Autonomous Fast Learning in a Mobile Robot*

Wolfgang Maass[1], Gerald Steinbauer[1,2], and Roland Koholka[1,3]

[1] Institute for Theoretical Computer Science
Technische Universität Graz, Austria
maass@igi.tu-graz.ac.at
[2] Roche Diagnostics, Site Graz
gerald.steinbauer@roche.com
[3] Knapp Logistik Automation GmbH, Graz
koholka@knapp.com

Abstract. We discuss a task for mobile robots that can only be solved by robots that are able to learn fast in a completely autonomous fashion. Furthermore we present technical details of a rather inexpensive robot that solves this task. Further details and videos of the robot are available from `http://www.igi.tugraz.at/maass/robotik/oskar`.

1 Introduction

We will discuss a task for an autonomous robot that *requires* learning insofar as a solution of this task by a non-learning robot is inconceivable (at least on the basis of budget constraints and currently available technology). On the other hand this task is also too difficult to be solved by a human. People can learn, but they do not have the mechanical skills which this task requires.

The task had been posed in the form of a student competition (Robotik 2000[1]) at the Technische Universität Graz. It can be outlined as follows. A 2×5 m white platform – surrounded by a black wall – had been divided by a black line into a *release zone* of about 1 m length and a *target zone* of about 4 m length (see Figure 1).

For each instance of the task one out of a large variety of green colored hills was placed at an arbitrary position – but at least 40 cm away from all walls – into the target zone. The hills were formed out of different kinds of hardening or non-hardening resins. These hills had a relatively rough surface and all kinds of odd shapes (diameters 30-60 cm), but they all had a round dip on the top of about 2 cm depth, with a diameters ranging from 8 to 12 cm, see Figure 2. The task was to accelerate and release a red billiard ball (diameter 5 cm, weight 142 g) in the release zone on the left part of the platform so that it comes to rest in the dip on top of the hill. To solve this task the ball has to be released with

* This research was partially supported by the Land Steiermark (Austria), project Nr. P12153-MAT of the FWF (Austria), and the NeuroCOLT project of the EC.

[1] detailed information about this competition can be found online under `http://www.igi.tugraz.at/maass/robotik`

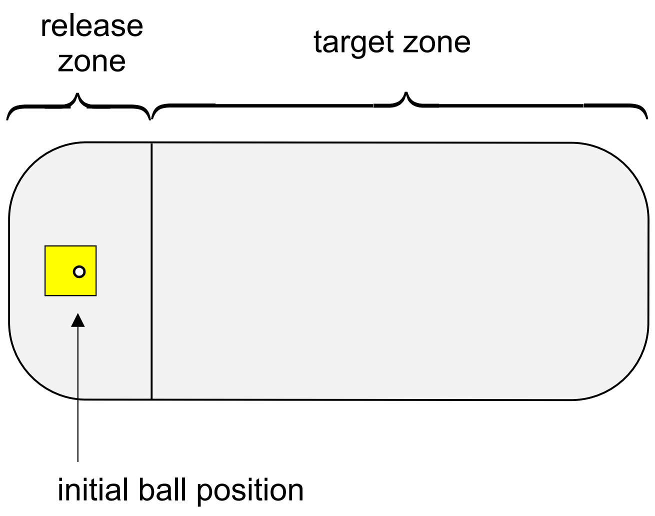

Fig. 1. Layout of the environment for the task.

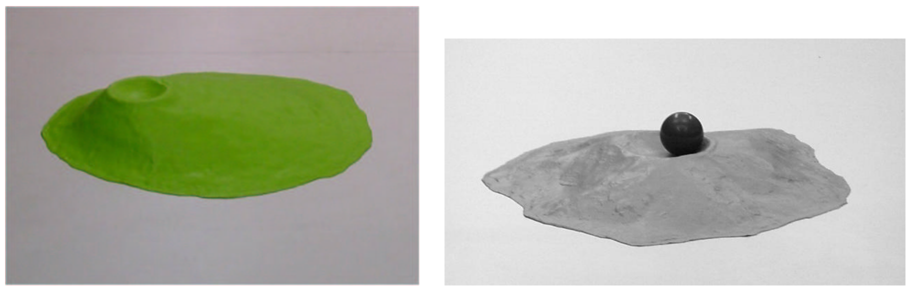

Fig. 2. Two of the hills that were used in the competition

just the right speed v and angle α. For most hill positions the set of parameters $\langle \alpha, v \rangle$ that solved the task was so small that even an experienced human typically needed 40 or more trials before the first success. The number of trials needed for a second successful shot was not significantly smaller, indicating that in spite of all trying a human can solve this task essentially just by chance.

After any unsuccessful trial the robots in the competition had to find the ball, move it back to the release zone, and initiate the next shot. All of this – just as the trials themselves – had to be done completely autonomously, using only on-board sensors, computing devices and power supply (the latter had to last for at least 45 minutes). The only signals or interventions from the outside that the robots were allowed to receive were the *start-signal* at the beginning (by pushing a button) when the ball had been placed on the initial ball position marked in Figure 1, and a *repeat-signal* after a successful trial. The repeat-signal was given by pushing another button on the robot, and it signaled to the robot that the ball had been removed by a person from the top of the hill and had been placed again on the initial ball position, whereas the hill and hill position had been left unchanged.[2] The performance measure for the competition was the total time needed by a robot until he had succeeded three times for the current hill and hill-position, averaged over a fair number of different hills and hill-positions.

A direct computation of the proper release values $\langle \alpha, v \rangle$ from measurements and other sensory data appears to be not feasible for an autonomous robot based on presently available technology.[3] Therefore a robot that was to be successful in this competition had no choice but to *learn* from his preceding trials in order to avoid a time consuming reliance on mere luck.

The winner of the competition was the robot Oskar[4] (see Fig. 3 and Fig. 4), designed and built by the second and the third author of this article. For technical

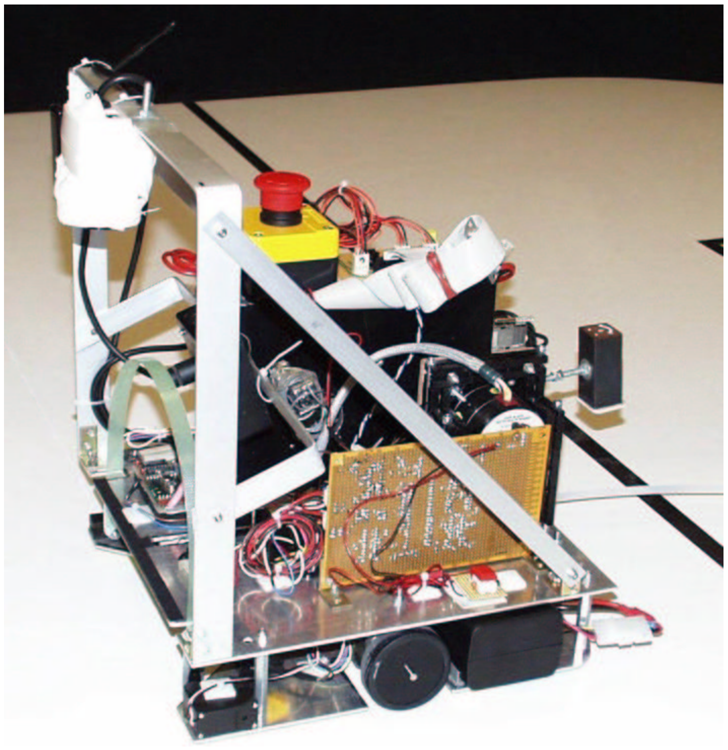

Fig. 3. This is the fast learning robot Oskar. The raised "lasso" for recapturing the ball is visible on his left. His hammer (visible on his right) has been pulled up, indicating that Oskar is about to shoot the ball from his ball-holding-bay underneath.

[2] We did not encourage that the robots themselves remove the ball from the top of the hill after a successful trial in order to protect our hills.

[3] More precisely: based on technology that can be afforded within the budget limit of 4000 Euro for each robot in the competition.

[4] see his homepage `http://www.igi.tugraz.at/maass/robotik/oskar`

details see Appendix B. Oskar usually needs 1-8 trials until the first successful shot, with a very high success rate for subsequent trials for the same hill in the same position. Oskar moves via two wheels powered by step motors, with two Castor wheels in the back. For the control architecture see Figure 7. After recapturing the ball Oskar repositions himself in the release zone with a precision of 1 mm and an angular precision of 0.2 degrees, using the visual markers on the field for localization. As one camera looks othogonally down onto the lines of the platform from 30 cm above, the accuracy of the repositioning is only limited by the mechanical precision. This accuracy in repositioning is essential in order to turn experience from preceding trials into useful advice for the next trial. Oskar accelerates the ball by hitting it with a hammer that can be pulled up to variable heights, thereby allowing control over the initial speed v of the ball. Oskar catches the ball by lowering a flexible "lasso". He transports the ball back into the bay in front of the hammer through a suitable body movement initiated by his two wheels ("hip swing"). Oskar receives sensory input from 2 on-board cameras and 3 photo sensors. Processing the input of both cameras requires the whole CPU time. Therefore the management of all actors was outsourced to a micro controler board, which communicates with the PC via serial interface. Fig. 7 shows a diagram of Oskar's control system.

Videos that show the performance of Oskar and competing robots can be downloaded from the homepage of the robot competition `http://www.igi.tugraz.at/maass/robotik/`.

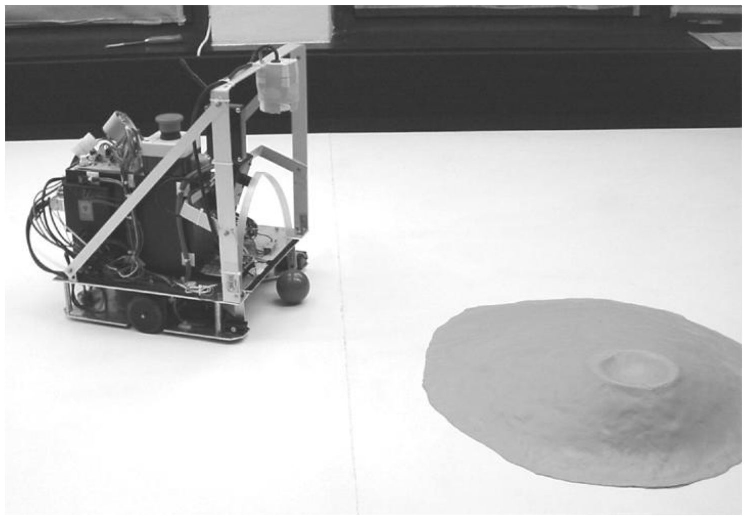

Fig. 4. The robot Oskar shown while recapturing the ball (which is visible just in front of the robot). The camera in the white housing suspended from the square metal frame is used for controlling the recapture operation. Oskar's main camera, which is used for localization, shot planning, and learning is visible close to the top of the raised "lasso".

2 Oskar's Learning Strategies

Oskar uses both a longterm- and a shortterm learning algorithm. In order to compute the initial speed v for a new instance of the task he uses *longterm learning* via a sigmoidal neural network (MLP) with 4 hidden units. The inputs to the neural network are the coordinates of the 4 corner points of a trapezoid (see Fig. 5) that approximates the segmented video image of the hill, recorded from the starting position.

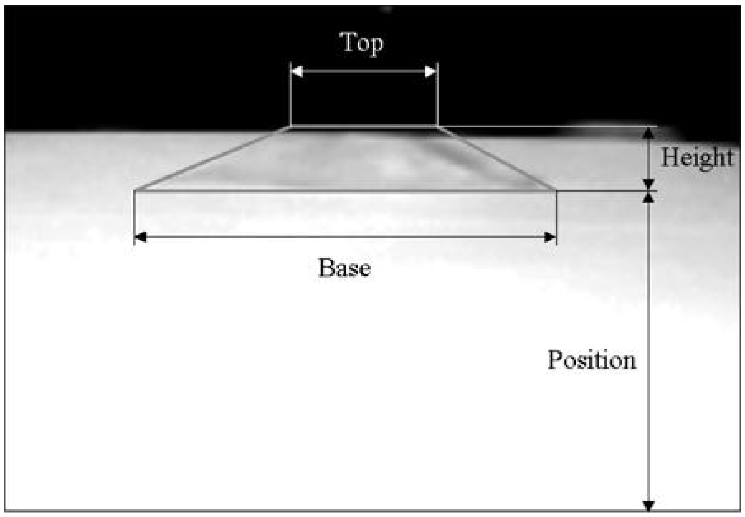

Fig. 5. A typical hill, as seen by the onboard camera of Oskar. Oskar approximates the hill by a trapezoid as shown and feeds its 4 corner points into a neural network that estimates a promising value for the speed v of the first shot.

The neural network is trained via backprop, with training examples provided by preceding successful trials. The neural network started to make useful predictions after it was trained with data from 60 successful trials (error on training set: 1.2%). The other parameter α for the first trial on a new hill is simply computed by aiming at the center of the upper horizontal line segment ("Top") in the trapezoid that approximates the hill.

Since the first trial of Oskar is likely to succeed only for very simple hills (e.g., for small hills with a fairly deep dip at the top), the key point of his operation is his capability to learn via *shortterm learning* autonomously from preceding unsuccessful trials for the same instance of the task. For that purpose Oskar records for each shot the trajectory of the ball, and extrapolates from it an estimated sequence of positions of the *center* of the ball (see Fig. 6 for

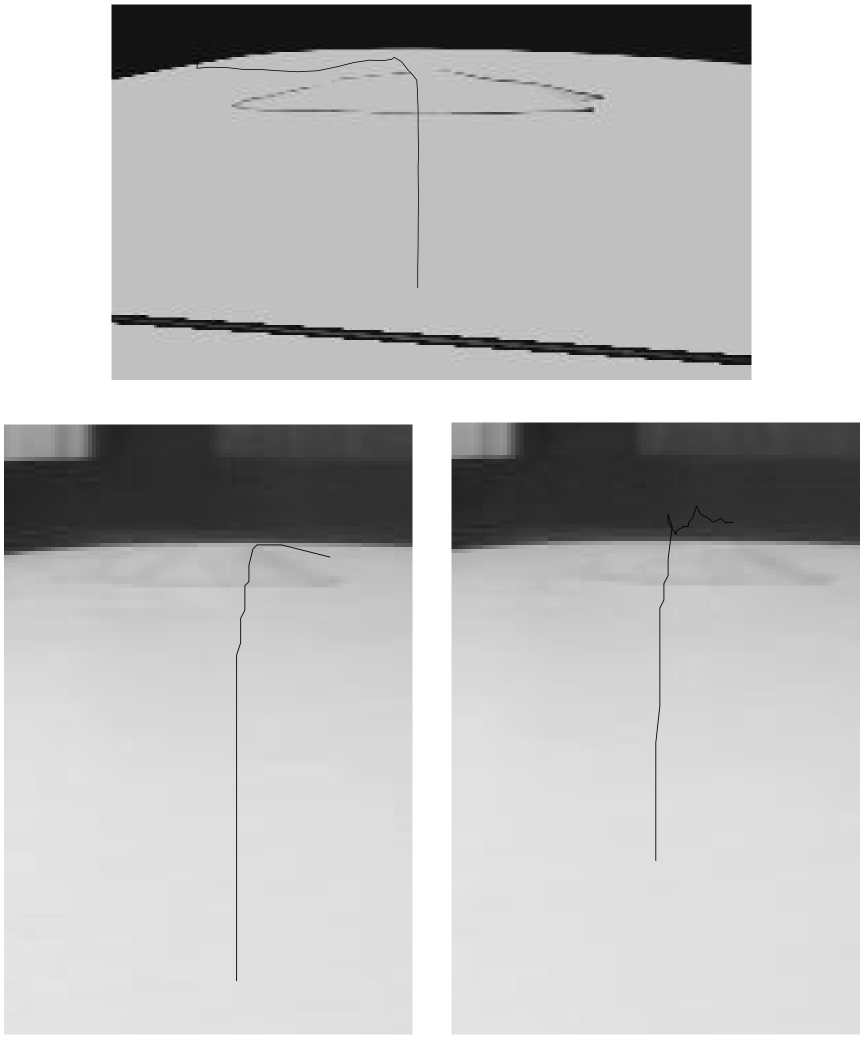

Fig. 6. Typical trajectories of the (estimated) center of the ball for unsuccessful trials, as computed by Oskar from his video recording. The trajectories shown were classified by Oskar into the classes 1,2, and 3. The characteristic feature of a trajectory in class 3 is that it goes up, down, and up again in the immediate vicinity of the hill.

a few examples).[5] This sequence of points is placed into one of 6 classes that characterize specific types of failures for a shot:

1. ball went too much to the right

[5] Oskar records 12 frames per second, yielding typically 3 to 25 frames for the period when the ball is on the hill.

2. ball went too much to the left
3. ball reached the dip, but went through it
4. ball rolled back from the hill
5. success
6. anything not fitting into classes 1-5

If the trial was unsuccessful, Oskar computes with the help of his learning algorithm QUICK-LEARN the parameters $\langle \alpha, v \rangle$ for the next trial from the classifications of the ball trajectories for the last 4 shots and a quantitative measure for their deviations to the left or right.

There are no theoretical results known to us which could guide the design of such learning algorithm. There exists some theoretical work on binary search with errors (see for example [1]; pointers to more recent work can be found in [2]). However these studies focus on a worst case analysis under the assumption that the total number of erroneous answers lies below some known bound. In contrast to that, we need here a learning algorithm that converges very fast *on average* in the presence of stochastically distributed errors. Furthermore in our case the probability that an answer is erroneous is highly non-uniform: it is quite large if one is already close to the target, but it drops quickly with the distance between the current query and the unknown target value. To the best of our knowledge, no theoretical analysis of binary search with errors is known that can be applied to this more realistic case. In addition, our target vector lies in a *2-dimensional* space. Since it can be shown empirically that for most instances of the task the solution set does not form a rectangle in this 2-dimensional space, the search problem cannot be reduced to two independend binary search procedures.

The algorithm QUICK-LEARN merges two simultaneous binary searches, for the right angle α and the right speed v, where each of these binary search procedures receives just partially reliable feedback through the classification of previous trials into the classes 1 to 6. The algorithm QUICK-LEARN proceeds in all cases except one on the assumption that the parameters α and v can be determined independently.

The parameters α and v are left unchanged if the last trial was classified into class 6. Classifications of failures into the classes 1 and 2 are used as feedback for the binary search for α, and classifications into the classes 3 and 4 are interpreted as feedback for the binary search for v. This feedback assignment is only partially justified, since for example a ball that is a little bit too fast and a little bit too far to the right frequently runs along the distant inner rim of the dip on the hill and rolls back from the *left* side of the hill. Such misinterpretation, along with various other sources of noise and imprecision, causes errors in the binary search procedures for α and v.

When a trial yields for the first time a trajectory belonging to a class in $\{1, 2\}$ or $\{3, 4\}$, the corresponding parameter α or v is changed by some minimal step size in the obvious direction. Special precautions are built into QUICK-LEARN that allow it to recover quickly from erroneous feedback. The step size of the relevant binary search procedure is halved - like in the classical binary search procedure - only if the last two trials have produced trajectories that belong to classes that reflect complementary errors for the parameter in question, like the classes 1 and 2 for α, or the classes 3 and 4 for v. Whenever the trajectory switches between the groups of classes $\{1,2\}$ and $\{3,4\}$, the binary search is switched to the other parameter (with a step size equal to the last value when that other binary search had been interrupted). The step size for binary search is left unchanged if the trajectories for the last 2 or 3 trials all fall into the same class. If the last 4 trials have produced trajectories that all fall into the same class in $\{1,2,3,4\}$, the corresponding parameter is not only moved into the appropriate direction, but in addition the step size is doubled. Furthermore after 4 or more trials that yielded trajectories in one of the classes 3 or 4, not only the speed v is changed, but also the angle α is changed by some minimal step size in the direction which is suggested by a quantitative measure `dev` that evaluates left/right deviations of the ball that were caused by the hill (for details see the pseudocode of the algorithm QUICK-LEARN given in Appendix A).

3 Discussion

In contrast to many other applications of machine learning in robotics, where robots learn to avoid obstacles or other functions that can also be solved without learning, the robot Oskar that we have discussed in this article is the result of an evolutionary challenge where learning was indispensable to solve the given task. This given task happened to be one which is too difficult for humans in spite of their learning skills, because they are inferior to robots with regard to their precision and reproducibility of movements. Hence one may argue that Oskar solves a task that can neither be solved by non-learning robots nor by humans, thereby demonstrating the tremendous potential of autonomous adaptive robots.

On a more abstract level, the task that is solved by Oskar requires a very fast search in a multi-dimensional space for a parameter vector that provides a solution of the current instance of the task. This search has to be carried out without any human supervision, by computing autonomously suitable feedback for learning from the sensory input of the robot. Tasks of a similar nature arise in many application areas where one needs to procede by trial und error, for example in the area of construction, repair, or rescue. Many of these tasks arise in environments that are inaccessible or too hostile for humans, and therefore are interesting application areas for robotics. In the area of medical robotics miniature robots that learn fast to solve a task of skill in an autonomous fashion may potentially be useful for carrying out surgical procedures inside a human body that are so delicate that human supervision is too unreliable.

The robot Oskar that we have discussed in this article has become one of the longest serving autonomous adaptive robots that are known to exist. He has been running continuously 9 hours per day (for permanently changing hills and hill-positions) during the 6 months of a major exhibition[6] in Graz (Austria), during which Oskar delivered 31.108 shots (6.127 of which were successful). Hence this

[6] Steiermärkische Landesausstellung 2000, entitled comm.gr2000az, curated by H. Konrad and R. Kriesche. Oskar is scheduled to move in the summer of 2001

robot demonstrates that autonomous learning capabilities can be implemented in a mobile robot in a stable and reliable manner, at a very low cost.[7]

4 Acknowledgement:

We would like to thank Prof. A. Pinz, Prof. R. Braunstingl, Prof. G. Wiesspeiner, Dr. Thomas Natschläger, Oliver Friedl, and Dipl.-Ing. Harald Burgsteiner for their helpful advice.

References

1. Rivest, R. L., Meyer, A. R., Kleitman, D. J., and Winklmann, K.: Coping with errors in binary search procedures. Journal of Computer and System Sciences 20 (1980) 396–404
2. Sereno, M.:Binary search with errors and variable cost queries. Information Processing Letters 68 (1998) 261–270

Appendix A: Pseudocode for the Learning Algorithm QUICK-LEARN

Definitions:

`last`:	array with the classifications for the last 4 trajectories
`last[0]`:	classification of the immediately preceding trajectory
`dev`:	horizontal deviation at the hill of the last trajectory[1]
`delta_a`:	current step size in binary search for angle (initial value = 1/6 of the width of the hill at the base)
`delta_v`:	current step size in binary search for velocity (initial value = 1/20 of the value suggested by the neural network)
`a`:	parameter angle
`v`:	parameter velocity
`a_min`:	minimal step size in search for angle[2]
`v_min`:	minimal step size in search for velocity[3]

`delta_a` is never allowed to assume values less than `a_min`.
`delta_v` is never allowed to assume values less than `v_min`.

to the permanent collection of the Ars Electronica Center in Linz (Austria), see `http://www.aec.at`.

[7] The total cost of the components of Oskar lies below 3000 Euro. The only replacement parts that were needed during his 6 months of continuous performance was a new set of tires.

```
QUICK_LEARN(classes last[4], dev)
switch(last[0])
{
class_1:
if (all last == class_1)      // last 4 trajectories too far right:
delta_a *= 2                  // double step size for angle
else

if (last[1] == class_2)       // last 2 trajectories in classes 1
delta_a /= 2                  // and 2: halve step size for angle
delta_a = max(delta_a, a_min) // step size not smaller than minimum
                              // step size

if (last[1] == class_5)       // failure (of class 1) after success
a -= a_min[4]                 // (class 5): minimal change for
                              // angle
else
a -= delta_a

class_2:
if (all last == class_2)      // last 4 trajectories too far left:
delta_a *= 2                  // double step size for angle
else

if (last[1] == class_1)       // last 2 trajectories in classes 1 and 2:
delta_a /= 2                  // halve step size for angle
delta_a = max(delta_a, a_min) // stepsize not smaller than minimum step size

if (last[1] == class_5)       // failure (of class 2) after success:
a += a_min                    // minimal change for angle
else
a += delta_a

class_3:
if (all last == class_3)      // last 4 shots too fast (class 3):
a += sign(dev) * a_min        // additional minimal change of angle[5]

if (last[1] == class_4)       // last 2 trajectories in classes 3 and 4:
delta_v /= 2                  // halve step size for velocity
delta_v = max(delta_v, v_min) // step size not smaller than minimum step size

if (last[1] == class_5)       // failure (of class 3) after success:
v -= v_min                    // minimal change of velocity
else
v -= delta_v                  // for all other classes: change with stepsize

class_4:
```

```
if (all last == class_4)       // last 4 shots too slow (class4):
a += sign(dev) * a_min         // additional minimal change of angle[5]

if (last[1] == class_3)        // last 2 trajectories in classes 3 and 4:
delta_v /= 2                   // halve step size for velocity
delta_v = max(delta_v, v_min)  // step size not smaller than minimum step size

if (last[1] == class_5)        // failure (of class 4) after success:
v += v_min                     // minimal change of velocity

else
v += delta_v                   // for all other classes:
                               // change with step size

class_5:
a = a                          // last shot was successful:
v = v                          // leave parameters unchanged
store_example                  // save angle, velocity and hill
                               // features as training example
                               // for the longterm neural
                               // network learning algorithm

class_6:
a = a                          // leave parameters unchanged,
v = v                          // no information extracted

}
```

Remarks:

[1] dev is the average of the horizontal deviations of points on the trajectory while the ball was on the hill, from a linear extrapolation of the initial segment of the trajectory defined by the points of the trajectory just before the ball reached the hill. Thus dev essentially measures the horizontal deviation of the ball caused by the hill.
[2] The minimum step size for the angle is one step of one of the two driving motors. The direction of the change determines which motor has to move.
[3] The minimal step size for the velocity is one step of the stepping motor that pulls up the hammer.
[4] Angles and angle changes to the left are counted negativ, angles and angle changes to the right are counted positiv. Angle 0 is defined by the straight line to the middle of the top of the hill (see Fig. 5).
[5] This rule prevents the algorithm to get stuck in just changing the velocity. Because of this rule one cannot view the algorithm as an implementation of 2 independent binary search procedures.

Appendix B: Technical Data of the Learning Robot "Oskar"

Processor board
Single-Board-PC 586LCD/S by Inside Technology AMD 586 @ 133 MHz, 32 MB Ram, ISA and PC 104 Bus runs under DOS and the DOS-Extender DOS4GW by Tenberry
2,5 Laptop-Harddisc with 400 MB memory

Image processing
b/w-framegrabber pcGrabber-2plus by phytec with four camera inputs, ISA Bus color-CCD-camera VCAM 003 by phytec for ball/hill-analysis, no-name-wide-angle b/w-camera for navigation and obstacle avoidance

Motor/servo-steering
Risc-Microcontroller AT90S8515 by Atmel,
8kB integrated flash-program-memory
running under operating system AvrX by Larry Barello

Driving
two 1,8° stepping motors in differential drive

Shooting
one 0,9° stepping motor + hammer (250g)

Power Supply
High Performance NiMH-Accu with 2,6Ah capacity and 12V. Two DC/DC Converters provides the also needed 5V and -12V.

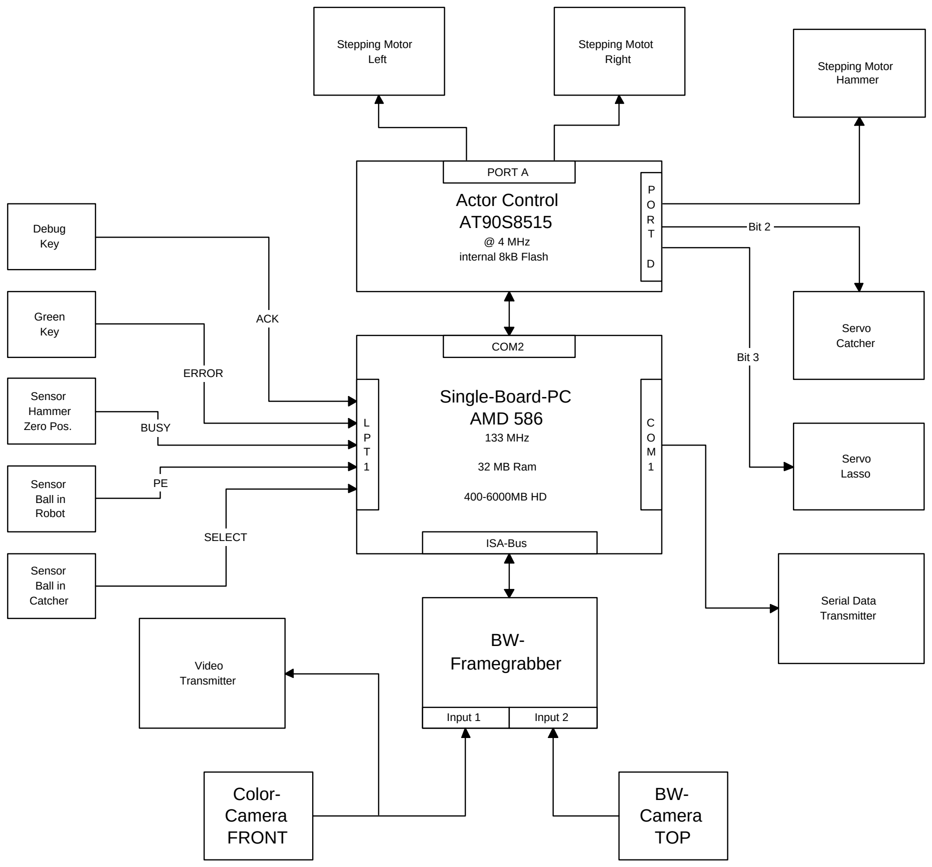

Fig. 7. Block diagram of Oskar's control system. It was designed to be simple and inexpensive. The main part is a low power Single Board PC. All control tasks except actor control are implemented on this PC. The actor control was sourced out to a microcontroller, to release the PC. The main sensors are two cameras connected to a 4-channel frame grabber on the ISA-Bus. To avoid an extra I/O-Module, the 3 photo sensors are read in via the parallel port.