Example-Based Head Tracking

Sourabh Niyogi § and William T. Freeman †

Dept. of Electrical Engineering
and Computer Science §
M. I. T.
Cambridge, MA 02139 USA
e-mail: sourabh@media.mit.edu

MERL, †
a Mitsubishi Electric Research Lab.
201 Broadway
Cambridge, MA 02139 USA
e-mail: freeman@merl.com

From: IEEE 2nd Intl. Conf. on Automatic Face and Gesture Recognition, Killington, VT, October, 1996.

1 Abstract

We want to estimate the pose of human heads. This estimation involves a nonlinear mapping from the input image to an output parametric description. We characterize the mapping through examples from a training set, outputting the pose of the nearest example neighbor of the input. This is vector quantization, with the modification that we store an output parameter code with each quantized input code. For efficient indexing, we use a tree-structured vector quantizer (TSVQ).

We make design choices based on the example application of monitoring an automobile driver's face. The reliance on stored data over computation power allows the system to be simple; efficient organization of the data allows it to be fast. We incorporate tracking in position and scale within the same vector quantization framework with virtually no cost in added computation. We show reasonable experimental results for a real-time prototype running on an inexpensive workstation.

2 Motivation

We are interested in the general problem of analyzing people in images. We focus on a specific application: monitoring the position and orientation of a driver's face, viewed from inside the car. The goal of such monitoring is highway safety; most traffic fatalities are caused by driver errors, and cars could become safer if they are able to monitor the state of awareness of the driver [6].

This particular problem imposes many restrictions which challenge the algorithm designer. The algorithm must be robust to changes in person, appearance, and lighting, and the head position will be initially unknown. The algorithm should be fast, continuously updating the estimated pose approximately every 100 msec. To achieve mass-market sales, it must be very low-cost.

Present head tracking systems do not meet these restrictions. Many rely on the assumption of small head motions [8, 10, 11, 15, 20, 22] or require too much processing power [2] for the low-cost, real-time application we have in mind. The methods used are predominantly based on templates [7, 18], feature tracking [1], or optical flow [2, 3, 5, 20, 22].

We have built a system based on tree structured vector quantization (TSVQ) which may meet the cost and speed restrictions for the driving application. We have written and studied a real-time prototype on an SGI Indy workstation.

3 Mapping Inputs to Outputs

We want to learn a non-linear mapping from the input image, $\mathbf{x}$, to the output model parameters, $\mathbf{y}$. Recovering closed-form parametric solutions from first principles is unmanagable. For input spaces with complex structure, however, *learning* the mapping from inputs to outputs is more tractable. This has been applied to complex input spaces in problems of face detection and recognition [14, 19] and to pose estimation for simple figures [17, 16].

Learning the mapping from inputs $\mathbf{x}$ to outputs $\mathbf{y}$ requires a training set of example pairs $(\mathbf{x}_i, \mathbf{y}_i)$ that represent typical mappings. To obtain these mappings, two solutions are natural: (1) obtaining real input-output mappings, for example, by capturing image data $\mathbf{x}$ while capturing pose parameters $\mathbf{y}$; (2) obtaining synthetic input-output mappings by constructing a *synthesizer* which outputs images for a particular parametric description $\mathbf{y}$. For this work, we have used the first approach.

Given training data, the challenge is to summarize the samples so that a new example $\mathbf{x}$ whose output code $\mathbf{y}$ is unknown can be estimated. Estimation techniques can be either parametric or non-parametric. Parametric techniques assume an input-output mapping of some form $\mathbf{y} = f(\mathbf{w}, \mathbf{x})$, where the task is to estimate the parameters $\mathbf{w}$. Parametric techniques are practical when the dimensionality of the inputs $\mathbf{x}$ and parameters $\mathbf{w}$ is small, when the number of weights is small, and when the parametric form is roughly powerful enough to handle. For the complexities of human body pose initialization, these conditions may not be met.

Non-parametric estimation techniques include nearest neighbor (NN) estimation techniques, and use techniques closely related to vector quantization [9]. In NN techniques, k codes $\mathbf{m}_j = [\mathbf{x}_j, \mathbf{y}_j], j = 1, \ldots, k$ express the input-output mapping simply. Estimation is done by finding the nearest input given a distance metric $D(\mathbf{x}_1, \mathbf{x}_2)$:

$$j^* = \arg \min_{j=1,\ldots,k} D(\mathbf{x}, \mathbf{x}_j)$$

. The output is just the output component of the code vector $\mathbf{m}_{j^*}$:

$$\mathbf{y} = \mathbf{m}_{oj^*}$$

. Variants include choosing not the output for the nearest neighbor, but more generally some combination of the outputs of the K-nearest codes.

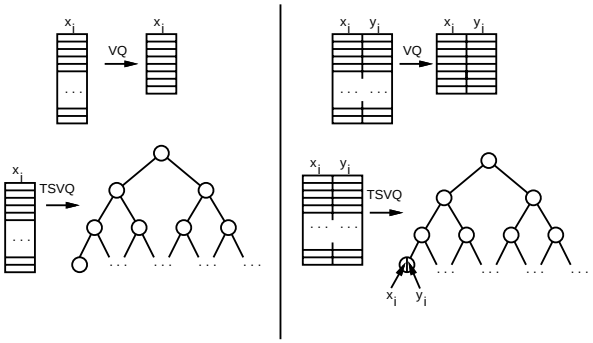

Figure 1: Left: Tree structured vector quantization (TSVQ), and, Right: our simple extension to it. In traditional VQ, many example inputs, (e.g., the images, $\mathbf{x}$), are quantized to a smaller set of possible values. For efficient indexing, TSVQ the quantized input values into a tree. At the right, we add an output code (the model parameter, $\mathbf{y}$) to each VQ quantized input code. Again, for efficient storage, we can organize the quantized codes into a tree. Each leaf node contains an input/output $\mathbf{x}$, $\mathbf{y}$ pair.

The distance metric is often Euclidean:

$$D(\mathbf{a}, \mathbf{b}) = (\mathbf{a} - \mathbf{b})^T (\mathbf{a} - \mathbf{b})$$

. We might anticipate some large outlier images to those in our database and can use a robust error measure instead [4].

NN classification is closely related to vector quantization (VQ): given a set of N dimensional inputs $\mathbf{x}_i$, produce a mapping onto one of k codes. The only difference is that the output is a not a code j, but a vector $\mathbf{y}$.

Learning the codes $\mathbf{m}_j$ is possible via a variety of approaches. One extreme solution is to do no learning at all - and just use the entire training set $(\mathbf{x}_i, \mathbf{y}_i)$ of Q examples to form the k codes $\mathbf{m}_j$, where $k = Q$. When Q is large, one must efficiently *index* into the large training set. (One can reduce the number of codes by clustering similar points into a common code [9]).

To reduce indexing costs, the data can be organized into a tree, shown in Figure 1. This technique of organizing codes is known as *tree-structured vector quantization* (TSVQ). At each node in the tree, a code is stored. Each node has a fixed number of children nodes, also with codes. In binary tree structured VQ, each node has 2 children. Indexing involves recursively choosing the child node which has the nearest code, until a leaf is reached. For our estimation problem, this yields an output code $\mathbf{y}_i$. This maintains a storage cost on the order of Q but reduces the indexing cost to $\log(Q)$.

Techniques to build tree-structured codebooks are intuitively easy to construct, and all naturally follow a recursive form. Consider binary TSVQ. Given a set of examples $(\mathbf{x}_i, \mathbf{y}_i)$, the simplest procedure is to split the examples into two piles. k-means clustering, where $k = 2$ might be used. Alternatively, more robustly, PCA can be used to find the direction of maximal variation (c.f. [21]). This is what we used, basing the principle component direction on that of a subsampling of all the example points, see Figure 2. Given a split of examples into two piles, each of those piles can be split, and then each of piles from those splits, etc. until the pile is too small for PCA or clustering to be applied, and table lookup makes sense instead.

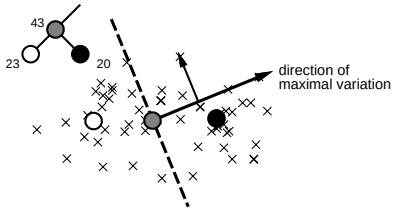

Figure 2: Binary tree-structured vector quantizers can be generated from a dataset by repeatedly: (a) computing the direction $\mathbf{e}_1$ of maximal variation of the dataset; (b) projecting each datapoint $\mathbf{x}_i$ onto the weight $w = \mathbf{e}_1^T$; (c) assigning each datapoint to the one of two children depending on whether $w > 0$ or $w < 0$.

4 Application: Head Pose Determination

We address the particular problem of tracking the position and pose of a driver's head in an automobile. We assume the camera is close enough to the face that we may use very low resolution images. The memory-intensive, computationally simple TSVQ approach is well suited to this particular problem, which requires fast, low-cost processing to be commercially successful.

To test the feasibility of this approach, we made a laboratory prototype of an automobile head tracker, using an SGI Indy workstation and 40×30 resolution imagery.

4.1 Acquisition of training data

We took images of 11 different subjects, in 15 different head poses each. We set 15 numbered markers on the wall, in a 3×5 grid, spaced $20°$ apart horizontally and $30°$ vertically. Inconsistant head poses in the training data can lead to a mis-classification of pose. (Of course, in the testing phase, the head pose is unconstrained, but it must be referenced back to the training poses at calibrated positions). Without physically constraining apparatus, it was difficult to obtain data for a calibrated set of head orientations. The approach we used was to seat the subject in a swivel chair, and instruct them to "point their chin" at each of a set of the 15 numbers. The swivel chair allowed fairly accurate control of the azimuthal head angle; subjects rotated the chair so that they faced each number squarely. Subjects were less consistant at head tilt. The photographer coached each subject to try to achieve a consistant amount of head tilt across subjects. Figure 3 shows a sampling of the resulting training set of 11×15 images.

We describe below our method to accomodate changes in the user's head position, but it requires training data at initially consistant head positions.

We put the training set into registration by shifting them so that the nose positions, identified with a mouse by an observer, coincided.

Figure 3: Samples of training images in the data set. 11 different people were used, each photographed at 15 different head orientations.

4.2 Head Tracking

For the pixel-wise comparisons between images to be meaningful, the test and training data must be precisely aligned. One could achieve this through several standard image processing techniques (e.g. [13, 12]), although at additional computational cost. An advantage of the TSVQ approach is that we can incorporate head tracking over position and scale in a very natural way. For each head in our data base, we generate synthetic examples (c.f. [3]) of the head at shifted horizontal and vertical positions, and scale. During run-time operation, if the head matches an example with a spatial offset, we simply adjust the cropping offset of a window around the head in an image until the head matches a centered example, as illustrated in Figure 4. This allowed us to use a tight, elliptical cropping window around each person's face, to minimize the influence of varying backgrounds.

The shift of the position of the cropping window is essentially a cost-free operation. An extra two images to accomodate a shift in each dimension of scale, horizontal, and vertical position increases the image storage requirements by a factor of 7. We thus achieve position tracking of the head at a cost of $\log_2(7) \approx 3$ additional image comparisons. Tracking in scale would involve the extra run-time cost of a bilinear image interpolation to stretch or squash the input image in scale.

4.3 Memory requirements

We can calculate the memory requirements of our prototype system. 11 heads $\times$ 15 poses $\times$ 7 images for tracking in position and scale $\times$ 40 $\times$ 30 pixels per image $\times$ 1 byte per pixel requires 1.4Mb storage for our prototype dataset.

A system to be installed in an automobile would require more data, although that could be offset by some additional care in data storage. One may need 25 subjects, each under perhaps 4 different lighting conditions. (We have yet to study the performance characteristics under variations in lighting). However, the actual cropped heads in our images were smaller than the full 40 by 30 image size; they were roughly 16×16 pixels. This gives a dataset storage size of, in the same order as above: $25 \times 4 \times 15 \times 7 \times 16 \times 16 \times 1 = 2.6$Mb. This memory requirement is consistent with a low-cost system at today's prices.

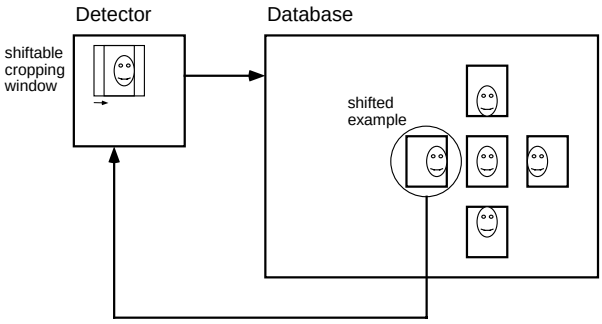

Figure 4: Head tracking method which is integrated with the main image matching loop. After training, we generate synthetic examples of spatially shifted (or shifted in scale) heads. If the best match to the input image is to one of the offset (or scale shifted) examples, we know to adjust the cropping window offsets until the best match head is one in a centered head position (not scale shifted).

4.4 Prototype performance results

With our calibrated training data, we evaluated the performance of the nearest neighbor method for tracking head pose, using a leave-one-out procedure. For each person in the training set, we removed their images from the training set and found the pose of the nearest neighbor head image. We did that for all poses and all people in the training set. The nearest neighbor method found the exact head pose 48% of the time. It found approximate pose (within one step horizontally or vertically) 87% of the time.

In our real-time system, we implemented left-right head tracking, as described in Section 4.2. This worked well to maintain the head centered within the cropping window.

To qualitatively evaluate the performance of our real-time set-up, we displayed the input camera image, the cropped image of the user's head, and the closest-match input code image, corresponding in our case to the head of the training set which had the smallest (robust) distance from the input image. These frames update on the SGI Indy in real-time at 11 frames per second.

We tested the system on people who were not in the training set. The identity of the closest match head constantly changed, but the pose of the best match generally matched well with the pose of the input image. Figure 5 is a random sampling of the system matches. The results are not perfect, but they are reasonable, showing that the TSVQ method can successfully initialize the pose of a human head from low-resolution images. Possible approaches to improving the results may include exploiting temporal consistancy constraints in the matches, or expanding the database of training images.

5 Summary

We have presented a simple approach to identify pose parameters associated with views of an object. Be-

cause of the high degree of nonlinearity of the mapping of input images to output parameters, we chose a non-parametric approach, based on tree structured vector quantization (TSVQ). As in standard TSVQ, we clustered and quantized the input vectors; to record the image to pose mappings, we also stored an output code with each quantized vector. The tree structuring allows for efficient indexing into the large dataset of training images. The system handles the nonlinear pose initialization problem, although the resulting system can be used as a tracker, as well.

We applied this approach to tracking the pose of the head. We made design choices based on a particular application of this problem, monitoring the position of automobile driver's face (toward the ultimate goal of monitoring driver awareness). We implemented a real-time prototype of this system, which achieved reasonable performance in monitoring the head pose of people not in the training set, at 11 frames per second. The TSVQ approach allows the computational burden to be very low, in effect trading memory for computation. Since the memory access is structured efficiently, one can build a fast and useful system using modest computational hardware.

6 Acknowledgements

Thanks to David Brogan and David Anderson for technical help.

References

[1] A. Azarbayejani, B. Horowitz, and A. Pentland. Recursive estimation of structure and motion using the relative orientation constraint. In *Proc. IEEE CVPR*, 1993.

[2] S. Basu, I. Essa, and A. Pentland. Motion regularization for model-based head tracking. In *Intl. Conf. on Pattern Recognition (ICPR '96)*, Vienna, Austria, 1996.

[3] D. Beymer and T. Poggio. Face recognition from one example view. In *Proc. 5th Intl. Conf. on Computer Vision*, pages 500–507. IEEE, 1995.

[4] M. J. Black and A. Rangarajan. On the unification of line processes, outlier rejection, and robust statistics with applications in early vision. *Intl. J. Comp. Vis.*, 19(1):57–91, 1996.

[5] M. J. Black and Y. Yacoob. Tracking and recognizing rigid and non-rigid facial motions using local parametric models of image motion. In *Proc. 5th Intl. Conf. on Computer Vision*, pages 374–381. IEEE, 1995.

[6] W. Bounds. Sounds and scents to jolt drowsy drivers. *Wall Street Journal*, page B1, May 3 1996.

[7] I. Essa, T. Darrell, and A. Pentland. Tracking facial motion. In *Proceedings of the workshop on motion of nonrigid and articulated objects*, pages 36–42. IEEE Computer Society, 1994.

[8] I. Essa and A. Pentland. Facial expression recognition using a dynamic model and motion energy. In *Proc. 5th Intl. Conf. on Computer Vision*, pages 360–367. IEEE, 1995.

[9] A. Gersho and R. Gray. *Vector quantization and signal compression.* Kluwer Academic Publishers, Boston, 1992.

[10] A. Lantis, C. j. Taylor, and T. F. Cootes. A unified approach to coding and interpreting face images. In *Proc. 5th Intl. Conf. on Computer Vision*, pages 369–373. IEEE, 1995.

[11] H. Li, P. Roivainen, and R. Forchheimer. 3-d motion estimation in model-based facial image coding. *IEEE Pat. Anal. Mach. Intell.*, 15(6):545–555, June 1993.

[12] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proc. Seventh IJCAI*, pages 674–679, Vancouver, 1981.

[13] B. Moghaddam and A. Pentland. Probabilistic visual learning for object detection. In *Proc. 5th Intl. Conf. Computer Vision*, pages 786–793. IEEE, 1995.

[14] B. Moghaddam and A. Pentland. Probablistic visual learning for object detection. In *Proceedings of the Fifth Internation Conference on Computer Vision*, page ?, Cambridge, MA, 1995.

[15] Y. Moses, D. Reynard, and A. Blake. Determining facial expressions in real time. In *Proc. 5th Intl. Conf. on Computer Vision*, pages 296–301. IEEE, 1995.

[16] H. Murase and S. Nayar. Visual learning and recognition of 3-d objects from appearance. *International Journal of Computer Vision*, 14:5–24, 1995.

[17] T. Poggio and S. Edelman. A network that learns to recognize three-dimensional objects. *Nature*, 343:263–266, 1990.

[18] A. Saulnier, M. L. Viaud, and D. Geldreich. Real-time facial analysis and synthesis chain. In M. Bichsel, editor, *Intl. Workshop on automatic face- and gesture-recognition*, pages 86–91, Zurich, Switzerland, 1995. Dept. of Computer Science, University of Zurich, CH-8057.

[19] K. Sung and T. Poggio. Example-based learning for view-based human face detection. A.I. Memo 1521, Artificial Intelligence Lab, MIT, Cambridge, MA 02139, 1994.

[20] D. Terzopoulus and K. Waters. Analysis and synthesis of facial image sequences using physical and anatomical models. *IEEE Pat. Anal. Mach. Intell.*, 15(6):569–579, June 1993.

[21] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):0, 1991.

[22] Y. Yacoob and L. Davis. Computing spatio-temporal representations of human faces. In *Proc. IEEE CVPR*, pages 70–75, 1994.

Figure 5: Screen display of our prototype system, illustrating its performance. The raw camera image is on the left. At right, two heads are displayed. The left one is the centered (by the memory-based tracking) and cropped face from the camera. The right one is the leaf node of the VQ tree which most closely matches the input image. The matches occur at 11 per second, these single frames digitized from a videotape are random draws from the set of matches made by the system. A few matches correspond to erroneous head poses, but most matches are in good agreement with the head pose of the input image (not in the training set).

Figure 6: Continuation of figure at left.