Learning Real-Time Stereo Vergence Control

Justus H. Piater, Roderic A. Grupen, Krithi Ramamritham
Computer Science Department
University of Massachusetts, Amherst, MA, 01003, USA

E-mail: {piater|grupen|krithi}@cs.umass.edu

Abstract

On-line learning robotic systems have many desirable properties. This work contributes a reinforcement learning framework for learning a time-constrained closed-loop control policy. The task is to verge the two cameras of a stereo vision system to foveate on the same world feature, within a limited number of perception-action cycles. On-line learning is beneficial in at least the following ways: (1) The control parameters are optimized with respect to the characteristics of the environment actually encountered during operation; (2) visual feedback contributes to the choice of the best control action at every step in a multi-step control policy; (3) no initial calibration or explicit modeling of system parameters is required; (4) the system can be made to adapt to non-stationary environments. Our vergence system provides a running estimate of the resulting verge quality that can be exploited by a real-time scheduler. It is shown to perform superior to two hand-calibrated vergence policies.

1. Introduction

Much research effort is currently expended on adaptive robotic systems that can receive abstract task specifications, and are capable of learning the detailed actions required to perform these tasks. There are many motivations for such machine learning approaches, such as the difficulties of controlling a large number of degrees of freedom [6], or a non-stationary and partially observable environment [7]. Under such circumstances, the best reported results are often achieved using learning techniques.

The majority of current robot learning research addresses relatively complex high-level tasks. This paper addresses the low-level problem of controlling a single-DOF manipulator under real-time constraints, using visual feedback. The term “real time” denotes the requirement that the resource requirements and performance of a system are deterministic [11]. Applied to robotic control, this may mean, for instance, that the functional relationship between speed and accuracy of a manipulator is precisely known. Since a manipulator as a physical system is subject to various disturbances that arise in the real world, the objective of determinism cannot strictly be attained. Therefore, an alternative formulation must be found that allows the application of real-time principles to robotic control. This issue has not received much attention, although it seems that there are many applications that would benefit from real-time scheduling of resources. Examples include scenarios where robots need to keep up with an environment whose behavior is not entirely controlled by the robots, and where resources are shared among multiple components.

We discuss an application involving an articulated stereo vision system. The control objective is to verge the cameras such that both image centers display the same world feature. This task is relatively difficult because the performance characteristics of an articulated camera system depend on many intrinsic and extrinsic factors that can hardly be accounted for by conventional system identification and calibration techniques. Therefore, we propose a direct learning approach that finds optimal control parameters for time-constrained vergence policies, and show that the learned parameters outperform a hand-calibrated system in a real environment. While the methods employed are specific to our image representation and the vergence task, they apply directly to other closed-loop visuomotor tasks, e.g. tracking of moving objects. The general principles of direct control learning and real-time characterization of a non-deterministic robotic system are applicable to other low-level control tasks as well.

The vergence task is described in the following section. Section 3 discusses the task and its role in a real-time framework. These two sections cover the aspects essential to this paper. For additional detail the interested reader is referred to [8]. 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

This work was supported in part by the NSF under grants CDA-9703217 and IRI-9704530, and by DARPA/AFOSR and AFRL/IFTD under Contracts F49620-97-1-0485 and F30602-97-2-0032.

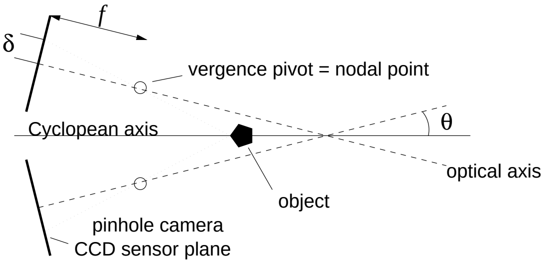


Figure 1: Symmetric vergence geometry.

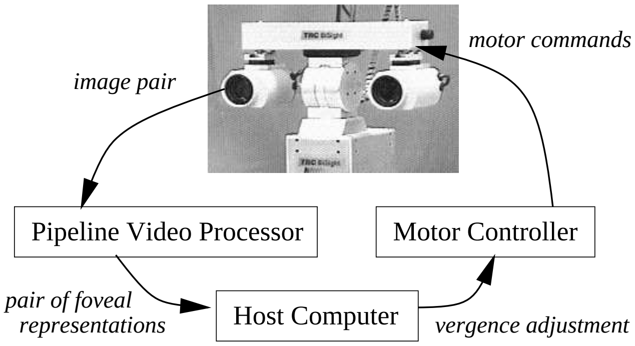


Figure 2: Simplified structure of the vergence task.

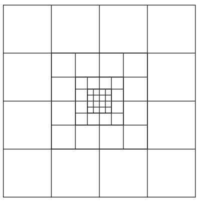


Figure 3: Logarithmic subdivision of an image. Each level is represented as a 4×4 grid. Each square within each grid represents a vector of 6 features, which are the responses of 6 different oriented Gaussian-derivative filters applied at this location.

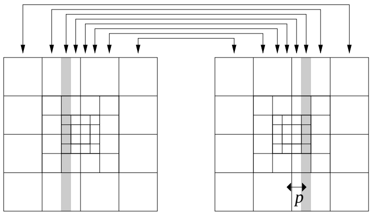


Figure 4: Illustration of the symmetric matching process (here shown for 3 levels of resolution). The shaded areas indicate one pair of matched columns at the highest resolution. A feature vector is constructed as the concatenation of all feature vectors associated with all squares at all levels of resolution 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.

point in the world (Figure 1). Figure 2 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.

For purposes of image data reduction, we generate a multi-resolution foveal representation by successively subsampling the image to half the (linear) resolution, and keeping the highest resolution only in the center of the image [2]. The resulting arrangement of the image is illustrated in Figure 3. 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 [9]. The subsampling and the convolutions are performed well within a frame time on the pipeline 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 (Figure 4). 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, a 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*. These cycles can be iterated to increase vergence accuracy.

3. Real-Time Issues

In our real-time scenario, the vergence system is allocated a fixed number of perception-action cycles for a given verge trial. The goal of the system is to choose the control actions at each cycle such that the quality of the resulting verge is maximized. The number of cycles is determined by an external scheduler, which takes into account other constraints as well as the required quality of the verge. Hence, the functional relationship between the expected performance of the vergence system and the number of perception-action cycles must be characterized appropriately. This is done by forming a probability distribution of the vergence qualities observed after a given number of cycles.

Since the number of remaining perception-action cycles is known in advance to the system, it should choose $\Theta(p)$ depending on this number. For instance, it may prove useful to apply conservative vergence angle corrections $\Delta\theta$ 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 such a $\Theta(p)$ optimally. This paper contributes a machine learning approach that learns to maximize the expected vergence accuracy, given the number of remaining perception-action cycles.

Our vergence task is nonstandard among real-time tasks, but typical of many robotics tasks in several ways because a mechanical plant is involved that is subject to perturbations. Therefore, no hard performance 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 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': 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.

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 in principle be solved by calibration techniques (see Section 5). On the other hand, a $\Delta\theta$ that is optimal in some appropriate sense may 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 attempt to foveate by a single adjustment. This will reduce jerky moves in response to outliers. If more than one perception-action cycle is available, it is still possible to achieve good final accuracy.

The traditional approach employs a closed-loop PD control model [5]. 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 for reliable matching, or from highly non-stationary scenes that degrade the predictable relationship between consecutive image pairs before and 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. Such an estimate is provided by the index p' of the second-best matching pair of columns, as noted above.

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 $\Theta(p)$. It should be less susceptible to noise, and it can provide a running estimate of the reliability of the achieved result. The function $\Theta_f(p, p', s_r)$ is learned in a reinforcement learning framework, which is described below. Reinforcement learning [10] is attractive for control problems because it allows an agent to learn reactive control policies directly by trial-and-error, and does not require a-priori specification of control parameters. Reinforcement methods have been applied successfully in a variety of robotic tasks [3, 1, 4, 6].

The problem of learning $\Theta_f(p, p', s_r)$ is expressed in the Q-Learning framework [12], 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 is forced to correspond 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 a given number of perception-action cycles. At the end of each trial, the accuracy achieved is estimated by acquiring a final image pair and computing p using a stereo match. Reward is then given by $r = -|p|$; 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

$$\begin{aligned} Q_{\text{new}}(s, a) &= Q_{\text{old}}(s, a) + \\ &\quad \alpha[r + \max_{a'} Q_{\text{old}}(s', a') - Q_{\text{old}}(s, a)] \end{aligned}$$

where s' is the state the system has found itself in after performing action a in state s. The Q function is represented as 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 some of the time. A fraction of all 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 [13]. 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 ($p = 0$). 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.

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 total 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 each perception-action cycle, the motor commands were selected according to an annealed softmax procedure [10]: 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. This popular procedure favors exploration in the early stages of training, and emphasizes exploitation as experience increases. During evaluation, all actions were selected greedily, which corresponds to setting $\tau = 0$.

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 is poor in this case, but improves with increasing experience. Likewise, performance estimates can be acquired and updated as the system operates.

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

Prediction of Success

In Section 3 successful vergence was defined as the two best matching image columns being neighbors. Figure 5 illustrates that this criterion is in fact an excellent predictor of success: The probability that the verge resulted in a best-matching column index p of -1, 0, or 1 was 0.69 in the case of neighboring best and second-best columns, and 0.06 in the non-neighboring case.

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. The accuracy distributions for the 1- and 5-step policies are shown in Figure 6. While they are not adequately approximated by Gaussian distributions, it may still be instructive to consider their standard deviations, here denoted by σ_c. These measures are summarized in Figure 7.

For the specialized length-k policies (solid curve in Figure 7), 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,

Figure 5: Histograms of best-matching column indices p after completed vergence. Clearly, neighboring best-matching columns are an excellent predictor of a successful merge. The relatively low count for $p = 0$ is a side effect of the asymmetry in the way p is measured (one of the two center columns represents $p = 0$).

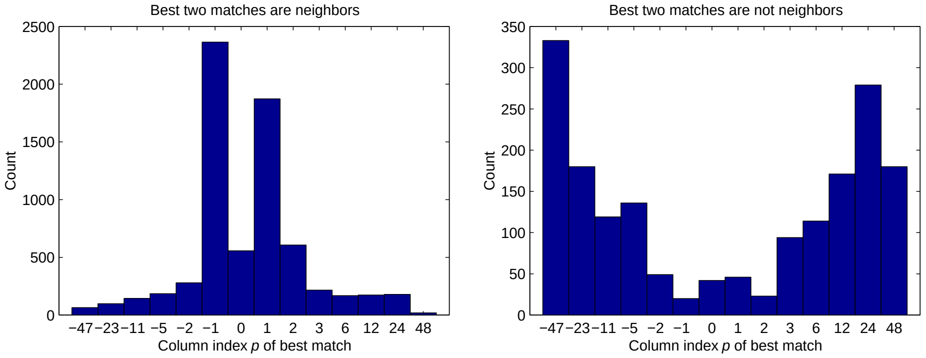
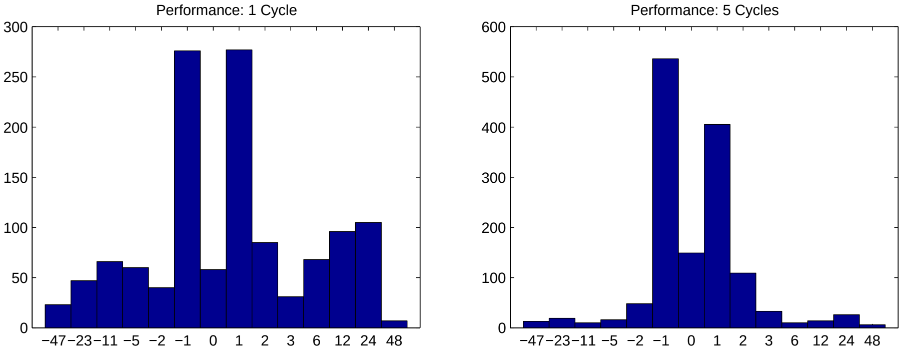


Figure 6: Histograms of accuracies (as measured by the final matching column index p) achieved in the case of success. The tails of the distributions become thinner as the number of perception-action cycles increases, indicating that the expected accuracy improves. Only the histograms for the 1- and 5-step policies are shown.



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.

Is it 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 7). 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 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.

How is the performance affected if a vergence task cannot execute all perception-action cycles it anticipated? The answer is given by 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 7) clearly demonstrate the graceful degradation of both 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 are shown in Figure 7. 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 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 cycle is available. Here, the learning procedure found policies that outperform iterated hand-tuned length-1 policies.

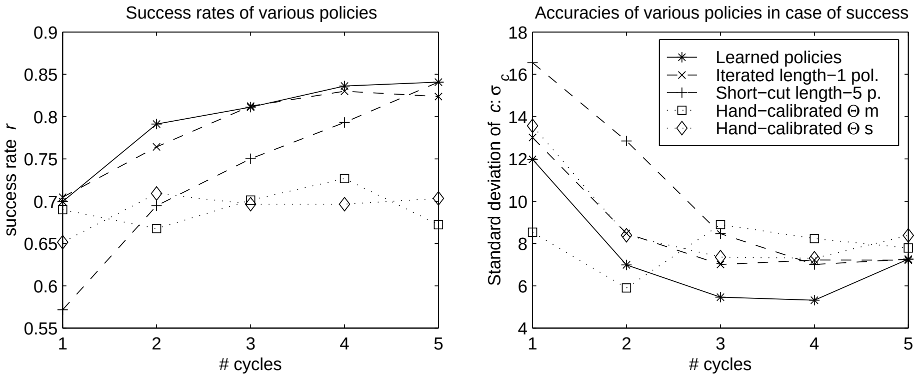


Figure 7: Performance variables. The solid curve in each graph represents five individual learned policies, one for each number of perception-action cycles. For example, the length-3 policy, applied iteratively for three cycles, achieved a success rate r of just above 0.8. The other four curves show the performance of a single policy, applied iteratively for the given number of cycles.

6. Discussion

We have contributed 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.

This work constitutes progress toward interaction of real-time schedulers with robotic systems, which is a difficult 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.

We demonstrated the usefulness of an on-line machine learning technique for learning control parameters of a mechanical system. In contrast to conventional techniques, 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 illustrated by a comparison with a hand-tuned system whose performance was inferior to that of learned policies, especially if more than one perception-action cycle is available.

Finally, on-line learning systems such as the one described here 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.

References

[1] C. G. Atkeson and S. Schaal. Learning tasks from a single demonstration. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 1706–1712, Albuquerque, NM, April 1997.

[2] C. Bandera and P. Scott. Foveal machine vision systems. In *IEEE Int. Conf. on Systems, Man, and Cybernetics*, Cambridge, MA, November 1989.

[3] J. Connell and S. Mahadevan, editors. *Robot Learning*. Kluwer Academic, Boston, MA, 1993.

[4] A. H. Fagg, D. Lotspeich, and G. A. Bekey. A reinforcement-learning approach to reactive control policy design for autonomous robots. In *Proc. IEEE Conference on Robotics and Automation*, 1994.

[5] M. Hansen and G. Sommer. Active depth estimation with gaze and vergence control using Gabor filters. In *In 13th Int. Conf. on Pattern Recognition*, volume A, pages 287–291, Vienna, Austria, 1996.

[6] M. Huber and R. A. Grupen. A feedback control structure for on-line learning tasks. *Robots and Autonomous Systems*, 22(3/4):303–315, 1997.

[7] F. Michaud and M. J. Matarić. Learning from history for behavior-based mobile robots in non-stationary conditions. *Machine Learning*, 31(1–3):141–167, 1998. Joint special issue with *Autonomous Robots* on Learning in Autonomous Robots.

[8] J. H. Piater, K. Ramamritham, and R. A. Grupen. Learning real-time strategies for binocular vergence. Computer Science Technical Report 99-06, University of Massachusetts, Amherst, Feb. 1999.

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

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

[11] A. M. van Tilborg and G. M. Koob, editors. *Foundations of Real-Time Computing: Scheduling and Resource Management*. Kluwer Academic Publishers, 1991.

[12] C. J. C. H. Watkins. *Learning From Delayed Rewards*. PhD thesis, University of Cambridge, England, 1989.

[13] C. J. C. H. Watkins and P. Dayan. Q-Learning. *Machine Learning*, 8:279–292, 1992.