Learning Real-Time Strategies for Binocular Vergence

Justus Piater, Krithi Ramamritham, Roderic Grupen

CMPSCI Technical Report 99-06

February 1999

Computer Science Department
Lederle Graduate Research Center
University of Massachusetts
Amherst, MA 01003-4601

{piater|krithi|grupen}@cs.umass.edu

Learning Real-Time Strategies for Binocular Vergence

Justus Piater, Krithi Ramamritham, Roderic Grupen

Abstract— This report presents a method for learning real-time strategies to verge a stereo pair of cameras such that their image centers display the same world feature. The method makes use of a foveal image representation in which image resolution decreases logarithmically from the center towards the periphery. This representation performs a significant image data reduction that reduces the computational cost of stereo matching, and is easily computed at frame rate on a standard PC platform.

The algorithm is applied to vergence control in a 2-DOF stereo pair. The resulting vergence controller is envisioned as one of several control options available for generic, sensorimotor problem solving. These options compete for resources and are therefore subject to a scheduler. The scheduler has access to *a-priori* performance estimates as a function of the compute time available. For this setting, a Q-Learning framework is defined for learning appropriate vergence strategies that depend on the number of perception-action cycles available. The learned strategies are characterized in terms of probability of success and accuracy.

This paper integrates issues from computer vision, real-time systems, and machine learning.

1 Introduction

In an integrated system that interacts with the real world, resources are constrained, and components of this system compete for them. The most apparent such resource is time: The world behaves according to its own characteristics, and an agent interacting with it must act and respond within appropriate time frames. This places constraints on the amount of time available to individual subcomponents of the agent. Often performance characteristics of such a component depend on the resources available to it. A common example is a speed/accuracy tradeoff: If time allows, more computation can be expended or more servo cycles can be run to improve the result. These dependencies can often be formally characterized. The allotment of time slices to the individual components, based on such performance profiles, is the task of a scheduler. Based on the time allotted to it, each component must select an appropriate set of actions and should adhere to the schedule and performance profiles in a predictable way.

Figure 1 shows a general real-time scenario consisting of several components, each of which can perform a specific task. Each task utilizes a specific set of resources, which may or may not be sharable. Each task has certain performance characteristics associated with it, which may depend on the resources available, as well as external factors.

There are several ways to make progress toward the goal, which are characterized by various sequences (or simultaneous executions) of tasks. Each such sequence has certain performance characteristics. In this paper, it is the job of the scheduler to schedule the tasks appropriately such that progress toward the goal is made. To support scheduling decisions, it can inquire about each task's performance characteristics, given a specific situation.

This paper focuses on one of these tasks and its performance characteristics as they are relevant to the scheduler. The task is to verge a stereo camera system such that the image centers of both cameras display the same world feature. This is typical of many robotic control problems in that the result depends on the available resources, and this dependency is not deterministic and can at best be described in probabilistic terms. This work contributes a vergence algorithm that provides an estimate of the reliability and accuracy achieved at any point in time. A probabilistic characterization of the vergence task in terms of these estimates can help a scheduler allocate resources appropriately.

The technical details of the vergence task are described in the following section. Section 3 discusses the task and its role in a real-time framework as

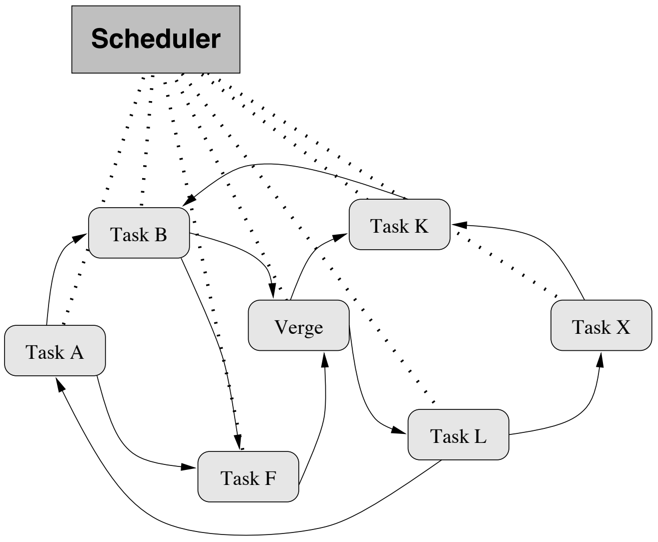

Figure 1: Real-Time scenario. Several tasks, coordinated by the scheduler, cooperate to accomplish an overall goal. Constraints on the sequence of tasks are indicated by arrows. The vergence system discussed in this paper constitutes one of these tasks.

outlined above. Section 4 introduces a machine learning setup for learning time-constrained vergence strategies, whose performance characteristics are then analyzed in Section 5.

2 On-Line Binocular Vergence

In this work, vergence involves turning both cameras symmetrically such that their image centers display the same point in the world (Figure 2). Figure 3 depicts the hardware setup. A pair of cameras mounted on a pan/tilt/verge platform provides live video to the Pipeline Video Processor. Controlled by the Host Computer, this grabs a pair of video frames and preprocesses it to reduce the amount of data. The Host Computer then computes a stereo match on the compact image representations, and determines an appropriate symmetric vergence adjustment angle $\Delta\theta$. This adjustment angle is passed to the Motor Controller, which computes the joint-space trajectories and motor torques applied to the camera actuators.

Like other typical computer vision systems, our setup is based on video cameras that deliver 512×480 pixel frames at 30Hz. The resulting immense

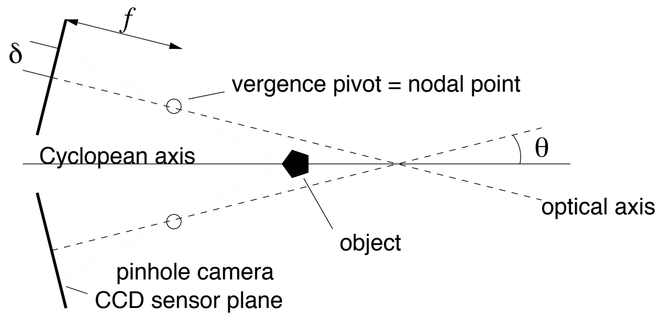

Figure 2: Symmetric vergence geometry.

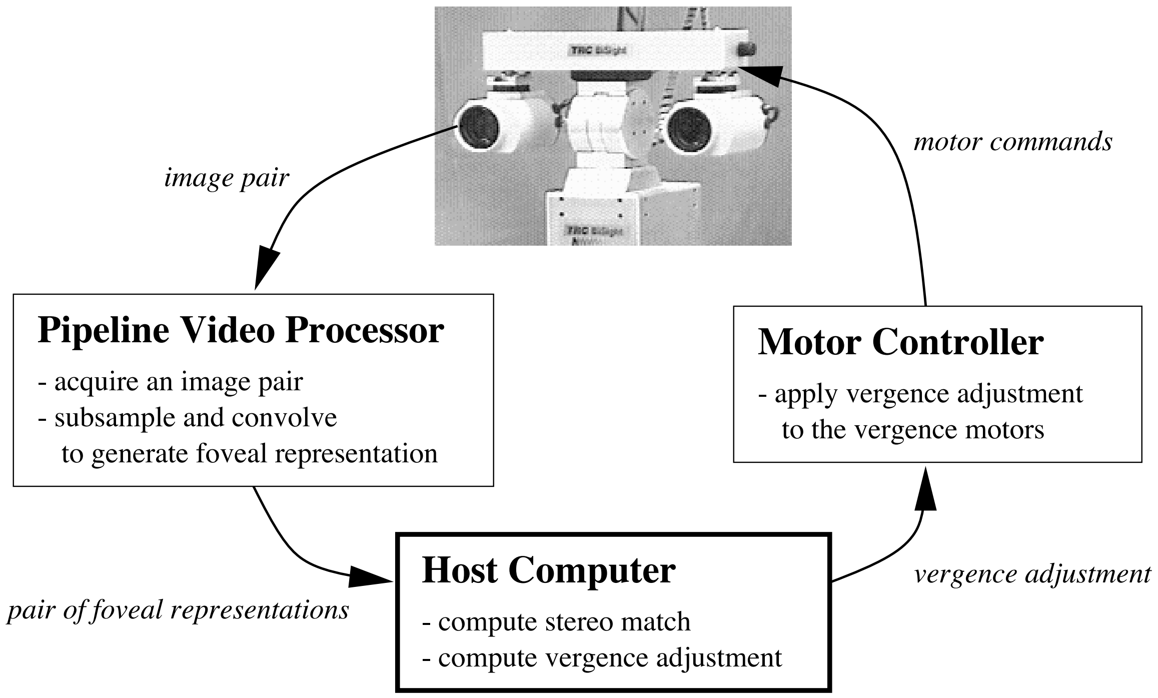

Figure 3: Simplified structure of the vergence task. The Host Computer constitutes the schedulable resource; the other three components consist of dedicated hardware.

data rate of 7372800 values per second is more than can be exploited by today's computing systems. Therefore, a tradeoff must be made between the amount of data processed per frame, and the number of frames processed per second. It is becoming increasingly popular to address this problem by using a foveal image representation, with high resolution near the image center and decreasing resolution towards the periphery. Various such schemes have been proposed (Weiman 1994; C. Capurro and Sandini 1997; Young et al. 1998). We generate a multi-resolution representation by successively subsampling the image to half the (linear) resolution, and keeping the highest resolution only in the center of the image. The resulting arrangement of the image is illustrated in Figure 4.

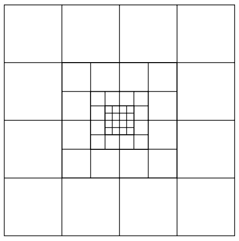

Figure 4: Logarithmic subdivision of an image using 4 levels of resolution. Each level is represented as a 4×4 block. Each square within each block represents a vector of 6 features, which are the responses of 6 different oriented Gaussian-derivative filters applied at this location.

Due to the low resolution in the periphery, little information is available for stereo matching between images. To compensate for this, we do not simply retain a single gray value at each location, but a vector of 6 expressive features obtained by a local convolution with one of 6 oriented Gaussian-derivative filters (Rao and Ballard 1995). The subsampling and the convolutions are performed well within a frame time on the pipleline video processor.

The resulting foveal representations of the stereo image pair are passed to the host computer. Here, a symmetric column-wise stereo match is performed as illustrated in Figure 5. The result is given by the best matching column pair, which is characterized as the signed distance p (in full-resolution pixels) of the column center to the image center. Then, an appropriate motor command is generated to turn the two cameras by an amount of $\Delta\theta = \Theta(p)$ so as to bring the best-matching column into the center of both cameras, i.e. to drive p closer to zero. The motor command is executed by the motor controller. On completion, the next image pair is acquired and preprocessed by the video processor. This procedure from image acquisition to completion of the mechanical adjustment constitutes a full *perception-action cycle*.

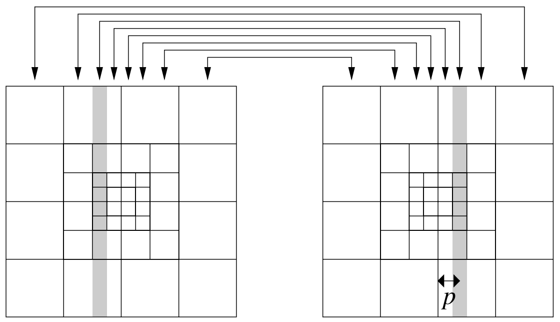

Figure 5: Illustration of the symmetric matching process (here shown for 3 levels of resolution). The shaded areas indicate one pair of matched columns at the highest resolution. A feature vector is constructed as the concatenation of all feature vectors associated with all squares at all levels of resolution that overlap the shaded area. The match score of such a column pair is given by the normalized cross correlation between the corresponding feature vectors. All symmetric pairs of columns are matched, as indicated by the arrows. The result of this match is given by the column pair with the highest match score.

3 Real-Time Issues

For the purpose of this study, the schedulable resource is time on the host computer (cf. Figure 3). The vergence system requires the host computer to compute the stereo match and the corresponding vergence command $\Delta\theta$. This process is then idle while the motor command is executed by the controller, and until the next image pair is acquired and has been processed by the pipeline video processor. This procedure can be iterated to improve the result. If more cycles can be executed, the accuracy and reliability of the resulting verge will generally increase. Thus, the vergence task consists of a potentially unlimited number of subtasks O_i, each of which is optional. The relative importance of each task O_i decreases as i increases. This is an instance of an *imprecise computation* (Liu et al. 1991) without any mandatory subtasks.

Each task O_i becomes ready after O_{i-1} has completed, and the next preprocessed image pair is available from the pipeline processor. The image acquisition and preprocessing process is fully deterministic and takes a fixed time to execute. The same is true for the computation of $\Delta\theta$, if the host computer is running a real-time operating system. However, the execution of the motor command is not deterministic, since the mechanical plant interacts with a nondeterministic physical world. This presents a challenge to a real-time scheduler, since the precise ready times of future subtasks O_i are not known in advance.

However, in reality there is relatively little variability in the duration of a perception-action cycle. A typical distribution of durations of cycles is plotted in Figure 6, and minimum and maximum durations are listed in Table 1. It is apparent from Figure 6 that there is a slight tendency for larger adjustments to take longer, but there is no fixed relationship between the two. While individual durations can be quite long, the total vergence task turns out to be reasonably robust with respect to a reduced number of perception-action cycles, as will be shown later. Therefore, a practical solution would be to schedule a fixed number of cycles beginning at regular intervals of about 0.7 seconds. Then, if the motor command following O_i has not completed when O_{i+1} is scheduled, subtask O_{i+1} can simply be dropped, reducing the total number of perception-action cycles by one. Thus, the vergence task constitutes an *interruptible anytime algorithm* as described by Zilberstein (1996). In the remainder of the paper, the scheduled time is expressed in terms of the number of available perception-action cycles.

$\Delta\theta$	0.1	0.2	0.5	1.0	2.0	5.0
min	0.614	0.614	0.631	0.628	0.618	0.631
max	1.635	1.740	1.233	1.230	0.899	1.234

Table 1: Table of practically occurring minimum and maximum durations of perception-action cycles in seconds, separated by the magnitude of vergence adjustments.

If the number of remaining perception-action cycles is known in advance to the system, it may choose $\Theta(p)$ depending on this number. For instance, it may prove useful to apply conservative $\Delta\theta$ in the presence of noise, if there are additional perception-action cycles left to bring the cameras fully into their desired position. It is not trivial to specify an optimal $\Theta(p)$ in this sense. This research contributes a machine learning approach that learns to maximize the expected vergence accuracy, given the number of remaining

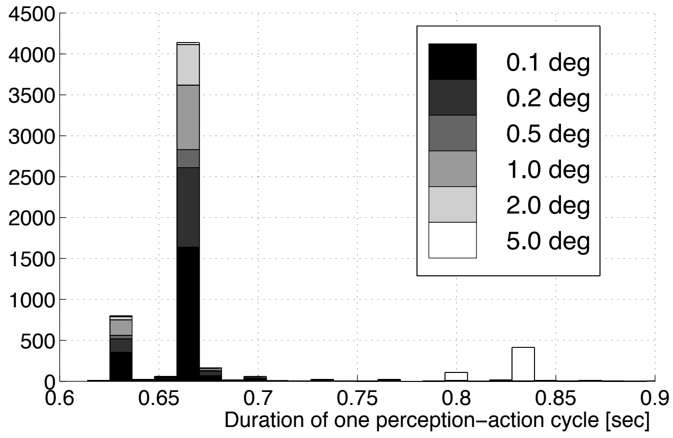

Figure 6: Histogram of practically occurring durations of perception-action cycles, colored by the magnitude of vergence adjustments in degrees. The apparent concentration at multiples of 33 msec is due to the fact that perception-action cycles always begin at a discrete frame time.

perception-action cycles.

As outlined in the introduction, our vergence task is nonstandard among real-time tasks, but typical of many robotics tasks in several ways because a mechanical plant is involved which is subject to perturbations. Therefore, no hard Quality-of-Service guarantees can be made for the system. The quality of a verge achievable within a given amount of time can at best be characterized in probabilistic terms. For a given verge, this quality is given in terms of the final magnitude of $|p|$ remaining after the task is completed.

This quality estimate is identical to the value optimized by the vergence controller. It estimates the accuracy of the achieved verge, given that the stereo match is correct. It makes no statement about the reliability of the stereo match underlying the verge. A binary estimate of this reliability is given by the index of the second best matching column p' (cf. Figure 5): A stereo match is considered reliable if the columns of p' and p are adjacent. The intuition behind this is that due to the wide spatial support of the convolution kernels that produce the features used for matching, a good match should extend over relatively wide areas. This is discussed further in Section 4.

The performance of a given vergence strategy can be characterized in terms of the distributions of these two measures achieved in practice. This information can be exploited by a scheduler in order to allocate an appropriate number of perception-action cycles under given performance requirements and overall constraints. When a particular task O_i is activated, it should receive as a parameter the number of cycles remaining after its completion. It can then choose that $\Theta(p)$ which yields the best expected performance.

4 Learning Real-Time Vergence

The central task addressed in this work is the specification of a function $\Theta(p)$ that maps a column index p to the vergence adjustment $\Delta\theta$. To determine this function is a typical system identification problem that can technically be solved to high accuracy by calibration techniques (see Section 5.2). On the other hand, a $\Delta\theta$ that is optimal in some appropriate sense may in fact depend on factors other than p. For instance, in the presence of measurement noise it may be better to use conservative vergence adjustments, i.e. undershoot, rather than attempting to foveate by a single adjustment. This will reduce jerky moves in response to outliers. If more than one perception-action cycle is available for vergence, it is still possible to achieve good final accuracy.

The traditional approach employs a closed-loop PD control model (Hansen and Sommer 1996). The gains of such a controller are tuned in a simulated environment to achieve robust and fast performance in the presence of simulated noise. However, one can do better than that. Note that the match uncertainty arises from self-similar scenes where several good matches compete, or more commonly, from poorly structured scenes that do not provide enough information to yield a reliable match, or from highly non-stationary scenes that degrade the predictable relationship between one image pair and the next after a vergence angle adjustment. If the reliability of a particular computation of p can be estimated, $\Delta\theta$ should clearly depend on this estimate.

As it turns out, such an estimate is provided by the index p' of the second-best matching pair of columns: If and only if p and p' are neighbors (i.e. the best-matching and second-best-matching columns are adjacent), then p can be taken as highly reliable. The reason for this is that due to the smoothing characteristic of the convolutions we applied, the correlations between a stereo image pair generally vary smoothly with location.

Consequently, a function $\Theta(p, p', s_r)$, where s_r is the number of perception-action cycles remaining, should be superior to a function $\Theta(p)$ in two ways: It should be less susceptible to noise, and it provides a running estimate of the reliability of the achieved result. The function $\Theta_f(p, p', s_r)$ is learned in a reinforcement learning (Sutton and Barto 1998) framework, which is described below.

The performance of the learned strategies can be evaluated by running them many times and gathering statistics. The variables of interest to the scheduler are the probability of success (defined as p and p' being neighbors, which is justified in Section 5.1), and the accuracy distribution given successful vergence.

4.1 A Reinforcement Learning Problem

The problem of learning $\Theta_f(p, p', s_r)$ is expressed in the Q-Learning framework (Watkins 1989), which learns values of actions taken in particular states of a Markov Decision Process. The value of an action a taken in a state s is denoted $Q(s, a)$.

The state space has three discrete dimensions: $|p|$ (0 is the center of the visual field), $\bar{p}$ which is a binary variable indicating whether p and p' are neighbors, and $s_r \in \{0, 1, \ldots, 4\}$, which indicates the number of perception-action cycles remaining after completion of the current cycle. The total number of cycles was limited to 5 for pragmatic reasons: Intuitively, the utility of further cycles diminishes, and longer policies take longer to learn.

An action consists of issuing a specific reference $\Delta\theta \in \{0.1, 0.2, 0.5, 1, 2, 5\}$ to the vergence controller. The angles are given in degrees. The direction of the movement corresponds to the sign of p, i.e. the system is prevented from diverging when the best matching columns demand convergence, and vice versa.

Training is done on fixed-length trials, where a trial consists of s perception-action cycles. At the end of each trial, the achieved accuracy is estimated by acquiring a final image pair and computing p by performing a stereo match. Reward is then given by $-|p|$; all within-trial rewards are zero. At the end of each cycle, the Q-value of the state at the beginning of the cycle is updated according to the conventional non-discounting Q-Learning rule

$$\Delta Q(s, a) = \alpha[r + \max_{a'} Q(s', a') - Q(s, a)]$$

where s' is the state the system has found itself in after performing action a in state s. The Q function is represented by a table, indexed by $|p|, \bar{p}, s_r, |\Delta\theta|$. This table implicitly represents the vergence policy: Among all possible vergence adjustments possible in a state $s = \langle |p|, \bar{p}, s_r \rangle$, the action $a = \Delta\theta$ is selected that maximizes $Q(s, a)$.

During learning, this greedy policy is followed only 90% of the time. The other 10% of actions are selected randomly. This ensures that all actions continue to be selected in all states, which is a necessary precondition for the Q function to converge to the optimal one (Watkins and Dayan 1992). For on-line learning systems, this precondition gives rise to the so-called exploration/exploitation conflict (see below): Increasing random exploration may speed learning, but impairs the utility of the running system while it is learning.

This learning procedure learns the actions leading to a camera configuration where the best-matching columns are in the center of each camera's field of view. If the viewed scene contains enough information to unambiguously identify the best match, then this implies that the cameras are verged on a common feature in the world. Note that this is a bootstrapping estimate of the achieved accuracy: The same procedure is used to adjust vergence angles and to assess the resulting accuracy. No ground-truth information about the viewed scene is involved.

4.2 The Learning Procedure

Some care was taken to speed up the learning process. To begin, recall that the remaining length s_r of a trial forms part of the state description. Therefore, the update of a Q-value for a length-2 trial depends on the value of a length-1 trial. This structural dependency was exploited by search space *shaping*: First, 1000 length-1 trials were run to determine length-1 Q-values to some accuracy. One thousand length-2 trials followed, exploiting the length-1 Q-values already learned. Next, 1000 length-3 trials were run, etc.

Furthermore, the random noise was assumed to be stationary. Therefore, for length-1 $Q(s,a)$-values the step size α was chosen such that all learning experiences are weighted equally, without preference to more recent experiences. This is achieved by setting $\alpha = 1/k_{s,a}$ where $k_{s,a}$ is the number of updates of the particular Q value, including the current update. In conjunction with the above learning rule, this ensures that $Q(s,a)$ always is the mean of all past rewards this state/action pair s, a received.

While this adaptive step size selection is optimal for length-1 trials, this is not the case for longer trials. Counterexamples exist where equally weighted experiences prevent convergence to the optimal Q values. On the other hand, if one assumes in our setting that the length-1 Q-values have converged before training on length-2 trials commences, then the above optimality argument carries over to length-2 trials, and likewise to longer trials. Therefore, for all values of s the α values were selected in this manner.

At the beginning of each trial, the stereo head was pointed at a random direction in space. At the end of a perception-action cycle, the motor commands were selected according to an annealed softmax procedure (Sutton and Barto 1998): In state s, action a is selected with Boltzmann probability

$$P_a = \frac{e^{Q(s,a)/\tau}}{\sum_{a'} e^{Q(s,a')/\tau}}$$

where $\tau = 1000 \cdot 0.99^{k_s}$ is the "temperature" which depends on the number k_s state s has been encountered. Initially, each action is selected with roughly equal probability, and as k_s grows, the best actions are selected with increasing probability. During evaluation, actions were selected greedily, which corresponds to setting $\tau = 0$.

Figure 7 shows the learning curves for a sample training session consisting of 9667 trials. As explained above, the first 1000 trials were all of length 1 perception-action cycle, the next 1000 trials were all of length 2, etc. Starting at trial 5000, the length of each trial was randomly selected from $[1, 2, \ldots, 5]$.

5 Evaluation of the Learned Policies

To evaluate the learned vergence policies, a series of 8715 trials was run. The length of each individual trial (given as the number of perception-action cycles) was chosen at random. Vergence adjustments were always selected greedily.

In principle, it is not necessary to perform separate training and test runs as was done here. Due to the incremental nature of reinforcement learning, the vergence system could be installed untrained and put into service right away. Initial performance would be poor, but with increasing experience it would improve. Likewise, performance estimates can be acquired and updated as the system operates.

In this particular setting, the exploration/exploitation conflict can be resolved in an elegant way: Since policies were trained only for a maximum of five perception-action cycles, the system could use any excess time allowed by the scheduler for exploration.

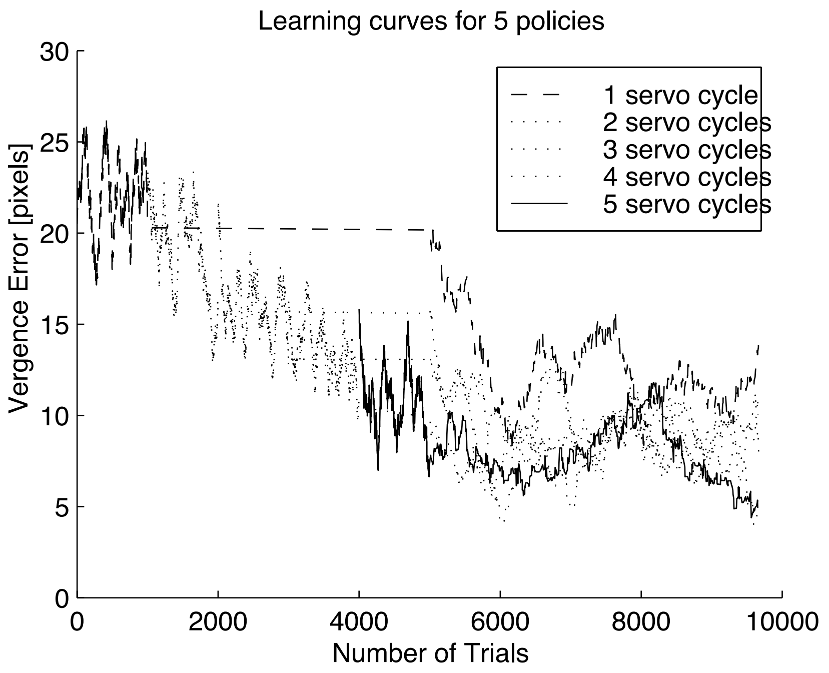

Figure 7: Learning curves for the five policies. Only the learning curves for length-1 and length-5 policies are clearly drawn; the others are indicated by dots.

5.1 Prediction of Success

In Section 3 successful vergence was defined as the two best matching image columns being neighbors. Figure 8 illustrates that this criterion is in fact an excellent predictor of success: The probability that the verge resulted in a best column index of -1, 0, or 1 was 0.782 in the case of neighboring best columns, and 0.095 in the case of non-neighboring best columns.

5.2 Characterization of Performance

The scheduler requires performance measures of the various learned policies. These measures are provided in terms of the success rate r, which is the proportion of trials ending with neighboring best-matching columns, and in terms of the distribution of accuracies achieved, as measured by the index of the best-matching column after completion of a trial. These distributions are shown in Figure 9. While they are not adequately approximated by Gaussian

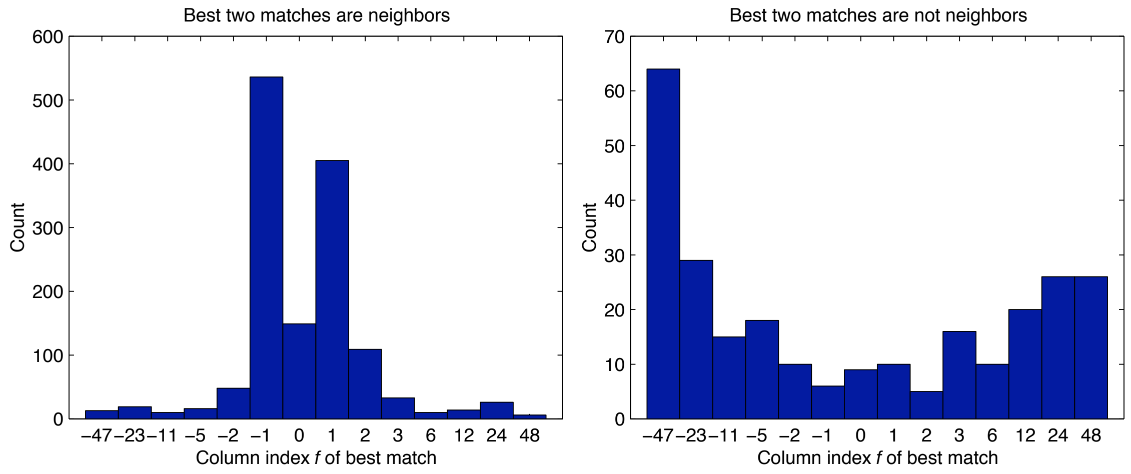

Figure 8: Histograms of best matching column indices p after completed vergence. Clearly, neighboring best-matching columns are an excellent predictor of a successful merge.

distributions, it may still be instructive to consider their standard deviations, here denoted by σ_c. These measures are summarized in Figure 10.

For the specialized length-k policies (solid curve in Figure 10), the success rate r increases with the number of available perception-action cycles, and – except for the length-5 trials – σ_c decreases monotonically. Hence, it is beneficial to use more cycles if time permits. To explain the inferior accuracy of the length-5 trials, it is reasonable to assume that the Q values for the length-5 trials had not yet converged. Since learning of the length-k values relies on the learned values of the length-$(k' < k)$ values, inaccuracies in the Q table accumulate as the trial lengths increase. Furthermore, the Q values corresponding to shorter trials were updated more often. For instance, at the end of the training run the length-1 values had been updated a full 8715 times (once at the last cycle of each trial), while the length-5 values had only been updated at the beginning of each length-5 trial, of which there were roughly 950.

Another question is whether it is advantageous to learn separate policies for various lengths of trials. To answer this, a separate run of 805 length-5 trials was run, while choosing greedy actions according to the length-1 Q-table at each cycle (see *Iterated length-1 policy* in Figure 10). The performance of this policy is inferior to the specialized policies for $k > 1$. We conclude that the improvement of the iterated versus the non-iterated length-1 policy

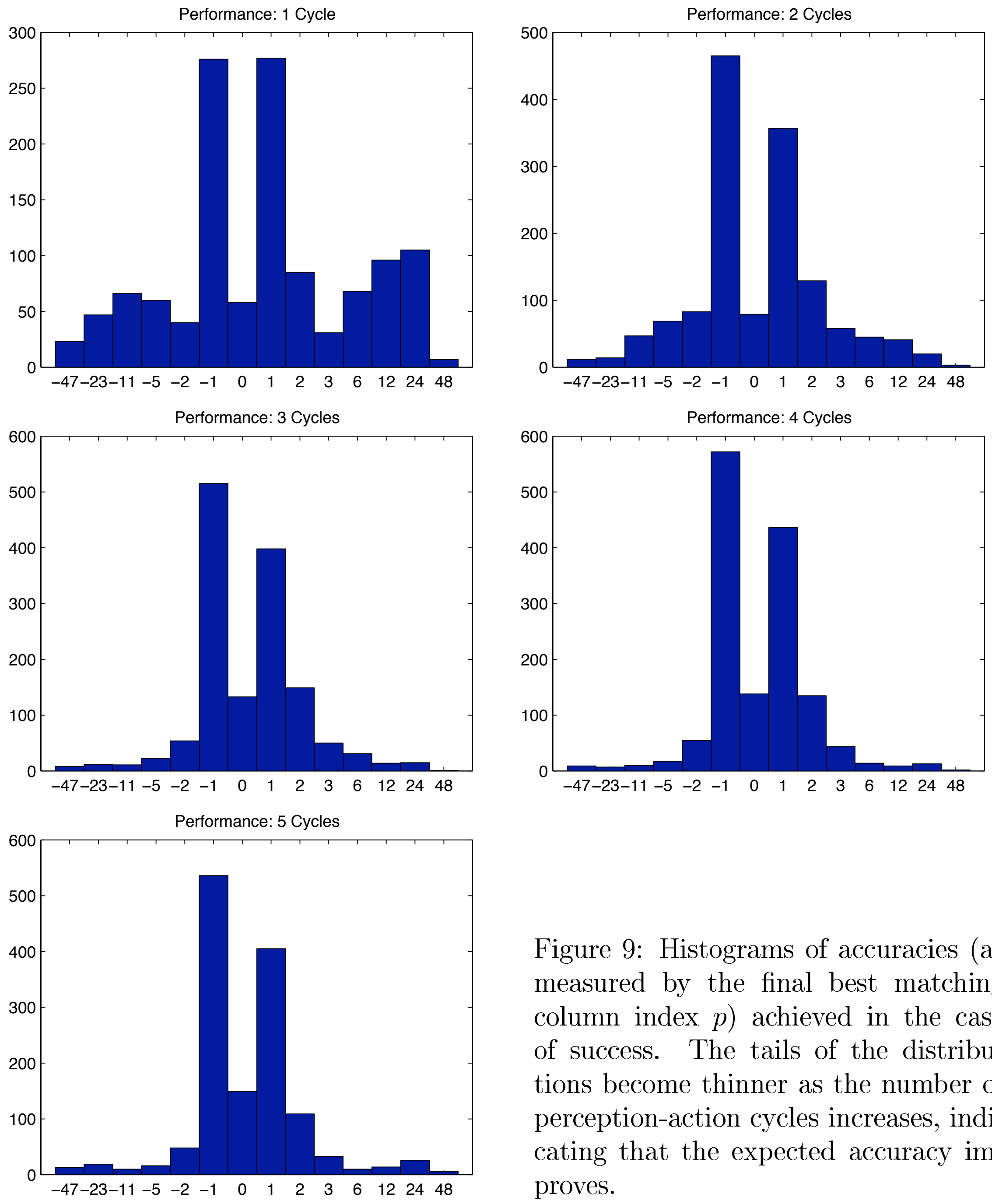

Figure 9: Histograms of accuracies (as measured by the final best matching column index p) achieved in the case of success. The tails of the distributions become thinner as the number of perception-action cycles increases, indicating that the expected accuracy improves.

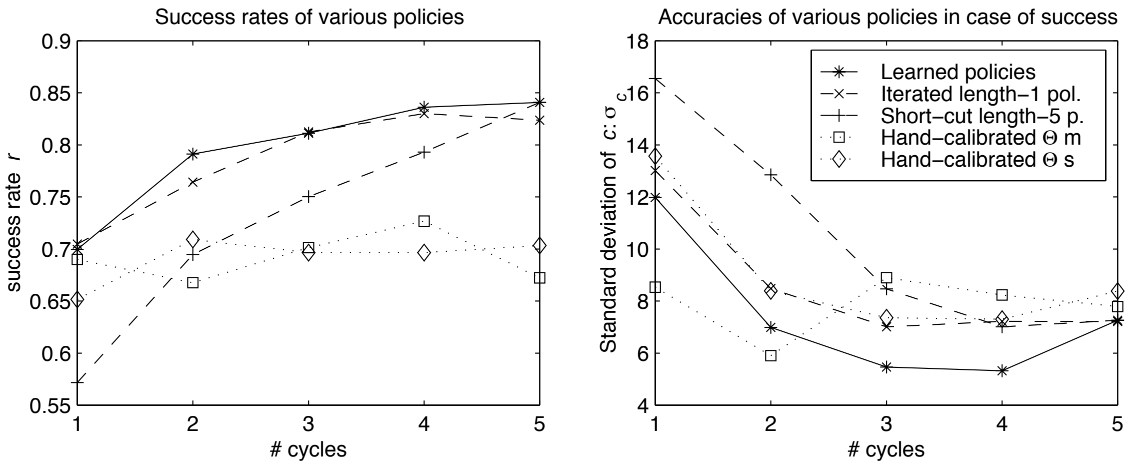

Figure 10: Performance variables. The solid curve in each graph represents five individual learned policies, one for each number of perception-action cycles. For example, the length-3 policy, applied iteratively for three cycles, achieved a success rate r of just above 0.8. The other four curves show the performance of a single policy, applied iteratively for the given number of cycles.

is due to the fact that more perception-action cycles were available for convergence, but the policy did not make maximum use of the additional cycles. Specialized policies take more advantage of additional cycles, as shown by both performance measures of the length-3 and length-4 policies, and the success rate r of the length-5 policy.

An intuitive explanation for this behavior is that a length-1 policy will use conservative vergence adjustments if best-matching columns are near the fovea, but aggressive adjustments if they are not. This will maximize the likelihood of achieving correct vergence in one shot, while minimizing the chance that an already foveated target is lost. If additional cycles are available, more conservative adjustments may perform better in the presence of noise.

This suggests a possible improvement to the learning setup: Presently, the state space includes information about the number of cycles to go. It might be beneficial to also include information about the number of cycles already executed. This would permit the emergence of a policy that "tries something else" after running for a while with little success.

How is the performance affected if a vergence task cannot execute all perception-action cycles it anticipated? To test this, we looked at the performance variables after executing only the first 1,2,3, or 4 perception-action cycles of the length-5 policy. The results (see *Short-cut length-5 policy* in Figure 10) clearly demonstrate the graceful degradation of all three performance variables as the task is cut short.

How do the learned policies compare with a hand-calibrated system? To test this, we calibrated our vergence axis to construct a table $\Theta : p \mapsto \Delta\theta$. We constructed an inverse mapping $\Theta^{-1} : \theta \mapsto p$ by acquiring a reference image at $\theta = 0$, and then turning the camera in small increments of θ while computing p values using the same matching procedure as described in Section 2, but with respect to the reference image taken earlier by the same camera. We ran this procedure multiple times to yield distributions of p column indices as a function of θ. From these we read off two Θ tables: One, Θ_m, turns the mean of the p distribution into the fovea, and a more conservative Θ_s turns just the edge of the p distribution into the fovea.

The performance measures of these hand-calibrated parameters is shown in Figure 10. The one-cycle Θ_m policy clearly outperforms the learned one-cycle policy in terms of σ_c. The conservative Θ_s performs similarly to the learned policy. However, the graphs show clearly that iterating these policies does not consistently improve their performance. As the number of perception-action cycles increases, the hand-calibrated policies are outperformed by the learned policies by increasingly wider margins. This shows that the optimal set of control parameters actually depends on the number of available perception-action cycles. It is not obvious how to find optimal Θ functions for tasks where more than one perception-action cycle is available. Here, the learning procedure found policies that do better than an iteratively repeated optimal length-1 policy.

5.3 Learned Policies

Figure 11 shows the greedy policies corresponding to the learned Q function. The greedy policy is given by the function

$$\Theta_f(p, p', s_r) = \arg\max_{\Delta\theta} Q(s, a),$$

where $s = \langle |p|, \bar{p}, s_r \rangle$ and $a = \Delta\theta$ as defined in Section 4.1.

The parabolic structure mapping large column indices monotonically to vergence angles can clearly be seen. Furthermore, one can see that more conservative vergence adjustments are chosen by the shorter-trial policies when the matched column index is near zero. On the other hand, it is quite obvious that the policies for the longer trials have not yet converged, especially in the case of neighboring best-matching columns.

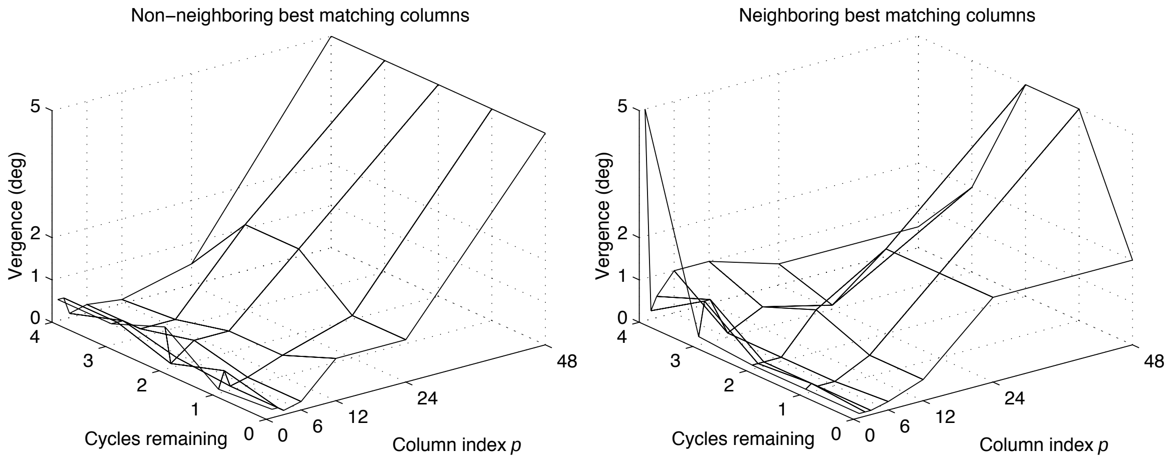

Figure 11: Decision surfaces implementing the policy. Vergence angles are given in degrees. The two plots correspond to the two possible values of $\bar{p}$. The vertical axes give the reference vergence angle adjustment $\Delta\theta$.

6 Discussion

This work contributes a machine learning approach to construct locally optimal time-constrained action policies. Probabilistic performance characteristics are estimated using experiences resulting from interactions with the real environment. These characterizations can be used by a scheduler to allocate an appropriate number of time slices to the vergence process, given certain performance requirements and resource limitations. The learned policies are designed to achieve maximal expected performance under the given schedule. They are robust with respect to unexpected changes of the schedule.

The interaction of real-time schedulers with robotic systems is still an open problem because of the unpredictable nature of the real world. For example, in a closed-loop control system the number of control cycles necessary to accomplish a task is only approximately known in advance. If the system is subjected to hard time constraints, the predictability of success may degrade to a probabilistic estimate of a binary success/failure outcome, e.g.

whether a pick-and-place task will or will not succeed. This work describes a task whose performance characteristics degrade gracefully in the presence of time constraints, and is therefore more amenable to a real-time framework.

From a robotics perspective, this work demonstrated the feasibility and usefulness of using a machine learning framework to learn control parameters of a mechanical system. This task is typically viewed as a system identification problem and is solved by calibration techniques. In contrast, this approach has the advantage that the control parameters are selected with regard to characteristics of visual scenes actually encountered. In the presence of noise (i.e. self-similar or poorly textured scenes that cause stereo mismatches), this is likely to pay off in terms of gained accuracy. This was not verified rigorously, but some anecdotal evidence is provided by a hand-tuned system whose performance was inferior to that of learned policies if more than one perception-action cycle is available.

An advantage of on-line learning systems like the one described here is that such systems can be put into service without initial calibration. They will improve their performance with experience, and – if a fixed minimum step size parameter α is used – will adapt to changing environmental and mechanical characteristics.

From a machine learning perspective, this work constitutes an example of reinforcement learning applied to a real-world problem involving a mechanical system. Such examples are still relatively rare because reinforcement learning techniques typically require a large amount of training, and training experiences involving a real environment are typically much more expensive than simulated experiences. To address this problem, the size of the Q table was kept small by using coarse quantization and by exploiting symmetries, and the learning performance was enhanced by shaping the exploration of the state space.

References

C. Capurro, F. P. and G. Sandini (1997). Dynamic vergence using log-polar images. *Int. J. Computer Vision 24*(1), 79–94.

Hansen, M. and G. Sommer (1996). Active depth estimation with gaze and vergence control using Gabor filters. In *In 13th Int. Conf. on Pattern Recognition*, Volume A, Vienna, Austria, pp. 287–291.

Liu, J. W. S., K. J. Lin, W. K. Shih, A. C. Yu, J. Y. Chung, and W. Zhao (1991). Algorithms for scheduling imprecise computations. In A. M. van Tilborg and G. M. Koob (Eds.), *Foundations of Real-Time Computing: Scheduling and Resource Management*, Chapter 8, pp. 203–249. Kluwer Academic Publishers.

Rao, R. P. N. and D. H. Ballard (1995). An active vision architecture based on iconic representations. *Artificial Intelligence 78*, 461–505.

Sutton, R. S. and A. G. Barto (1998). *Reinforcement Learning: An Introduction.* Cambridge, Massachusetts: MIT Press.

Watkins, C. J. C. H. (1989). *Learning From Delayed Rewards.* Ph. D. thesis, University of Cambridge, England.

Watkins, C. J. C. H. and P. Dayan (1992). Q-Learning. *Machine Learning 8*, 279–292.

Weiman, C. F. R. (1994). Log-polar binocular vision system. Technical Report CR-188375, NASA.

Young, S. S., P. D. Scott, and C. Bandera (1998). Foveal automatic target recognition using a multiresolution neural network. *IEEE Trans. on Image Processing 7*(8), 1122–1135.

Zilberstein, S. (1996). Using anytime algorithms in intelligent systems. *AI Magazine 17*(3), 73–83.