0 to 1 when it feels a block under it. That is why the possible environmental states can be coded in $\{\text{black}, \text{red}, \text{blue}\}^{n^2} \times \{0,1\}$. The possible motor actions are N = North, S = South, E = East, W = West, G = Grip and R = Release. As mentioned, Birk implemented this experiment also on a real robot. Therefore the coding was not changed. Figure 10 shows the robot and its environment for n = 5.

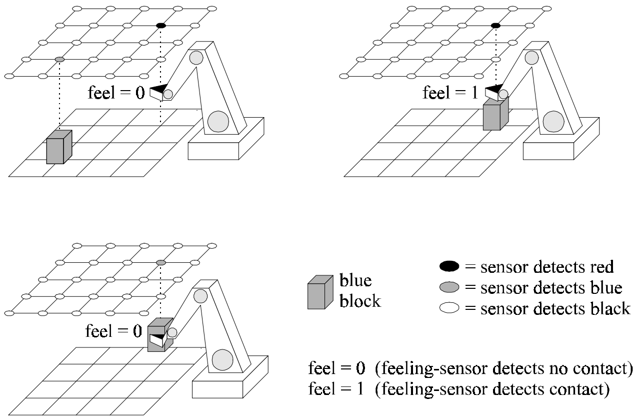

Fig. 10. : The Robot and its Environment (cf. Birk [1, p. 35])

3.1.2 The size of the knowledge

As in all Classifier Systems the knowledge is represented by its classifiers. The ACS has to generate in each position a classifier that moves in each direction with or without the block, onto the block or off the block. Furthermore, it needs a classifier that is able to grip and to release the block in each position (in case the block is under it). Table 1 shows the necessary raw classifiers for a 2×2 matrix for each of this possibilities, if *specification of unchanging components* is not used..

Table 1. Sorted representation of the classifier list that consists of the knowledge generated without further specialization for n=2 (r: red; b: blue; s: black; N: north; S: south; W: west; E: east; G: grip; R: release)

move onto the block	move off the block	move without contact to the block	transport the block	grip the block	release the block
(b#r#0,N,r#s#1)	(s#r#1,N,r#b#0)	(s#r##,N,r#s##)	(s#b##,N,b#s##)	(r###1,G,b###0)	(b###0,R,r###1)
(#b#r0,N,#r#s1)	(#s#r1,N,#r#b0)	(#s#r#,N,#r#s#)	(#s#b#,N,#b#s#)	(#r##1,G,#b##0)	(#b##0,R,#r##1)
(r#b#0,S,s#r#1)	(r#s#1,S,b#r#0)	(r#s##,S,s#r##)	(b#s##,S,s#b##)	(##r#1,G,##b#0)	(##b#0,R,##r#1)
(#r#b0,S,#s#r1)	(#r#s1,S,#b#r0)	(#r#s#,S,#s#r#)	(#b#s#,S,#s#b#)	(###r1,G,###b0)	(###b0,R,###r1)
(br##0,W,rs##1)	(sr##1,W,rb##0)	(sr###,W,rs###)	(sb###,W,bs###)		
(##br0,W,##rs1)	(##sr1,W,##rb0)	(##sr#,W,##rs#)	(##sb#,W,##bs#)		
(rb##0,E,sr##1)	(rs##1,E,br##0)	(rs###,E,sr###)	(bs###,E,sb###)		
(##rb0,E,##sr1)	(##rs1,E,##br0)	(##rs#,E,##sr#)	(##bs#,E,##sb#)		

Therefore the size of the knowledge is n^2 each for gripping and releasing, $4(n-2)^2 + 3 \cdot 4(n-2) + 2 \cdot 4 = 4n^2 - 4n$ for transporting the block, moving without contact to the block, moving off the block and moving onto the block. Because of the passthrough signs in ACS moving without contact to the block is independent from the actual position of the block and therefore has got the same size as moving off the block, moving onto the block or transporting the block. All in all that is

$4(4n^2 - 4n) + 2n^2 = 18n^2 - 16n$. Table 2 shows the size of the necessary knowledge for n=2 to 7.

Table 2. The number of different classifiers necessary for a complete knowledge

n=2	n=3	n=4	n=5	n=6	n=7
40	114	224	370	552	770

3.1.3 More specific knowledge

Considering the classifiers for transporting the block, one can realize that each of these classifiers can also match in a state where the arm and the block have no contact as each #-sign could either match with an 's' for black or with an 'r' for the robot arm. The same problem arises when looking at the classifiers for releasing the block. These classifiers have to be specialized by using *specification of unchanging components* to make sure that they are only applicable in the correct state.

Taking e.g. the first classifier for transportation in table 1: s#b## N b#s##, meaning that when transporting the block one step northwards the third component becomes black (s) and the first component becomes blue (b). It is applicable in the states ssbs0, srbs0 and ssbr0 but only successful in ssbs0, as in the other two states the robot is in a different position without the block. Once generated it will also be applied in the other states and will be marked there. Let us assume that the classifier has got a mark in state srbs0. Now it is applied again in the successful state ssbs0 and realizes that it has got the mark srbs0. That is why it will generate a new classifier that will explicitly carry an 's' (instead of the '#') in the second component to differentiate itself from the other state: ssb## N bss##. After setting another mark in ssbr0 and again being applied successfully the classifier will be ssbs# N bsss# and will only match the correct state from now on.

The classifiers for releasing the block have to be specialized in the same manner, as they can also be applied in states where the gripper is in a different position than the block. Furthermore, the classifiers for moving without contact to the block have to be specialized, too, i.e. in the last (the feeler) component, to make sure not to be over the block. Let's consider e.g. the first classifier for moving without contact to the block in table 1: s#r## N r#s##. It is applicable in the states sbrs0, ssrb0, ssrs1, but only correct in sbrs0 and ssrb0. So eventually it will be marked by ssrs1. Let's say it is successfully applied in sbrs0 again. Now it is possible to specialize the second or the last component. While the generated classifier that is specified in the second component matches from now on only in sbrs0, the generated classifier that is specified in the last component matches in both correct states and therefore is more useful than the other. Currently this is an unsolved problem in ACS. Despite this, eventually the classifier list will contain the classifiers shown in table 3, which represent a complete and unique knowledge of the environment.

Table 3. Sorted representation of the classifier list that consists of the complete and unique knowledge (after specification of unchanging components) for n=2 (r: red; b: blue; s: black; N: north; S: south; W: west; E: east; G: grip; R: release)

move onto the block	**move off the block**	**move without contact to the block**	**transport the block**	**grip the block**	**release the block**
(b#r#0,N,r#s#1)	(s#r#1,N,r#b#0)	**(s#r#0,N,r#s#0)**	**(ssbs#,N,bsss#)**	(r###1,G,b###0)	**(bsss0,R,rsss1)**
(#b#r0,N,#r#s1)	(#s#r1,N,#r#b0)	**(#s#r0,N,#r#s0)**	**(sssb#,N,sbss#)**	(#r##1,G,#b##0)	**(sbss0,R,srss1)**
(r#b#0,S,s#r#1)	(r#s#1,S,b#r#0)	**(r#s#0,S,s#r#0)**	**(bsss#,S,ssbs#)**	(##r#1,G,##b#0)	**(ssbs0,R,ssrs1)**
(#r#b0,S,#s#r1)	(#r#s1,S,#b#r0)	**(#r#s0,S,#s#r0)**	**(sbss#,S,sssb#)**	(###r1,G,###b0)	**(sssb0,R,sssr1)**
(br##0,W,rs##1)	(sr##1,W,rb##0)	**(sr##0,W,rs##0)**	**(sbss#,W,bsss#)**		
(##br0,W,##rs1)	(##sr1,W,##rb0)	**(##sr0,W,##rs0)**	**(sssb#,W,ssbs#)**		
(rb##0,E,sr##1)	(rs##1,E,br##0)	**(rs##0,E,sr##0)**	**(bsss#,E,sbss#)**		
(##rb0,E,##sr1)	(##rs1,E,##br0)	**(##rs0,E,##sr0)**	**(ssbs#,E,sssb#)**		

3.1.4Testing the knowledge

For monitoring purpose of the evolved knowledge a testing method of the currently achieved knowledge is necessary. The testing method is taking all the reliable classifiers and checks if they are capable to solve all the tasks mentioned in 3.1.2. Although it should be irrelevant where the block is situated in the skill *moving without contact to the block* all block positions are tested for making sure, that all correct and unique classifiers are formed (see problem in 3.1.3). Meaning that once achieved a 100% test result all possible goals will be reached successfully and even in an optimal number of steps. Therefore the size of the tested knowledge is $2n^2$ for gripping and releasing, $n^2(4n^2-4n)$ for all possible movements without the block considering all possible block positions and $4n^2-4n$ for all possible movements with the block resulting in $2n^2+(4n^2-4n)\cdot(n^2+1)$.

Note the difference between the growth of the different action tests. While the complexity of the actions for the possible movements without the block is $\in O(n^4)$, the complexity of the other, actual more difficult actions is $\in O(n^2)$. Table 4 shows the growth of the sizes.

Table 4. The size of all tested actions and in comparison the size of the possible actions with the block

n	**No. of all tested actions**	**No. of actions with the block**
2	48	16 (32%)
3	258	42 (16%)
4	848	80 (9.4%)
5	2130	130 (6.1%)
6	4512	192 (4.3%)
7	8498	266 (3.1%)

3.2 Goal-directed Learning

In the introduced environment the probabilities of the occurrences of each state are not equal. Transporting the block is only possible if the system tried to grip the block while over the block. This will occur quite seldom as the latent learning process is not rewarded anywhere and gripping in any other position than the block position will result in weakening the quality *q* of the grip-classifier. Transporting needs a successful grip before it can be executed so that it is still more unlikely. As it will be shown goal-directed learning is a possibility to decrease this effect.

3.2.1 The Exploration Phase and the Action Planning Phase

Action planning is done without any interactions with the environment (cf. section 2.3). This paper will not focus on the question which goal state is desirable but the goals will simply be generated externally by a so called goal-generator, that is part of the environment and not part of the ACS itself.

In the action planning phase an ACS requests a goal from the goal-generator. It tries to find the shortest possible path from the actual state to the goal by using the bi-directional search mechanism mentioned in 2.3. If it finds such a path, then it executes the classifiers of the path like in the exploration phase. If the execution was successful, then the ACS requests a new goal from the goal-generator. If the execution of any found sequence failed or if the search mechanism did not find a sequence at all, then the ACS leaves the action planning phase and reenters the normal exploration phase. During the exploration phase the ACS explores its environment, as described in Stolzmann [10,this volume, pp. ???-???]. Every 50 steps the action planning phase is taking place.

3.2.2 The application of goal-directed learning in the experiment

The goal-generator generates continuously the following goals:

1. Move to a randomly selected place, where the block is not situated.
2. Move to the block
3. Grip the block.
4. Release the block at a different randomly selected position.

The goal-generator and the switching between the action planning phase and the exploration phase is analogous to Birk (cf. [1, p. 61ff]).

Let's consider an example in which the ACS has already learned the complete world model (see table 3). Let's e.g. assume the following constellation of a 2x2 matrix: sbsr0, meaning the block is situated Northeast and the robot hand Southeast. Now the action planning phase starts. The goal generator in phase one creates e.g. the goal sbrs0. The search mechanism will find a sequence that consists of only one classifier: ##sr0 W ##rs0. After the execution of this classifier the environmental state will be sbrs0 and the ACS requests a new goal. The goal generator, now in phase two, will generate the goal srss1. The search mechanism generates the sequence: ##sr0 E ##rs0, #b#r0 N #r#s1 or s#r#0 N r#s#0, rb##0 E sr##1 and executes it. In phase three the goal is sbss0 and the sequence #r##1 G #b##0 is generated. After the execution the environmental state is sbss0 and the goal generator enters phase four and generates e.g. rsss1, that leads the ACS to the sequence sbss# W bsss#, bsss0 R rsss1. After the execution of these two classifiers the goal generator again enters phase one and the process starts from the beginning.

If the world model is not learned completely, then on the one hand a sequence might be longer than the optimal path and on the other hand the action planning process might fail. In this case the ACS reenters the exploration phase.

3.2.3 Advantage of goal-directed learning

Due to the permanent request of the goal-generator during phase 2, to move to the block and to grip it during phase 3, the system will be much more often in a position over the block and so experiences the results of gripping much more often. That is how the unequal distribution of probabilities between the different states shown in table 4 is partly neutralized. For there is a quadratic growth of the number of fields but the number of blocks stays the same (only one block) this advantage should increase with the size of the matrix.

3.3 Results

This section shows on the one hand the positive effect of adding goal-directed learning to ACS and on the other hand the overall performance of ACS in this simulation.

3.3.1 Comparing ACS in conjunction with and without goal-directed learning

Figure 11 shows the mean results of 100 simulations of the $n \times n$ matrix for n=2 and n=3 without and with goal-directed learning. The positive effect of goal-directed learning becomes obvious.

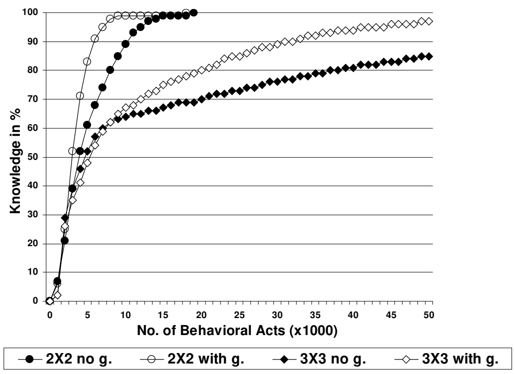

Fig. 11. Results of the simulation with n=2, 3 without and with goal-directed learning

Table 5 shows the same results in numeric form and the mean results of 40 simulations for n=4 without and with goal-directed learning.

Table 5. Average results for n=2-4 with and without goal-directed learning

	2x2	2x2	3x3	3x3	4x4	4x4

	no g.	with g.	no g.	with g.	no g.	with g.
>75%	7200	4400	27200	15000	14500	16000
>80%	8000	4800	36400	19400	37500	24000
>85%	9000	5200	48200	24000	97500	49000
>90%	10400	6000	62400	30800	187500	75000
>95%	11800	6800	82000	40800	320500	119000
>99%	15000	8200	114800	61800	>600000	301000

Obviously the ACS with goal-directed learning needs about half the behavioral steps of the one without goal-directed learning. The effect is still increasing for increasing numbers of n. This confirms the reflections in section 3.2.3.

The result that 75% of the 4x4 matrix are learned faster than 75% of the 3x3 matrix is an effect of the measurement of the knowledge. As explained in 3.1.4 the complexity of the possible movements without the block is $O(n^4)$ while the one with the block is $O(n^2)$. During the exploration phase the movements without the block are taking place most of the time so that most of the corresponding classifiers are learned very quickly. For n=4 this is 90.6% of the world model and for n=3 only 84%. The same effect explains the slightly better performance of ACS without goal-directed learning than the performance of ACS with goal-directed learning for the 4x4-matrix in the 75% results because an ACS without goal-directed learning is always in the exploration phase.

Figures 12 and 13 show the results of 100 simulations each of a 3x3 matrix without and with goal-directed learning in detail revealing the differences between the single experiments and again the positive effect of goal-directed learning.

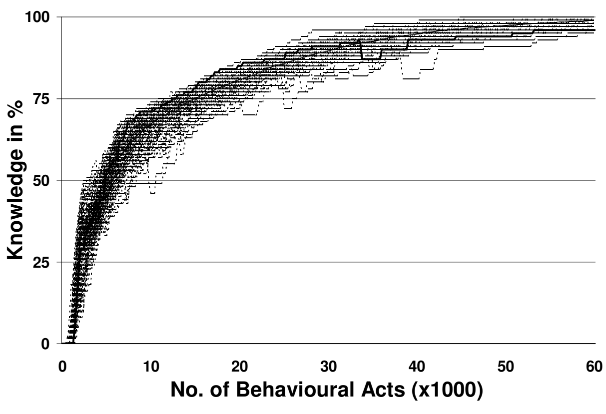

Fig. 12. 100 single experiments in a 3x3 matrix with goal-directed learning

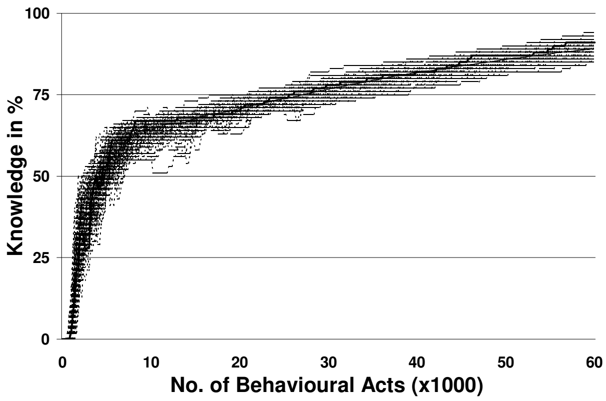

Fig. 13. 100 single experiments in a 3x3 matrix without goal-directed learning

3.3.2 More results with action planning

As shown in 3.3.1 the performance of ACS in conjunction with goal-directed learning is much better than without goal-directed learning. Therefore some more results for $n \times n$-matrixes with goal-directed learning are calculated. Table 6 shows the average performance for n=4 (for reference purposes), 5 and 6 with 10 experiments each and for n=7 one example experiment is given.

Table 6. The results for n= 4 to 7 in the learning constellation with goal-directed learning

	n=4	**n=5**	**n=6**	**n=7**
>75%	16000	16000	18000	18000
>80%	24000	23000	22000	24000
>85%	49000	42000	44000	30000
>90%	75000	99500	86000	54000
>92%	89000	141500	154000	80000
>94%	107500	194500	236000	158000
>96%	139000	276500	412000	428000
>98%	195500	479500	710000	1188000
>99%	261500	727500	1115000	1938000

3.4 Comparison with Birk's SRL

Birk [1] developed a learning algorithm called "Stimulus Response Learning" (SRL) to solve the presented experiment. In a situation S_2 the last behavior R and its starting situation S_1 are used to build a totally specialized rule $S_1 - R - S_2$. Such rules are generalized by using genetic algorithms. An ACS learns similar rules but it does not need genetic algorithms.

Analogous to Birk we were able to show the positive effect of goal-directed learning, although his experimental results seem to prove an even greater effect.

But first of all Birk only refers to the time his computer needed for the different tasks, so that a direct comparison is not possible. Moreover, his system is not creating simple rules but a graph whose edges are not trying to generate common results but indeed monitoring the effects completely. Last but not least the knowledge about the environment is not complete as in ACS. The junctions in SRL are too common. That is why a gripping action is possible to take place where no block is under the robot-arm.

In the current system the possibilities of planning are restricted (...) the system is normally not able to generate a successful plan for gripping a block [1, p. 71], own translation).

These restrictions are not given in ACS. Once the system has generated a complete world model it is not anymore limited and all possible tasks will be performed correctly.

4 Conclusions

This chapter showed that ACS can be used for learning robots. The performance of an ACS was examined when being applied to a mobile Khepera robot and a robot gripper that works in conjunction with a camera. Furthermore, ACS's ability of doing action planning was introduced and successfully applied.

Section 2 introduced an extended simulation of Seward's experiment about latent learning in rats. The experiment was firstly simulated by Riolo [6]. Stolzmann [8,9] discussed a first simulation with ACS. A comparison with Riolo's CFSC2 is given by Stolzmann [9]. The here introduced extension shows that Seward's experiment can be replicated with a mobile robot (in this case a Khepera robot) that learns by using ACS. Furthermore, in contrast to the earlier simulations, the ACS that controlled the mobile robot had to handle aliasing states.

In Section 3 the ability of an ACS to generate plans was shown. The experiment of a hand-eye coordination task simulated with a robot gripper and a camera proved that an ACS is able to take advantage out of generating plans to given goals, even while learning. It is able to learn much faster in environments where the probability of entering each state is not equal.

Although the goal generating process was part of the environment in this simulation, integrating the goal generator into the ACS should be possible. A possibility would be to add a temporary memory to the ACS out of which it could select states, that were e.g. experienced quite long ago. This approach can even be psychologically confirmed. In contrary to the known "joy of satisfaction", which could be compared to the normal latent learning in an ACS, a so called "joy of function" is observable especially in young children, when they are enjoying doing something over and over again [4]. Future research will show if such a self-goal-generating process can be successfully applied in the ACS.

Finally, it was shown that the marking mechanism does not always reveal the definite position for doing specification of unchanging components. In [2] this problem is discussed in further detail. To be able to generate always most general classifiers in an ACS this issue deserves future research, as well.

References

[1] Birk, Andreas (1995). *Stimulus Response Lernen*. Dissertation. University of Saarbruecken. Germany.

[2] Butz, M., Goldberg, D., & Stolzmann, W. (1999). New Challenges for an Anticipatory Classifier System: Some Hard Problems and Possible Solutions. IlliGAL Report No. 99019. Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign

[3] Croake, J. W. (1971). Unrewarded exploration and maze learning. *Psychological Reports*, 1971, 29, 1335-1340.

[4] Hoffmann, Joachim (1993). *Vorhersage und Erkenntnis*. Göttingen: Hogrefe.

[5] Michel, Olivier (1998). Webots - User Guide. Cyberbotics (*http://www.cyberbotics.com*)

[6] Riolo, Rick L. (1991). Lookahead planning and latent learning in a classifier system. In Meyer, J.A., and Wilson, S. W. (editors.). *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*. Cambridge, MA: The MIT Press. 316-326.

[7] Seward, John P. (1949). An Experimental Analysis of Latent Learning. *Journal of Experimental Psychology*. 39 177-186.

[8] Stolzmann, Wolfgang (1997). *Antizipative Classifier Systeme*- PhD Dissertation. Fachbereich Mathematik /Informatik. University of Osnabrueck. Aachen: Shaker Verlag.

[9] Stolzmann, Wolfgang (1998). Anticipatory Classifier Systems. In Koza, John R. et al. (editors). Genetic Programming 1998: *Proceedings of the Third Annual Conference, July 22-25, 1998, University of Wisconsin, Madison, Wisconsin.* San Francisco, CA: Morgan Kaufmann. 658-664.

[10] Stolzmann, Wolfgang (2000). An Introduction to Anticipatory Classifier Systems. In Lanzi, Stolzmann, Wilson *Learning Classifier Systems: An Introduction to Contemporary Research.* Pagg. 189-208. LNAI 1813. Springer, Berlin(This Volume)