Recognizing Faces with a Transputer Farm

Martin Lades, Jan C. Vorbrüggen, Rolf P. Würtz

Institut für Neuroinformatik, University of Bochum, FRG

Abstract

We present an OCCAM programme that recognizes human faces from video images. It implements a simplified version of the dynamic link architecture (DLA), a neural net paradigm first proposed in 1981. Objects are represented by labeled graphs, where vertex labels are local power spectra derived from the 128×128 pixel gray-level image and edge labels are the distance vectors of vertices. Matching is achieved by minimizing a similarity function between possible image graphs and the graph representing a candidate object by simulated annealing.
The programme has been implemented on a PC hosted Parsytec system with up to 31 T800 transputers and a combined frame grabber/display board. Feature extraction and matching processes are parallelized. The programme is built on top of our general-purpose farm and RPC software and uses some specialized transputer facilities (e. g., `MOVE2D`) to achieve good scaling of performance. Additionally, some core procedures were written in assembler to speed up the matching process twofold. The programme reliably recognizes a face from a database of 80 in about 30 seconds.

1 System Description

The system is a simplified implementation of the dynamic link architecture (DLA). We will only give a brief overview of the dynamic link architecture and labeled graph matching here; for in-depth discussions see [1, 2, 3]. DLA exploits the fine scale temporal structure of neural signals to allow complex feature binding. It provides a massively parallel tool for object recognition [4, 5].

For the implementation on a transputer network simplifications had to be made. Sets of neurons with receptive fields centered at the same point but of different shape are represented by nodes of a graph (simply the corresponding point) and labeled with their activity vectors. Bindings between them are replaced by the labeled edges in this graph. This reduces the problem of object recognition to one of labeled graph matching. The following two sections detail the steps required to accomplish the object recognition task.

1.1 Data Acquisition and Preprocessing

The first step captures a 512 × 512 pixel snapshot with a CCD camera at 8 bits (256 gray levels) of resolution. The image is then sampled down to 128 × 128 pixels. As motivated by receptive field properties, this image $I(\vec{x})$ is convolved with a bank of DC free, complex-valued Gabor-based wavelets $\psi_{\vec{k}}(\vec{x})$ (bandpass filters) parametrized by their spatial frequency $\vec{k}$. In our case 6 frequency levels and 8 orientations seemed sufficient according to power spectrum analysis for faces and biological data [6]. The

Fourier transform of the filter functions are given by the following equation:

$$\left(\mathcal{F}\psi_{\vec{k}}\right)(\vec{\omega}) = \exp\left(-\frac{\sigma^2\left(\vec{\omega}-\vec{k}\right)^2}{2k^2}\right) - \exp\left(-\frac{\sigma^2\left(\vec{\omega}^2+\vec{k}^2\right)}{2k^2}\right)$$

where the second term makes $\psi_{\vec{k}}$ DC free. $\vec{k}$ is restricted to

$$\vec{k} = \begin{pmatrix} k\cos\phi \\ k\sin\phi \end{pmatrix};\ k = \pi(\sqrt{2})^{-\kappa}, \kappa = 2\ldots7;\ \phi = \nu\frac{\pi}{8}, \nu = 0\ldots7.$$

A vertex $\vec{x_0}$ is labeled with the vector of the absolute values of the responses $\mathcal{J}I(\vec{x_0})$:

$$(\mathcal{J}I)(\vec{k},\vec{x_0}) := \left|\int \psi_{\vec{k}}(\vec{x_0}-\vec{x})\, I(\vec{x})\, d^2x\right|$$

This choice of vertex labels guarantees some robustness against local distortions.

To add a new face to the database, we extract and store the $\mathcal{J}I$ at the points of a 8×10 grid, reducing the precision to fit a byte for each filter response. This compresses the data by a factor of approximately 4 from the original 128×128 images.

Graph edges are labeled with the distance vector $\vec{\Delta}$ between their endpoints in order to ensure that similar vertex labels in noncorresponding regions of the image are not identified during matching.

1.2 Matching

The recognition step is a two stage matching process. In both stages two graphs are compared by the weighted sum of comparison functions of the two types of labels (E and V are the edge and vertex sets, respectively):

$$C_{total} := \lambda \sum_{(i,j)\in E} \mathcal{S}_e\left(\vec{\Delta}^I_{(i,j)}, \vec{\Delta}^O_{(i,j)}\right) - \sum_{i\in V} \mathcal{S}_v\left(J^I_i, J^O_i\right)$$

For $\mathcal{S}_v$ we chose the cosine of the angle between the jets to achieve some degree of intensity independence. For $\mathcal{S}_e$ we used the squared difference of the edge labels. This yields a cost function for a pair of image and object graphs which is to be minimized.

In a first step, the approximate location of the face in the image is determined by comparing it with one fixed object. For that purpose C_{total} is minimized varying only the center of the graph and leaving its geometrical shape unchanged. This estimate initializes the second step, where for every object graph in the database C_{total} is minimized varying all vertex positions in the image graph independently. After convergence we have a final cost value for each object; the smallest of them belongs to the recognized person if it is significantly different (according to some statistical criteria) from the other matches. Both optimizations are done by simulated annealing at zero temperature, for which the cost landscapes are sufficiently well behaved [5].

It is also possible to start with less than the full number of frequency levels and vertices and then to increase both during the matching process. We also implemented size invariance exploiting the log-polar form of the vertex labels [7].

Figure 1: This figure shows the overall structure of our OCCAM recognition system. Each of the boxes denotes one T800 transputer, the arrows transputer links. Each ellipsis stands for a concurrent process, lines interconnecting them represent pairs of OCCAM-channels. Only the black processes `User.M` and `User.W` are application specific. At the top left is the *host* transputer that runs the TDS and gives access to PC resources. Only a standardized I/O server but no application code is run here, as errors occuring on this processor require a reboot of the TDS and thus are hard to debug. A software system (dark grey processes) based on remote procedure calls allows the user to access all channels to the PC from `User.M` as if executing on the host transputer. It comes with a virtual terminal handler that allows switching between different keyboard/screen pairs at runtime. This is necessary because parallel processes cannot write to the same terminal channel in a controlled way. One virtual terminal can be reserved to run a DOS loop, i.e., DOS commands or PC programmes can be executed while the transputer application is running — very useful for file manipulation. The existence of the router indicates that this system can be extended to more processors when desired.
The processes in light grey constitute the farm system. There is one *manager* processor and any number of *worker* processors. They can be interconnected in any tree structure with the manager at the root of the tree. The user provides a process that splits the problem up into subtasks and another one that collects the results. Both run in parallel and are part of `User.M`. The subtasks themselves are executed in `User.W`. The monitor processes allow a simple evaluation of load balancing.
The TFG (bottom left) is a specialised processor that drives the video camera and graphics display. It runs a standardized programme which provides flexible acquisition of video images and the graphical display of results.

2 Implementation

Figure 1 gives a detailed view of the hard- and software of our system. It has three main components that can be parallelized:

(i) the fast Fourier transform of the image data, performed as two sets of onedimensional FFTs on the rows and columns of the data array,

(ii) computation of the wavelet transform using the result of (i), and

(iii) comparison of the transformed image with the stored objects.

All of these are data-parallel, i. e., they can be divided into tasks operating on independent pieces of data, and as such are amenable to parallelization on a processor farm. These three operations are, however, quite different in the balance of computation and communication (with (i) being communication and (iii) computation bound) and in the amount of global data required ((i) requiring none, (iii) using the whole set of transforms of the image).

2.1 Routing and Farm Subsystem

Because of these differing requirements, we chose to implement a general-purpose farm system (see figure 1). At the lowest level, a set of router processes running at high priority connects the processors, with two additional features:

- The network supported can be any type of (regular or irregular) tree, with the farm controller at its root. The routing tables are built when the network is booted by listening on all but the boot link for a message. It is thus possible to reconfigure the network (e. g., add or remove processors) without recompiling.

- The routers can send messages by the farm controller to either a designated processor or broadcast them to all processors.

Messages are sent as number of packets, which have a fixed maximum size. Currently, the application (`User.M` and `User.W` processes in figure 1) has to perform the splitting and reassembly of messages larger than the packet size.

At the next level is the farm software itself, which consists of the manager and a buffer manager on each worker processor. This allows double buffering of tasks in the workers, so that they can immediately continue working after delivering a result. The manager communicates with the user program via channels. Usually, the user will first broadcast some amount of global data and then start a run, i. e., perform one of the above operations. At this point, both the manager and the user process split into two parallel processes, one set generating tasks and the other collecting results. This allows optimal use to be made of the computational resources of the manager processor, which for large numbers of processors ultimately becomes the bottleneck. The necessary synchronization (waiting for all results to come in before continuing) is done implicitly by the farm software; the user only signals that no more tasks are available. Provision is made for results requiring more than one packet, while tasks passed to the workers have to fit into one packet.

N	Elapsed Time (s)	Controller Load (%)	Worker Load (%)	Relative Speedup
1	148.2	0.7	99.9	1.00
2	74.7	1.5	99.8	0.99
4	38.1	3.1	99.7	0.97
6	25.9	4.5	99.4	0.95
8	19.8	5.9	99.0	0.94
16	10.9	11.6	95.0	0.85
24	8.1	16.0	88.0	0.76

Table 1: Performance information of the Morlet transform as a function of the number of worker processors. Only numbers of processors which are a divisor of the number of tasks (48) are listed. The programme was run on 20 MHz T800s with 3 cycle memory, configured as a binary tree on a parsytec MultiCluster II system. Timing was obtained using the transputer timer. Load information was collected by the simple monitor described in the text, which when sampling every 3 ms introduces an overhead of less than 1%.

2.2 Optimizations

In order to achieve good speedup, we have invested some effort into using special features of the transputer. It turns out that the farm controller becomes a bottleneck when inserting data into result arrays. For instance, a twodimensional FFT is performed by performing onedimensional FFTs on all the rows of the data and then a similar set of FFTs on the columns. When accepting the results of the first phase, the OCCAM compiler automatically generates a block move. For the second phase, however, data elements must be stored in non-contiguous elements of the array; doing this with a simple loop leads to a large load on the manager processor. In cases like these, the `MOVE2D` instruction of the T800 can be used to good advantage, as it will move arbitrarily sized elements which can be addressed by a constant stride.

The success of this modification was verified by a small monitor process running at high priority, which periodically checks for an interrupted low priority process, thus giving an indication of processor load (see also table 1).

During the recognition step, which minimizes C_{total} by simulated annealing, most of the work goes into computing $\mathcal{S}_v$. This requires the computation of scalar products and the rescaling of the filter responses from byte to floating point representation, a function not contained in popular vector libraries. We therefore rewrote these procedures in embedded assembler. We obtained an improvement factor ≈ 3 for the scalar product and ≈ 2 for this part of the programme over the naïve OCCAM version; unrolling the OCCAM loops only yielded a speedup of ≈ 1.4 [8].

Finally, we show in table 1 some performance measures for the computation of the complete Morlet transform on a 128×128 image. This involves

- computing the FFT of the image data in parallel;
- broadcasting the result to all processors;
- performing the 48 convolutions in parallel; each task here consists of multiplying the data with the convolution kernel, transforming the result back into the space

domain and reducing the precision.

As can be seen from the table, we obtain very good speedup up to 16 processors. Adding further processors still yields an improvement, but the effect of the serial part of the programme (e. g., broadcasting data and setup times) is being felt. Furthermore, monitoring information indicates that the communication of results to the manager processor is becoming a bottleneck. If a much larger transputer system were available, we would either have to increase problem size (for instance by using 512×512 pixel images) or consider a different method of parallelization.

Acknowledgements: We are grateful for help from our colleagues at Bochum and Los Angeles. This work was funded by grants from the German Federal Ministry of Science and Technology (ITR-8800-H1), from AFOSR (88-0274), and by the Stimulus Programme of the European Community (BRAIN).

References

[1] Christoph von der Malsburg. The correlation theory of brain function. Technical report, Max-Planck-Institute for Biophysical Chemistry, Postfach 2841, Göttingen, FRG, 1981. Reprinted 1994 in: Domany, E., Schulten, K., and van Hemmen, J.L. (eds.), *Models of Neural Networks*, Vol. 2, pages 94–119, Springer Verlag.

[2] Christoph von der Malsburg. How are nervous structures organized? In E. Başar, H. Flohr, H.Haken, and A.J. Mandell, editors, *Synergetics of the Brain, Proceedings of the International Symposium on Synergetics*, pages 238–249. Springer, Berlin, Heidelberg, New York, May 1983.

[3] Christoph von der Malsburg. Pattern recognition by labeled graph matching. *Neural Networks*, 1:141–148, 1988.

[4] Joachim Buhmann, Jörg Lange, Christoph von der Malsburg, Jan C. Vorbrüggen, and Rolf P. Würtz. Object recognition with Gabor functions in the Dynamic Link Architecture — parallel implementation on a Transputer network. In B. Kosko, editor, *Neural Networks for Signal Processing*, pages 121–159. Prentice Hall, Englewood Cliffs, NJ, 1992.

[5] Martin Lades, Jan C. Vorbrüggen, Joachim Buhmann, Jörg Lange, Christoph von der Malsburg, Rolf P. Würtz, and Wolfgang Konen. Distortion invariant object recognition in the dynamic link architecture. *IEEE Transactions on Computers*, 42(3):300–311, 1993.

[6] J.P. Jones and L.A. Palmer. An evaluation of the two-dimensional Gabor filter model of simple receptive fields in cat striate cortex. *Journal of Neurophysiology*, 58(6):1233–1258, 1987.

[7] Joachim Buhmann, Martin Lades, and Christoph von der Malsburg. Size and distortion invariant object recognition by hierarchical graph matching. In *Proceedings of the IJCNN International Joint Conference on Neural Networks*, pages II 411–416, San Diego, June 1990. IEEE.

[8] J. C. Vorbrüggen. Parallelverarbeitung in Hardware: Optimierung numerischer Routinen auf dem T800. In R.Grebe, editor, *Transputer-Anwender-Treffen 1990*. Springer, 1991.