Learning Saccadic Gaze Control via Motion Prediction

Per-Erik Forssén
Department of Computer Science
University of British Columbia
2366 Main Mall, Vancouver, B.C.
perfo@cs.ubc.ca

Abstract

This paper describes a system that autonomously learns to perform saccadic gaze control on a stereo pan-tilt unit. Instead of learning a direct map from image positions to a centering action, the system first learns a forward model that predicts how image features move in the visual field as the gaze is shifted. Gaze control can then be performed by searching for the action that best centers a feature in both the left and the right image. By attacking the problem in a different way we are able to collect many training examples in each action, and thus learning converges much faster. The learning is performed using image features obtained from the Scale Invariant Feature Transform (SIFT) [14] detected and matched before and after a saccade, and thus requires no special environment during the training stage. We demonstrate that our system stabilises already after 300 *saccades, which is more than* 100 *times fewer than the best current approaches.*

1. Introduction

This paper describes a system that autonomously learns to perform *saccadic gaze control* on a stereo pan-tilt unit. It is well established that saccades in humans are too fast to involve visual feedback [6]. Thus, the goal of saccadic gaze control is different from visual servoing, where an error in visual space (or some state space, e.g. pose) is minimised through closed-loop control. Instead, given the coordinates of an object in both left and right image of a stereo pair, the system should directly output the pan and tilt states for which the object is best centred in the stereo pair. Once such a mapping is known, the system will be able to shift its gaze quickly between different objects in the scene.

Saccade control in humans is useful for rapid visual exploration of a scene. Beside scene exploration, we also intend to make use of the saccade control ability in an assisted learning setting, where the robot should be able to make inquiries about objects to a nearby person, by means of looking at an object and asking a question, see Figure 1 for an illustration. Using gaze to indicate objects of interest requires that the gaze shift is both fast (to retain the interest of the assisting person) and accurate, such that there should be no doubt about which object the robot is inquiring about.

Figure 1. Using gaze instead of pointing to indicate an object of interest. Left: Robot and person indicating a common object of interest. Right: Close-up of the pan-tilt stereo head.

In human-robot interaction, saccadic gaze control is also important for making eye contact. Here gaze is used as a non-verbal indication that one is paying attention to the other party in a conversation [18], and to indicate that one intends to address someone [11]. Gaze can also be used as an aid to grasping, by indicating in which direction an object is located [9], and for pose estimation in the action space [12].

1.1. Review of related work

Bruske et al. [6] learned saccade control using a neural network called Dynamic Cell Structures (DCS), which was trained using *feedback error learning* (FEL). In their approach the computer vision aspect of the problem was bypassed by having a controlled stimulus in the form of a dark room and a single point light source. Another example of a

system using FEL and a point light source is described by Berthouze et al. [3]. This system, however learns both saccade control and tracking within the same framework. They also track objects and imitate wavy motions using image optical flow as input.

Hoffmann et al. [9] have a more biologically inspired approach to saccade control learning. They use somewhat more general stimuli, in the form of a white table filled with brightly coloured blocks. In their approach a target is selected in the left image. The corresponding pattern is then found in the right image using cross-correlation. A saccade attempting to centre the pattern in both views is then made, according to the currently best strategy, and the pattern is then detected in the two images, again using cross-correlation. If the error in the image plane is less than average, this sample is stored and eventually used to learn a new, better strategy. This approach is called *staged learning*. It improves on feedback error learning by not having to know how errors in the image plane relate to errors in gaze.

Another related system is the one proposed by Léonard et al. [12]. This system does not rely on a controlled environment, instead a slowly moving camera is used, and this allows the use of tracking for learning to center arbitrary scene points. Since frames are captured during the centering movement more training data can also be acquired during each centering action. The gaze control function is then learned on-line using reinforcement learning and incremental least squares in a regular gridding of the input space. To simplify the gaze control function, the authors separate the learning problem into one for learning pan control from differences in horizontal image positions, and one for tilt control from differences in vertical image positions. From a practical point of view this approach seems quite reasonable, it is also more versatile than the other approaches since it does not rely on a controlled environment.

Finally, for a camera without radial distorsion, which rotates purely about the camera centre, the motion of scene points are given by a *homography* [8]. If the demands on gaze accuracy are not too high, one can simply assume that these conditions hold, and avoid learning altogether, as was done in e.g. [7].

1.2. Contributions

Our main contribution is that in contrast to the reviewed approaches [12, 9, 6, 3], we do not solve the problem using *reinforcement learning* (RL) [17]. In RL, each action gives us one example of inputs and outputs, and an associated error, or reward, and thus many actions are needed to successfully learn the mapping. For instance, Hoffmann et al. report seven learning stages of $10\,000$ trials each [9] to learn a saccade control mapping, Bruske et al. report $40\,000$ trials [6], and Léonard et al. report 400 actions, with $100 - 150$ frames during each action [12].

Instead of using RL, we propose to learn a forward model that predicts how features in the image move as the gaze is shifted. This learning problem is functional, which reduces the amount of samples needed. Furthermore, each action gives us many examples of the desired mapping (one for each feature correspondence), and consequently our approach speeds up learning by several orders of magnitude.

Our approach is also an improvement compared to state of the art, in the respect that we make no simplifying assumptions about the relations between image changes and the motor controls as was done in [6, 9, 12]. This makes our approach easier to apply to any two-camera pan-tilt unit, without needing to align the cameras.

We also demonstrate that the use of image features obtained from the *Scale Invariant Feature Transform* (SIFT) [14], removes the need for slow camera motions during learning [12]. Instead the camera directly performs saccades and then SIFT features are matched before and after the saccade.

1.3. Notation

Throughout the paper, we will denote the pan-tilt state at time t by $\mathbf{s}(t) = (\phi(t),\ \theta(t))^T$. Image positions in the left and right images at time t are denoted $\mathbf{p}_l(t) = (x_l(t),\ y_l(t))^T$, and $\mathbf{p}_r(t) = (x_r(t),\ y_r(t))^T$ respectively.

2. Saccade Control Learning

The purpose of saccade control learning is to find a mapping that outputs a pan-tilt state that centres a given object in the visual field

$$\begin{pmatrix} \mathbf{s}(t) \\ \mathbf{p}_l(t) \\ \mathbf{p}_r(t) \end{pmatrix} \mapsto \mathbf{s}(t+1)\,. \tag{1}$$

The quality of a given output action $\mathbf{s}(t+1)$ is given by the *centering error*:

$$\varepsilon(\mathbf{s}(t+1)) = ||\mathbf{p}_l(t+1) + \mathbf{p}_r(t+1)||\,. \tag{2}$$

Here it is assumed that the image origin is the centre of the image. Note that even though both $\mathbf{p}_l$ and $\mathbf{p}_r$ can never be centered perfectly at the same time (2) still becomes zero when they are positioned symmetrically about the origin.

By performing *trial* actions, and observing the resultant *errors* (2), the learning problem can be approached using *reinforcement learning* [17]. This is what was done in all the approaches mentioned above [12, 9, 6, 3].

2.1. Movement Prediction Learning

In contrast to the reinforcement learning approach, we propose to learn saccade control indirectly, by instead learning to predict how features will move in the image as the camera is moved, given the current pan-tilt state. Once this skill is mastered to some degree, it can be used in reverse to generate reasonable saccades for centering an observed object.

The inputs of the sought mapping are the states before and after the saccade $\mathbf{s}(t)$, and $\mathbf{s}(t+1)$, and the image coordinates before the saccade $\mathbf{p}_l(t)$, and $\mathbf{p}_r(t)$. The outputs are the predicted image coordinates $\mathbf{p}_l(t+1)$ and $\mathbf{p}_r(t+1)$. That is, we learn the 8 to 4-dimensional predictive mapping

$$\begin{pmatrix} \mathbf{s}(t) \\ \mathbf{s}(t+1) \\ \mathbf{p}_l(t) \\ \mathbf{p}_r(t) \end{pmatrix} \mapsto \begin{pmatrix} \mathbf{p}_l(t+1) \\ \mathbf{p}_r(t+1) \end{pmatrix} . \tag{3}$$

Note that we do not make any simplifying assumptions about the geometry, such as $y_l = y_r$ [6, 9], or $\phi(t+1) = f(x_l(t), x_r(t), \phi(t))$ [12]. This makes our approach easy to apply to any two-camera rig.

2.2. A Forward Model

Our motion prediction learning can be considered a *forward model* [10] of (1), also known as a *system identification* step [13] in control theory. Forward models are useful in situations where the *inverse model*, i.e. (1), is non-unique, or multi-valued. For the saccadic gaze control problem, a forward model would learn to predict how a point that the system tries to center actually moves. Since our mapping is one-to-one, there would seem to be no obvious advantage with using a forward model here.

However, instead of directly going for the centering goal, where each action only gives us one observation, our system first has the goal of predicting how *all* points in the scene move. To attain this goal it is obvious that as many motion vectors as possible should be collected in each saccade. Only when the motion prediction skill is mastered, the system switches to the centering goal.

3. Perception System

We make use of SIFT features [14] to find corresponding points before and after a saccade. A SIFT feature consists of a *location* (x and y coordinates, a reference scale and a rotation) and a 128-byte *descriptor* vector that describes the local image region around the location. We find corresponding points by matching SIFT features according to least squares descriptor distance. For a correspondence to be accepted we require that the best match from left to right image also is a best match from right to left. Additionally we require the correspondence to be the best one from the first stereo pair to the second, and from the second to the first. By using this forward-backward checking approach, we remove the need for thresholds, and eliminate nearly all false matches.

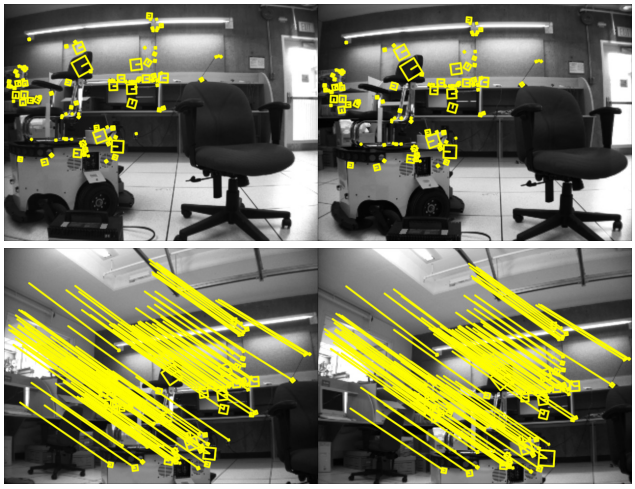

Figure 2. Feature matching across a saccade. Top: Stereo pair before saccade, with matched SIFT features painted in. Bottom: Stereo pair after saccade, with 114 estimated feature motion vectors painted in. The saccade shown has a pan change of 20° and tilt change of 14°. Each image has a 640 × 480 resolution.

Figure 2 shows an example of the training data obtained by matching SIFT features across a saccade. For this particular example we obtained 114 different motion vectors that all share the same start $\mathbf{s}(t)$ and end state $\mathbf{s}(t+1)$.

3.1. Hardware

The stereo camera used in the experiments is a Bumblebee from PointGrey Research. It delivers pairs of synchronised frames in 640 × 480 resolution at 30 Hz rate. The camera is mounted on a PTU-D46-17.5 pan-tilt unit from Directed Perception, see Figure 1, right. As can be seen in Figure 2, the images we obtain are uncalibrated, and thus contain significant amounts of radial distortion (e.g. straight world lines are bent in the image plane), and the vertical axes of the two cameras are not perfectly aligned (see e.g. the cable loop at centre bottom of the two upper images). As we shall see in the following, our algorithm has no problem coping with this.

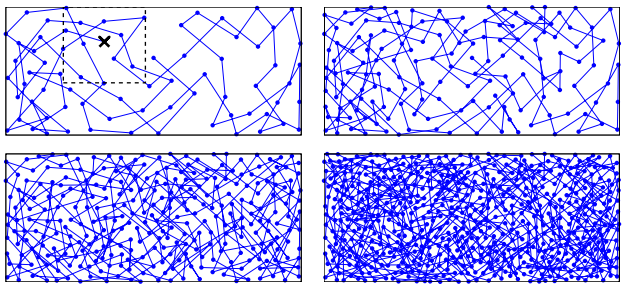

Figure 3. Pan-tilt states visited after 75, 150, 300, and 600 saccades. The search box for a new saccade is shown by the dashed box in the top left figure. The full range for pan (horizontal axis) is $[-90, 90]$, and the tilt range (vertical axis) is $[-47, 31]$.

4. Exploratory Movements

To acquire training data for learning the movement prediction map (3), the robot generates semi-random exploratory movements that successively cover the state-space with increasing density, see Figure 3. The exploratory actions are chosen one-at-a-time using knowledge of K previously visited states $\{\mathbf{s}_k\}_{k=1}^K$. This knowledge is specified in the form of a *kernel density estimator* (see e.g. [4]):

$$p(\mathbf{x}) = \gamma \sum_{k=1}^{K} \exp(-.5||\mathbf{s}_k - \mathbf{x}||^2)\,. \tag{4}$$

The scaling $\gamma > 0$ ensures that (4) sums to 1. We now want to find a new action $\mathbf{s}_{K+1}$ that maintains the density $p(\mathbf{x})$ close to uniform, under the constraint that the action should center something that is actually in the visual field. To find a suitable new action, we randomly generate 50 possible new states $\mathcal{P} = \{\mathbf{x}_n\}_{n=1}^{50}$ in a square search box with side $50°$ (should be smaller than the field-of-view) centred around the current state. The search box is cropped by the box defined by the maximum and minimum pan and tilt states, see Figure 3, top-left. The action chosen is the one which minimises (4), i.e.:

$$\mathbf{s}_{K+1} = \arg\min_{\mathbf{x}_n \in \mathcal{P}} p(\mathbf{x}_n)\,. \tag{5}$$

As can be seen in Figure 3 this mechanism is effective in producing an approximately uniform density as the number of samples is increased.

Mere coverage of the state space is not enough to ensure a good sampling for learning (3), however. Neighbouring samples in time, $\mathbf{s}(t)$ and $\mathbf{s}(t+1)$, also have to cover the set of possible movement distances and directions well. Figure 4 shows the distribution of gaze displacements $\mathbf{d} = \mathbf{s}(t) - \mathbf{s}(t+1)$, after 600 saccades. As can be seen, with the exception of very small gaze displacements, we obtain a good coverage. We can thus expect a slightly lower accuracy for very small saccades.

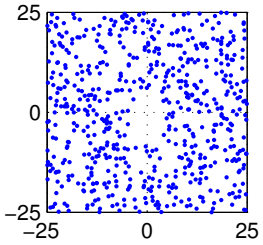

Figure 4. Gaze displacement coverage.

4.1. Learning Motion Prediction

We make use of *memory based learning* and *locally weighted regression* [1, 15] to represent the predictive mapping (3). That is, we store a number of prototype samples, and produce an output by interpolation from these prototypes.

To select prototypes, we employ a variant of batch K-means clustering [4]. Instead of starting the clustering by assigning each sample a random label, we start by randomly picking K samples as cluster centres. This ensures that each cluster has at least one member after K-means has converged. In all other respects our implementation is identical to the formulation in [4].

We apply K-means to the full 12 dimensional space spanned by both inputs and outputs of the sought map (3). This gives us a set of prototype vectors $\mathbf{a}_n^T = \begin{pmatrix}\mathbf{s}(t) & \mathbf{s}(t+1) & \mathbf{p}_l(t) & \mathbf{p}_r(t)\end{pmatrix}^T$ with associated responses $\mathbf{u}_n^T = \begin{pmatrix}\mathbf{p}_l(t+1) & \mathbf{p}_r(t+1)\end{pmatrix}^T$, which can be arranged in a tree for fast near neighbour access [2].

To evaluate the mapping for a new query sample $\mathbf{a}$, we find its D nearest neighbours $\{\mathbf{a}_d\}_{d=1}^D$, and their associated responses $\{\mathbf{u}_d\}_{d=1}^D$, by accessing the tree. We then fit a linear model $\mathbf{u}_d = \mathbf{C}\mathbf{a}_d$, to the neighbours using weighted least squares:

$$\hat{\mathbf{C}} = \arg\min_{c_{ij}} \sum_{d=1}^{D} ||\left(\mathbf{u}_d - \mathbf{C}\mathbf{a}_d\right)\mathbf{W}||^2\,. \tag{6}$$

Here $\mathbf{C}$ is a 4×8 matrix representing the local model, and $\mathbf{W}$ is a diagonal matrix containing weights for each sample. The weights w_d are a Gaussian function of the prototype distance:

$$w_d = \exp(-.5||\mathbf{a}_d - \mathbf{a}||^2/\sigma^2)\,. \tag{7}$$

The kernel scale σ is adapted to the local sample density according to:

$$\sigma = \frac{r}{D/2} \sum_{d=1}^{D/2} ||\mathbf{a}_d - \mathbf{a}|| \,, \tag{8}$$

where r is a parameter to be tuned [15].

The network output for the query point $\mathbf{a}$ is now given by

$$\hat{\mathbf{u}} = \hat{\mathbf{C}}\mathbf{a} \,. \tag{9}$$

Often prototype-based learning instead makes use of a weighted average of the neighbours, i.e.

$$\hat{\mathbf{u}} = \frac{\sum_{d=1}^{D} w_d \mathbf{u}_d}{\sum_{d=1}^{D} w_d} \,, \tag{10}$$

see, e.g., [15]. Such an approach makes the implicit assumption that the function is locally constant. Our alternative is slightly more computationally expensive, but has the advantage of using a local hyperplane model. This reduces the number of prototypes needed, and gives better extrapolating behaviour whenever most neighbours end up on one side of the query point.

5. Saccade Control from Motion Prediction

We will now make use of the learned predictive map (3) to find an action that centers a pair of corresponding points. In addition to the corresponding points $\mathbf{p}_l(t) \leftrightarrow \mathbf{p}_r(t)$ we also know the current state $\mathbf{s}(t)$ in (3). This means that by choosing an action $\mathbf{s}(t+1)$ we can predict the new feature positions using (3), and then compute the centering error (2). Since the best action $\mathbf{s}(t+1)$ is the one with the smallest predicted centering error, we can now find a good centering action by performing a minimum value search.

5.1. Minimum value search

The minimum value search of (2) is performed by iteratively fitting a polynomial to 9 points sampled in a 3×3 grid around the current best guess. The search is started with a grid box with offsets of $-30°$, $0°$ and $30°$ relative to the current value, for both pan and tilt. After each iteration, the box is centered around the minimum of the estimated polynomial and the box size is reduced by a factor $2/3$, see Figure 5 for an illustration. If the displacement of the box is larger than $1/2$ of the box side, the displacement is aborted, and the box size is instead increased a factor 1.1. Typically about 20 iterations, and thus 180 evaluations of (3) are required to zoom in on the minimum at PTU accuracy (which is $0.0129°$). When the number of training samples is low, the iterations should instead be stopped when the accuracy

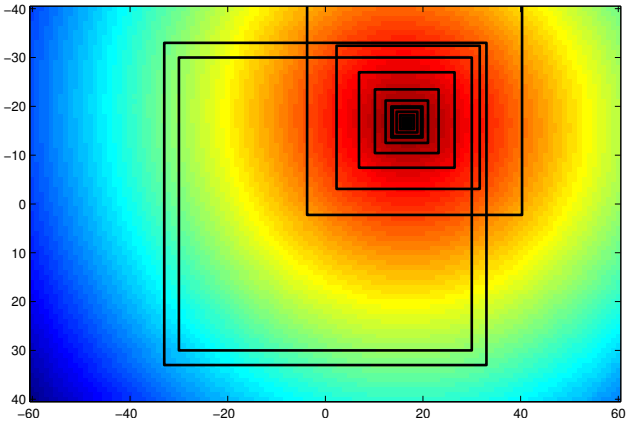

Figure 5. Minimum value search to find desired pan-tilt state using the predictive map. In this example, the first search step generates a displacement larger than 50% of the box side, and this causes the box to be expanded.

of the map is reached. This algorithm is a simplified successive interpolation algorithm [5]. This class of algorithms is easy to apply to regression outputs, since they do not require computation of derivatives. For the search to guarantee convergence more sophisticated point sampling, and adaptive size change of the search window are required, see e.g. Brent's method [5]. For the problem at hand however, this algorithm appears to be sufficient.

The minimum value search is currently implemented in Matlab, and takes on average 0.5 seconds to complete on a 3 GHz Pentium 4 system. Since the implementation relies heavily on `for`-loops, re-implementing this in C would speed it up by a factor $5 - 10$.

6. Experiments

6.1. Evaluation of Motion Prediction

In order to evaluate the algorithm used for learning the motion prediction map (3), we generated a trajectory of 300 saccades according to section 4, and trained the predictive map according to section 4.1. The 300 saccades gave a total of 31 715 input-output samples for (3). This data-set will be referred to as DS1.

A second data-set, DS2, was then generated from a different trajectory of 300 saccades (this gave us 29 525 samples). In order to see to what extent the learned mapping would generalise to another scene structure, we then moved the robot to a new location, with more nearby objects, and generated a third 300 saccade data-set, DS3, (with 20 448

Data set	x_l	y_l	x_r	y_r
DS1	2.2118	1.9747	2.2252	1.9905
DS2	5.1275	4.4339	5.1701	4.4349
DS3	6.6577	6.3325	6.6814	6.3254

Table 1. RMS pixel errors for $K = 16\,000$, $D = 40$, and $r = 1.0$.

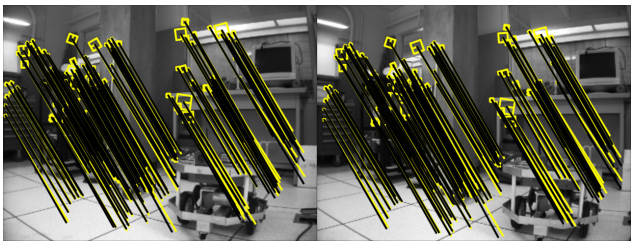

Figure 6. Predictive map output. A stereo pair with corresponding SIFT features before a saccade painted in bright. Bright lines show actual motion vectors generated by the saccade. Dark lines show the predicted motion vectors. The saccade shown has a pan change of $14°$ and a tilt change of $24°$.

samples).

The RMS errors on all three data-sets, after training on DS1, are given in Table 1. For these results, the relative kernel scale r for the weighted least squares was set to $r = 1.0$. This value was chosen by trying $r \in \{0.2, 0.4, \ldots 2.0\}$ and picking the one with the smallest RMS errors. A more flexible approach is to instead learn r from training data, as was done in [15]. To decide on the number of near neighbours D, we tested $D \in \{10, 20, 30, 40, 50, 60\}$ and selected $D = 40$ since increasing the number made little improvement.

To put the size of the errors in Table 1 into perspective, we have drawn the predicted motion vectors together with the actual motion vectors in Figure 6. As can be seen here, although the errors are a few pixels, the overall pattern is correct.

For number of prototypes we selected $K = 16\,000$. As the number of prototypes is increased, the errors on DS1 will continue to drop, but on the other data-sets it tends to level off and then increase slightly, with the minimum being approximately at $K = 16\,000$. This is shown in Figure 7. To easier see where the optimum lies, we averaged over all four coordinates and the data sets DS2 and DS3 to obtain the single curve in Figure 8, left. The fact that the errors are slightly lower for $K = 16\,000$ than when using all samples demonstrates that K-means actually serves a purpose other

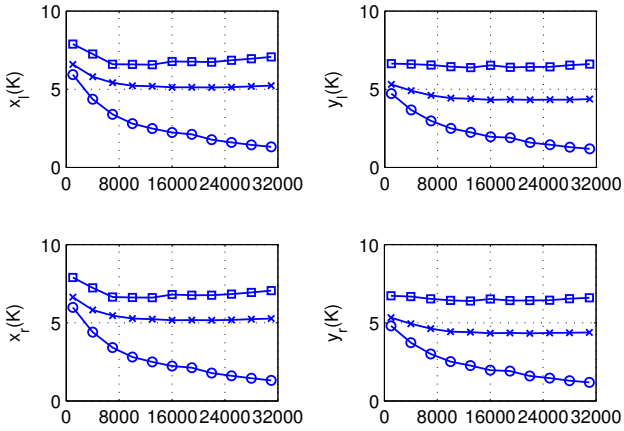

Figure 7. RMS pixel errors as function of #prototypes for all three data sets. The datasets are plotted as: DS1 circles, DS2 crosses, and DS3 squares.

than saving memory.

If we look at the errors for individual coordinates, see Figure 7, we can note that the errors in prediction of the vertical coordinates (y) is slightly smaller than the horizontal prediction errors (x). The reason for this is probably that most of the disparity changes are horizontal.

Since the errors for DS2 are significantly smaller than for DS3, which uses a different scene with more nearby objects, we can also conclude that the chosen prototypes do not fully separate disparity from position in the scene. To get rid of this effect, one would have to move the robot around as it is learning. In section 7 we will introduce a second, life-long tuning stage, that will allow the system to adapt to scene changes.

To demonstrate that 300 saccades is sufficient, we extended DS1 to 600 movements, and tried learning (3) with successively increasing numbers of actions between 50, and 600. All other parameters were kept constant. The result of this experiment is shown in Figure 8, right. As can be seen, the errors level off, and decrease much more slowly after about 300 saccades.

6.2. Evaluation of Centering Saccades

In order to evaluate the quality of the centering saccades obtained from the predictive mapping and the minimum value search, we let the robot try to center SIFT correspondences randomly selected from those currently available in the field of view. After the saccade we find the features again, and compute the centering error (2). This was repeated until 500 successful trials had been carried out. Dur-

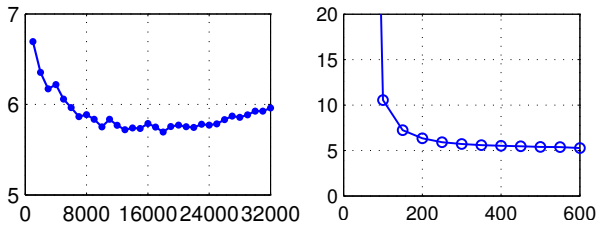

Figure 8. RMS pixel errors as function of #prototypes (left) and #saccades (right). The curves are averages of all four coordinates over the two data sets DS2**, and** DS3**.** $K = 16\,000$**,** $D = 40$**, and** $r = 1.0$**.**

ing this time, another 57 trials were carried out, but for these the SIFT pair was not found again after the saccade. The distribution of centering errors for the 500 successful trials, is shown in Figure 9, left. The right plot shows the distribution of gaze displacements $\mathbf{d} = \mathbf{s}(t) - \mathbf{s}(t+1)$. During the trials the min-value search diverged for 70 of the chosen SIFT correspondences, and another SIFT correspondence had to be selected for centering.

The maximal horizontal gaze shift is 320 pixels, which corresponds to approximately 35°. Thus a typical centering error of 5.5 pixels (the position of the peak in Figure 9, left) translates to about 0.60° gaze accuracy. As can be seen in Figure 9, right, the distribution of gaze displacements shows good coverage, with a slight preference for the centre. Good coverage is a desirable quality if these trials should be used for further tuning of the mapping.

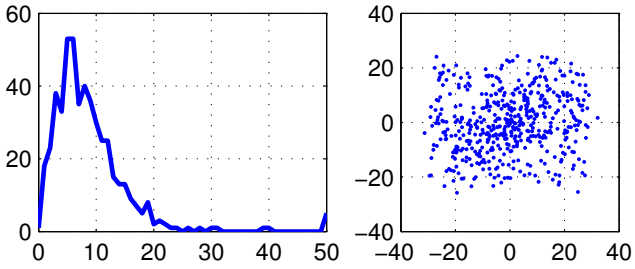

Figure 9. Evaluation of generated trials. Left: distribution of centering errors (in pixels) for generated trial actions. Right: Distribution of corresponding gaze displacements (in degrees) showing good coverage.

7. Life-Long Improvement

The memory based approach is actually well suited to life-long improvement. A simple way to continuously improve the centering saccade performance during use, is to incorporate all successful centering saccades into the motion prediction mapping. The most simple way of doing this would be to add the new observations to the prototype set.

To demonstrate that the new samples are beneficial for centering saccade accuracy, we repeated the experiment from section 6.2, after appending 500 new observations to the prototype set. The new observations were again obtained by letting the robot pick random SIFT correspondences and try to center them. The result of this is shown in Figure 10, left. As can be seen, the new observations helped to move the error distribution somewhat towards zero, and the most likely error is now 0.49° (4.5 pixel) instead of 0.60° (5.5 pixel).

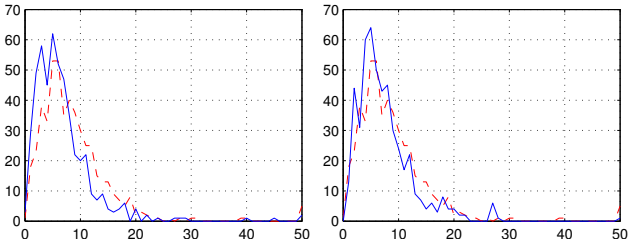

Figure 10. Evaluation of continuous improvement. Left: Distribution of centering errors (in pixels) after appending the 500 **trial actions to the prototype set. Right: Distribution of centering errors (in pixels) after letting the** 500 **trial actions replace the most similar prototypes. Dashed: The performance before adding new actions (same as in Figure 9, left).**

Just adding new prototypes will have the problem that the memory will eventually be filled, and thus we also tried letting each new observation replace the most similar prototype in the memory. The result of this is shown in Figure 10, right. As can be seen, there is still a significant improvement in performance, compared to before the prototypes were replaced. The reason for this is that the new observations have gathered in a different way. Since the mapping is going to be used for centering saccades, observations that do center a correspondence will probably be more useful than those obtained from the exploratory movements when learning the motion prediction map. Clearly some sort of prototype replacement strategy makes sense, but the strategy presented here is not necessarily the best one.

7.1. Cognitive Stages

In developmental sciences, the term *cognitive stages* refers to the idea that learning related skills can facilitate learning of a given difficult problem [16]. That is, to master a difficult problem, the agent first learns a simple but related problem, and then using the acquired skill learns the difficult problem. For instance in child development, the baby sequentially learns the competences *sit*, *roll over*, *crawl*, and *stand* before it learns to walk. Each of these competences can be viewed as *enabling* the following one, since its requirements on muscle strength, control and coordination are also needed in the next competence. In the same manner, our motion prediction stage can be viewed as enabling learning of saccadic gaze control.

8. Concluding Remarks

We have demonstrated how the competence of saccadic gaze control can be mastered by learning to predict how image features will move as the gaze is shifted. By attacking the problem in a different way we are able to collect many training examples in each action, and this results in a more than 100 times faster convergence than the previous approaches [9, 6, 12].

While the obtained accuracy of 0.60° after 300 saccades is sufficient for an accurate gaze, there might be applications where higher accuracy is needed. For such applications, the second learning stage, which continuously tunes the mapping, will be useful.

We have also demonstrated that learning of saccadic gaze control can be performed without requiring special calibration hardware [6, 3], a controlled environment [9], or visual tracking and slow camera motions [12]. Instead we can directly perform saccadic motions that center a given feature of interest in the stereo pair.

9. Acknowledgements

The author thanks Jim Little and Julia Vogel for helpful comments on earlier versions of this manuscript, and David Lowe for providing code for SIFT feature detection. This work was supported by the Swedish Research Council through a grant for the project *Active Exploration of Surroundings and Effectors for Vision Based Robots*.

References

[1] C. Atkeson. Using locally weighted regression for robot learning. In *IEEE Conf. on Robotics and Automation*, pages 958–963, Sacramento, CA, 1991.

[2] J. Beis and D. Lowe. Shape indexing using approximate nearest-neighbour search in highdimensional spaces. In *Conference on Computer Vision and Pattern Recognition*, pages 1000–1006, Puerto Rico, 1997.

[3] L. Berthouze and Y. Kuniyoshi. Emergence and categorization of coordinated visual behavior through embodied interaction. *Machine Learning*, 31(1-3):187–200, 1998.

[4] C. M. Bishop. *Neural Networks for Pattern Recognition*, chapter 5. Oxford University Press, 1995.

[5] R. P. Brent. *Algorithms for Minimization without Derivatives*. Prentice-Hall, 1973.

[6] J. Bruske, M. Hansen, L. Riehn, and G. Sommer. Adaptive saccade control of a binocular head with dynamic cell structures. In *ICANN'96*, pages 215–220, 1996.

[7] S. Feyrer and A. Zell. Tracking and pursuing persons with a mobile robot. In *Int. Workshop on Recongition, Analysis and Tracking of Faces and Gestures in Real-Time Systems*, Sept 1999.

[8] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.

[9] H. Hoffmann, W. Schenck, and R. Möller. Learning visuomotor transformations for gaze-control and grasping. *Biological Cybernetics*, 93(2):119–130, 2005.

[10] M. Jordan and D. Rumelhart. Forward models: Supervised learning with a distal teacher. *Cognitive Science*, 16(3):307–354, 1992.

[11] Y. Kuno, D. Miyauchi, and A. Nakamura. Robotic method of taking the initiative in eye contact. In *Conference on Human Factors in Computing Systems*, pages 1577 – 1580, Portland, Oregon, USA, 2005. ACM Press, New York.

[12] S. Léonard and M. Jägersand. Incremental learning for mapping image variations to actions. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 4235–4240, Barcelona, Spain, April 2005. IEEE.

[13] L. Ljung. *System Identification: Theory for the User*. Prentice-Hall, 1987.

[14] D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[15] D. G. Lowe. Similarity metric learning for a variable-kernel classifier. *Neural Computation*, 7(1):72–85, January 1995.

[16] M. Lungarella, G. Metta, R. Pfeifer, and G. Sandini. Developmental robotics: A survey. *Connection Science*, 15(4):151–190, December 2003.

[17] R. S. Sutton and A. G. Barto. *Reinforcement Learning, an Introduction*. MIT Press, Cambridge, Massachusetts, 1998. ISBN 0-262-19398-1.

[18] Y. Yoshikawa, K. Shinozawa, H. Ishiguro, N. Hagita, and T. Miyamoto. Responsive robot gaze to interaction partner. In *Proceedings of Robotics: Science and Systems*, Cambridge, USA, June 2006.