# Stereo Vision and 3D Reconstruction on a Distributed Memory System

N.H.L. Kuijpers
TNO Physics and Electronics Laboratorium
The Hague, The Netherlands

G. Paar
Institute for Digital Image Processing
Joanneum Research, Graz, Austria

J.J. Lukkien
Dept. of Mathematics and Computing Science
Eindhoven University of Technology, The Netherlands

**Abstract**

An important research topic in image processing is stereo vision. The objective is to compute a 3-dimensional representation of some scenery from two 2-dimensional digital images. Constructing a 3-dimensional representation involves finding pairs of pixels from the two images which correspond to the same point in space. Several stereo matching algorithms are developed to find matching pairs. Hierarchical matching can be applied to improve the effectiveness and reliability of the matching algorithms. Hierarchy is established by multi-resolution images, also known as pyramids.
Pixel based matching algorithms require lots of processing power and memory space. However, stereo vision applications are often subject to strict real-time requirements. In order to meet timing constraints, a multiprocessor implementation of the hierarchical matching algorithm is realized. The parallel algorithm is implemented on a distributed memory system on top of a communication library. The resulting system is integrated in a prototype apparatus for surface measurements during the construction of tunnels.

*Keywords:* stereo matching, 3D reconstruction, parallel computing, embedded systems

## 1 Introduction

Stereo vision is an important research topic in the image processing field. A 3-dimensional model is created by processing two 2-dimensional images. Usually these images are generated by CCD cameras. Applications are found both in industry and space research. A robust stereo vision algorithm which can be used for all kinds of applications has been developed at Joanneum Research in Graz, Austria.

In this paper it is described how a parallel implementation of stereo matching is realized on a processor network using a communication library. It is assumed that the processor network is a distributed memory system and that exchange of information between nodes can only be established by passing messages along communication links. Several processes can be executed by one node. The communication library abstracts from the low-level physical detail of the processor network. Information between processes is passed through channels or, in case they are executed by the same node, through shared memory. In the latter case, synchronization is realized by means of semaphores.

A system is introduced which uses stereo vision for 3-dimensional (3D) surface reconstruction of tunnel walls [1]. We focus on the hardware and software issues which are crucial in terms of the real-time requirements imposed by the tunnel construction process. To meet all requirements in terms of price, performance and accuracy as well as physical robustness, a parallel implementation of the stereo matching algorithm had to be developed.

The paper is organized as follows. In Section 2 the concept of communication libraries is introduced and a few examples are mentioned. In Section 3 methods for stereo matching are introduced and in Section 4 it is explained how a parallel algorithm for stereo matching is implemented on top of a communication library. Applications for parallel stereo matching are named in Section 5 and the prototype system in which the algorithm is embedded is described in more detail. Finally, conclusions are drawn in Section 6.

## 2 Communication Libraries

A parallel program can be regarded as a collection of *communicating sequential processes* [2]. Communication between the processes in the program is performed through channels or, if efficiency requires, through shared memory. The processes and channels are mapped onto the processors and the physical communication links, respectively. In general this implies multiplexing of both the processors and the links: a processor may have to execute several processes and more than one channel may be mapped onto a link. In addition, the mapping should be such that processes sharing their memory are mapped onto the same processor.

A communication library hides these mapping details from the programmer. A coherent programming model is provided: the processor network is a collection of machines each capable of executing a number of processes. Processes executed by the same machine share their memory and may synchronize through this memory, e.g. through the use of semaphores. An arbitrary pair of processes can communicate by using channels. Communication along these channels may be synchronous (not-buffered, blocking the sender) or asynchronous (buffered, not blocking the sender).

An important aspect of the use of a communication library is the potential for increased portability. In order to port a parallel program from one architecture to another, only the source code of the communication library has to be adapted. A requirement is that the implementation of the library is efficient. The extra overhead for one communication should be small; in particular, the latency of communications should be low.

The collection of processes is not static: processes are stopped and new processes are created. When the creation of a new process is initiated by a process executed by another node, we speak of a *Remote Process Call*. In case of a remote function invocation, we speak of a *Remote Procedure Call*. In this paper, we will show that we can implement stereo matching on a processor network if communication is possible between any pair of processes, and that the availability of a Remote Process Call or a Remote Procedure Call mechanism simplifies the design significantly.

The use of Remote Procedure Calls results in an abstraction of communications. If RPC's are used frequently, as a regular communication primitive, the mechanism must again be implemented efficiently with only a small overhead. RPC's capable of using the global memory on each processor are not common for distributed memory systems. They can be implemented easily if the following conditions are satisfied:

1. The system is a *Single Program Multiple Data* (SPMD) system, which means that on each node the same program runs with different data.

2. The processes on one node have shared memory access.

Three examples of communication libraries that support Remote Procedure Calls are NX, CMMD and ECL. The NX *message passing interface* [3] is built on a kernel of four basic message passing functions. Among these are synchronous, typed send and receive actions. On top of this kernel, asynchronous message passing is implemented to perform communication and computation in parallel. Asynchronous message passing is also used to implement RPC's by so-called *Handler Calls*.

The CMMD *message passing library* [4] implements Remote Procedure Calls for the CM-5. This system supports *active messages.* An active message consists of a pointer to a function followed by its arguments. The receiving process invokes this function with the communicated arguments. An advantage of using active messages instead of the common send/receive pair is the reduced software overhead necessary for synchronization and avoidance of deadlock. This software overhead increases the latency, which is especially bad in case of small-sized messages.

A third communication library which supports Remote Procedure Calls is the *Eindhoven Communication Library* (ECL) [5]. ECL was developed at Eindhoven University of Technology in the Netherlands. Originally, it was implemented as a small communication library for a transputer network. ECL supports Remote Process Calls (similar to active messages) and a simple form of Remote Procedure Calls. A useful application of Remote Procedure Calls is, for instance, the storing of a value in the local memory of another node ("remote write") or the reading of a value from the memory of another node ("remote read").

We have chosen for the Eindhoven Communication Library to implement stereo matching on a distributed memory system for several reasons. Firstly, the library is small and causes little overhead during communication. Secondly, the ECL interface is quite portable; implementation for other parallel machines such as the Silicon Graphics Power Challange and the Parsytec Power Xplorer have already been developed. Thirdly, the transputer source code was available which made porting to our hardware straightforward.

# 3 Stereo Matching

Reconstructing the 3D scene from a set of stereo images involves finding pixels from both images which correspond to the same point in the underlying scene. Two equally sized images of the scenery are generated and stored in a computer memory as two grids of pixels. The value of a pixel is a grey-level. To distinguish between the two images, one is called the *reference image* and the other the *search image*. Matching is performed to compute for each reference pixel the search pixel that corresponds best. From the result, the *disparity map* is constructed. It maps each reference pixel onto the difference in coordinates with its corresponding search pixel. Most matching algorithms are based on the fact that two corresponding pixels have corresponding neighborhoods. Several methods are developed to use correspondence of neighborhoods for matching. We name three that can be applied if the *smoothness constraint* is satisfied, i.e., the respective terrain heights at neighboring pixels do not differ too much from each other.

A first method for matching is to replace all pixel-values by a value that characterizes their neighborhood, a so-called *feature-value*. The search pixel whose feature-value is closest to the feature-value of the reference pixel is chosen as a candidate. An example of a feature-value of a pixel is the local variance. Generalizing this method to multiple features leads to the method of *Feature Vector Matching*.

A second method is the following: enumerate the neighborhood around each pixel as a sequence of pixel-values; this is done in the same way for all pixels. Now, choose that search pixel whose sequence resembles the sequence of the reference pixel best. In order to compare two sequences, a rate of similarity between them is defined by means of *cross correlation*. A comparison between Feature Vector Matching and the more traditional correlation method is presented in [6]. It is concluded that FVM results in a better disparity map than correlation matching.

A third method matches a row of pixels at the same time. This is performed by the so-called *Optimal Correspondent Subsequence Algorithm* [7]. However, this method can only be applied in case of epipolar geometry. Good results have been achieved using this algorithm for the matching of SPOT satellite images.

Since Feature Vector Matching results in very accurate disparity maps, this method is chosen as basis for parallel stereo matching.

## *3.1 Feature Vector Matching*

A *feature* of a pixel is a value which depends on the grey-levels of the pixels within the neighborhood of the pixel. If we calculate the feature for all pixels of an image, we get a so-called *feature image*. We present a method to match pixels by comparing a number of features. Selected features are the local average, a number of horizontal and vertical edge operators and the local variance. See [8] for a list of commonly used features. Most of them are described as *convolutions* or can be approximated by means of convolutions. Applying a convolution on an image means that for each pixel a value is computed which depends on the grey-levels of the surrounding pixels and the convolution *kernel*. The result of computing a convolution is another image containing for each pixel its convoluted value. An example of a feature is the local $13 \times 1$ highpass horizontal edge operator. The kernel of this feature is described as

$$(2\ 1\ 0\ \text{-}1\ \text{-}2\ \text{-}1\ 0\ 1\ 2\ 1\ 0\ \text{-}1\ \text{-}2).$$

Suppose there are $m$ features. If we put all features of one pixel together in a vector, we get the *feature vector* $\vec{f}$ for this pixel. From the contents of the feature images we can derive this vector for each pixel of the stereo image pair. Finding a match is performed by comparing each feature vector of the reference image, the *reference vectors*, to all feature vectors of the search area which is a part of the search image. We name the reference image and the search image $r$ and $s$, respectively, and the 'images' consisting of the corresponding feature vectors $\vec{r}$ and $\vec{s}$, respectively. Then, for a point $p$, $\vec{r}(p)$ is the feature vector of $p$ in the reference image and $\vec{s}(p)$ the feature vector of $p$ in the search image. The $l$th component of a vector $\vec{f}$ is denoted by $\vec{f}_l$.

In order to compare a reference vector to a search vector, the *feature distance* between the two vectors is computed. The feature distance is defined such that each component of the vectors is weighted. If we denote the weight of feature $l$ as $w_l$, then the feature distance between the vectors $\vec{f}$ and $\vec{g}$ is defined as

$$|\vec{f} - \vec{g}| = \sqrt{\frac{\sum_{l=1}^{m}((\vec{f}_l - \vec{g}_l) \cdot w_l)^2}{\sum_{l=1}^{m} w_l{}^2}}.$$

Computing, for a point $p$, the distance between $\vec{r}(p)$ and *each* vector in the search image is in general too expensive. We assume that we can restrict ourselves to a certain *search space* $\sigma_p$. This search space is defined by the search area, i.e., the center $(i,j)_p$ (which is assumed to be given) and the extensions $\delta_h$ and $\delta_v$ (which are the same for all points):

$$\sigma_p = \{\ \vec{s}(q) \mid q \in [i - \delta_h, i + \delta_h] \times [j - \delta_v, j + \delta_v]\ \}.$$

For a point $p$, best correspondence is found at position $q$ in the search space, where the distance between the reference vector and the search vector is minimal, i.e.,

$$|\vec{r}(p) - \vec{s}(q)| = \min_{\vec{f} \in \sigma_p} |\vec{r}(p) - \vec{f}|.$$

Feature Vector Matching is depicted in Figure 1. The algorithm is split into three parts:

1. Create feature images for both the reference and the search image.
2. Compare each reference vector to all search vectors of the search space. Best correspondence is found where the feature distance is minimal. The difference in $x$ and $y$-coordinate is stored as disparity vector. If the minimum feature distance exceeds a given threshold, the correspondence is invalid and the reference pixel is not matchable. As a result, the disparity for the reference pixel remains undefined.
3. Remove errors and interpolate undefined disparities.

## *3.2 Hierarchical Matching*

In order to evaluate the center of the search area for each point and to improve robustness and efficiency of the matching algorithm, *pyramids* of the input images are generated [9]. Level 0 of the pyramid is the original image. To create the next level, the average grey-level of four

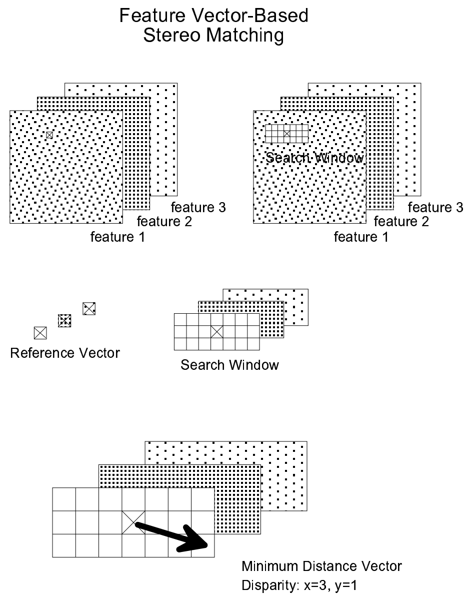

Figure 1: Feature Vector Matching.

pixels in a square is computed and stored as one pixel in a new image. Matching starts at the top level of the pyramid with large search areas for each pixel. The resulting disparity map is filtered and undefined disparities are interpolated, before it is used as input initial disparity map (defining the centers of the search areas) for matching the next, lower, level of the pyramid.

To assure that the disparities from the left to the right image are of good quality, matching from right to left is performed as well. This is called *backmatching*. On each point $l$ of the left image, the left disparity map is applied. The result is $r$. Next, the right disparity map is applied on $r$ resulting in $l'$. The match is defined invalid when the distance between $l$ and $l'$ exceeds one pixel.

Incorporating pyramids, backmatching and filter algorithms leads to *Hierarchical Feature Vector Matching* or HFVM [8]. HFVM is developed at Joanneum Research in Graz, Austria. The algorithm can be described as follows: for both images of the stereo image pair perform the following steps:

1. Build the pyramid.
2. Compute the feature images for each pyramid level.
3. Match the top level of the pyramid.
4. Filter the resulting disparity map.
5. Check matching consistency by backmatching.
6. Interpolate the undefined disparities.
7. Use the resulting disparity map as initial disparity map to match the next, lower, pyramid level.

Steps 4 through 7 are repeated till a disparity map of level 0 is computed.

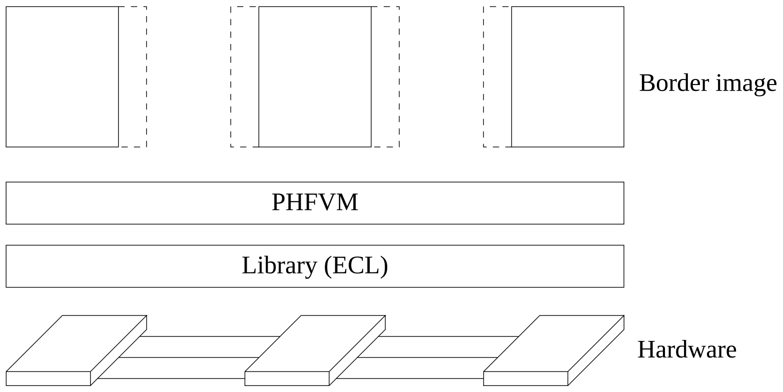

Figure 2: Architecture of parallel HFVM.

## 4 Distributed Memory Implementation

A distributed memory implementation of HFVM is realized on a network of TMS320C40 digital signal processors. As a first step, the Eindhoven Communication Library was ported to the chosen processor network. Next, a parallel implementation of HFVM was realized on top of this communication layer.

Storage of the feature vectors for each pixel of both input images requires lots of memory space. Therefore, data parallelism is applied: both images are split up in horizontal slices and spread out over the nodes of the processor network.

On top of ECL we developed a parallel program that exchanges image lines between nodes. Using this program, a *border image*, i.e., an extension of the local part of the image with data from neighboring nodes, is created. In this way, all data required for filtering is available in the local memory of the nodes and filter algorithms become sequential programs. Border images are used to perform the following filter operations: convolution, local variance, median filtering and interpolation. Formal methods were used to derive algorithms for all filter operations. These derivations resulted in efficient programs while serving as correctness proofs at the same time [10, 11].

Parallel Feature Vector Matching was implemented directly on top of ECL. Asynchronous and synchronous channels as well as Remote Procedure Calls were used for data transport between nodes. In Figure 2 it is depicted how a layered software architecture is designed to implement HFVM on a distributed memory system [12].

### *4.1 Parallel Feature Vector Matching*

Feature Vector Matching requires access to data from both the reference and the search image. Cooperation between the node that tries to find a match for a certain pixel and the nodes that have access to the corresponding search space has to be established. The implementation is far from trivial since it is not known in advance which nodes have to cooperate. Feature Vector Matching is performed by four different processes running on each node. The *main* process tries to find a match for each pixel of the local part of the reference image. It sends the feature vector corresponding to the reference pixel to the process *match* running on the node which has access to the center of the search area. Since the search area may not be entirely accessible by that node, help could be required from neighboring nodes. Help is provided by the processes

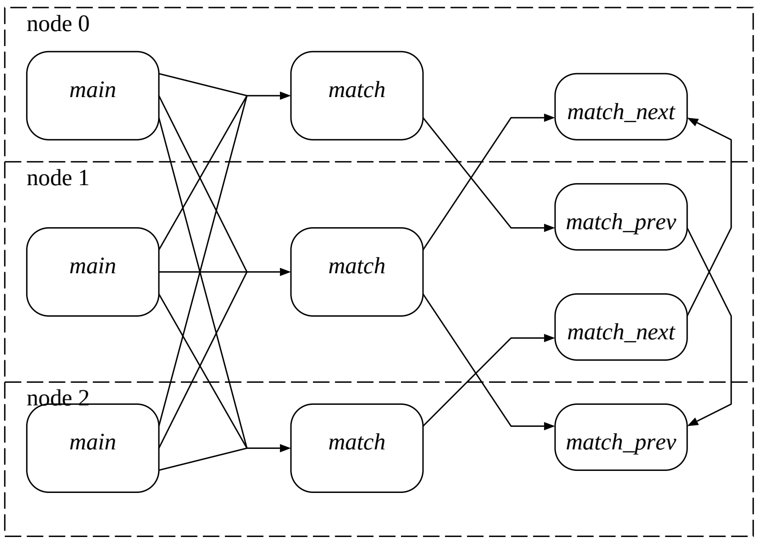

Figure 3: Processes for Feature Vector Matching.

*match_next* on the previous node or *match_prev* on the next node. In Figure 3 the four processes and the (a)synchronous communication channels between them are depicted.

Communication channels are realized in ECL by connecting two *channel ports*. A fixed number of channel ports is available on each node. It is possible to make a many-to-one connection by connecting many source ports to one destination port. In order to transport reference point $p$ and reference vector $\vec{r}(p)$ to one of the *match* processes, we use these many-to-one connections as asynchronous communication channels. In a similar way, asynchronous message passing is applied to send requests for help to one of the processes *match_next* or *match_prev*. As shown in Figure 3, both processes *match* and *match_prev* running on node $n$ can send messages to *match_prev* running on node $n + 1$. Analogously, *match_next* running on node $n - 1$ may receive messages from the processes *match* and *match_next* running on node $n$.

Because of the asynchronous communication, we needed to add buffer processes at the receiving side of the channels. Buffer processes are created for each *match*, *match_next* and *match_prev* process.

Storing the results of matching is established by means of Remote Procedure Calls. The matching processes determine for point $p$ and reference vector $\vec{r}(p)$ which search vector $\vec{s}(q)$ in search space $\sigma_p$ corresponds best to the reference vector. The disparity vector contains the differences in $x$ and $y$-coordinates of points $p$ and $q$, and has to be stored in the disparity map on location $p$ which could be in the local memory of another processor. To avoid software overhead and to obtain better readable programming code, Remote Procedure Calls are used to do the job.

## 5 Applications and Results

Hierarchical Feature Vector Matching is a robust algorithm which is being used for all sorts of applications such as the modelling of terrain [6, 8], navigation of spacecraft or autonomous robots [13], the visualization of weather satellite images [14] and as input for image warping. A first application for the parallel implementation of HFVM is surface measurement during the construction of tunnels. The processing chain for a practical stereo vision system for outdoor surface reconstruction can be split into three separate tasks:

1. *Image acquisition.* Digital images from the surface are generated using CCD cameras.
2. *Batch processing.* A 3D surface description is generated for the area which is covered by the acquired stereo images. Results are merged automatically.
3. *Interactive evaluation.* Parts of the 3D reconstructions are displayed and printed upon request of the operator.

## 5.1 System Overview

The image acquisition unit consists of two cameras tightly attached to each other. One acquires the so called 'scene images', which is the surface to be measured, the other camera is used for orientation of the system with respect to a given world coordinate system. It acquires images from a region where a set of accurately known reference points can be found and recognized automatically. Both cameras acquire their data at the same time. An automatic calibration process shortly after image acquisition is performed to verify the success of the orientation-finding procedure [15]. Already during image acquisition the hardware configuration used for batch processing is involved, interfaced by a digital frame grabber with two inputs.

Parallel matching of each stereo image pair is immediately followed by a 3D reconstruction process. Using the Locus Method the resulting *digital elevation map* (DEM) is projected on a cylinder [16]. Each stage of the tunnel construction process is described: rough surface, rough concrete and final concrete. Interactive monitoring of the construction can be performed separately from the construction site. 3D views, volume measurements and various statistical methods for quality control and inspection can be applied by an operator. Since the images can be overlaid to the DEMs, a thematical analysis may also be performed.

The processing hardware consists of a standard Pentium PC, standard hard disk and a processor board containing three Texas Instruments TMS320C40 with 17MB memory each. The 2 channel frame grabber which is connected to one of the C40's with a 20MB/sec link is mounted on the same board. The C40's are extensively used for stereo matching. A touch screen completes the list of hardware components to ensure easy remote action under wet, muddy and dark conditions.

Table 1 lists the main parameters and requirements involved during the system design.

## 5.2 Performance, Scalability and Reliability

The parallel implementation of HFVM is extensively tested. An acceptance test plan was defined and carried out. In order to measure performance, time measurements were carried out on processor networks of 1, 2 and 3 C40's, respectively. Both images were equally divided over the nodes. Two images of size $512 \times 512$ were used as input. The measured speed-up for two and three nodes was 1.88 and 2.54, respectively, which results in an efficiency of 0.94 and 0.85. The part of the computation time which is spent on matching is about 80% and increases slightly when more processors are added.

Parallel HFVM is developed such that the number of nodes in the processor network can be increased, as long as the number of image lines of the top level of the pyramid is larger than

Table 1: Parameters and requirements for surface reconstruction.

| | |
|---|---|
| Camera Resolution | 1000 $\times$ 1000 pixels |
| Delay of autocalibration | 5 seconds |
| Matching of one stereo pair | 20 minutes (every single pixel) |
| 3D Reconstruction from disparities | 20 seconds |
| Memory on C40's | 50 MB |
| Weight of entire system | Less than 20 kg |
| Hardware costs | Less than 50 000 $ |
| Number of Reference points for Autocalibration | Between 8 and 20 |
| Autocalibration Accuracy | Better than 5 mm |
| Surface Reconstruction Accuracy in 10 m distance | Better than 20 mm |

or equal to the number of nodes. In case of 1024 $\times$ 1024 input images and 5 pyramid levels, up to 32 processors can be incorporated. However, the efficiency reduces substantially because of the increasing number of interprocess communications. In case of three processors, the second node communicates with both the first and the third node. The number of interprocess communications per node does not increase if a fourth or even fifth processor is added, as long as the size of the input images is large enough. In general it can be said that a larger processor network is only efficient in case of large, say 1024 $\times$ 1024, input images.

Besides performance, also robustness and reliability were tested. Images of unusual size were used as input; many combinations of input paramaters, such as the size of search areas, the number of pyramid levels and the number of features were tried; and several types of input images (synthetically generated, satellite images, tunnel surface images, etc.) were used. Reliability was measured using synthetically generated images as input. For these images, the disparity maps are known in advance and can be compared to the results of the parallel HFVM program. It is concluded that the program is robust and reliable, and satisfies all predefined requirements.

## 6 Conclusions

An implementation of Hierarchical Feature Vector Matching was realized on a network of TMS320C40 digital signal processors. As a communication layer, the Eindhoven Communication Library was used to obtain a virtually fully connected processor network and to perform function invocations on an arbitrary node. On top of ECL a parallel program to exchange image lines between nodes was implemented. Using this program, filter operations such as convolution, local variance, median filtering and interpolation reduced to sequential programming problems. Distributed matching was realized using asynchronous communications and Remote Procedure Calls. The entire parallel HFVM program was tested thoroughly to check the requirements in terms of performance, robustness and reliability.

The parallel implementation of HFVM has been embedded in a stereo vision system for tunnel surface reconstruction. The basic features, functions and components of this system were described. It has been demonstrated that the use of a digital signal processor network satisfies the strong industrial requirements in terms of performance, robustness and price. The entire system is currently in prototype stage and will be ready for production in the summer of 1996.

We conclude that parallel programming techniques are becoming more and more important for the development of embedded systems. We would like to stress the importance of a communication library to create a virtually fully connected processor network and to perform function invocations on any node. Using the Eindhoven Communication Library we were able to implement Hierarchical Feature Vector Matching on a processor network, without having to bother about software overhead for synchronous communications or deadlock prevention. Parallel HFVM can easily be maintained and ported to other hardware configurations. The program has been written such that the processor network can be extended to achieve a further decrease of the computation time, without having to adapt the programming code. Accurate real-time stereo vision will be feasible using the techniques described in this paper provided the appropriate hardware is available.

## Acknowledgements

The development of Parallel HFVM has been carried out at Joanneum Research in Graz, Austria, in the framework of the postmaster's programme Software Technology. This postmaster's education is organized at Eindhoven University of Technology and is concluded by a nine-month project to be carried out in an industrial or research environment. The authors would like to thank the postmaster's education and TNO Physics and Electronics Laboratory, who made the writing and presentation of this paper possible.

## References

[1] G. Paar, W. Pölzleitner and A. Bauer, *A system for on-line 3D measurement of rough natural surfaces* in Proc. Photonics East, Philadelphia, October 1995

[2] C.A.R. Hoare, *Communicating Sequential Processes*, CACM, Vol.21(8), pp.666-677, August 1978

[3] P. Pierce, *The NX message passing interface* in Parallel Computing 20 (1994) 463-480

[4] L.W. Tucker and A. Mainwaring, *CMMD: Active messages on the CM-5* in Parallel Computing 20 (1994) 481-496

[5] J.J. Lukkien, *The Construction of a small Communication Library*, TR9501, Eindhoven University of Technology, 1995

[6] G. Paar and A. Almer, *Fast Hierarchical Stereo Reconstruction* in Proc. 2nd Conference on Optical 3-D Measurement Techniques, pp. 284-291, Zürich, Switzerland, October 1993

[7] Y.P. Wang and Th. Pavlidis, *Optimal Correspondence of String Subsequences* in IEEE transactions on pattern analysis and machine intelligence, vol. 12, no. 11, November, 1990

[8] G. Paar and W. Pölzleitner, *Stereovision and 3d Terrain Modeling for Planetary Exploration* in Proc. 1st ESA Workshop on Comp. Vision and Image Processing for Spaceborne Applications, ESTEC, Noordwijk, The Netherlands, June 1991

[9] P.J. Burt, *The Pyramid as a Structure for Efficient Computation* in Multiresolution Image Processing and Analysis, A. Rosenfeld, Ed., pp. 6-35, NY: Springer-Verlag, 1984

[10] E.W. Dijkstra, *A discipline of programming*, Prentice Hall, Englewood Cliffs, NJ, 1976

[11] A.Kaldewaij, *Programming: The Derivation of Algorithms*, Prentice Hall, 1990

[12] N.H.L. Kuijpers, *Parallel Implementation of Stereo Vision on a Distributed Memory System*, Eindhoven University of Technology, Joanneum Research, August 1995, ISBN 90-5282-503-3

[13] G. Paar and W. Pölzleitner, *Robust Disparity Estimation in Terrain Modeling for Spacecraft Navigation* in Proc. 11th ICPR, International Association for Pattern Recognition, 1992

[14] O. Sidla and G. Paar, *Visualization of Weather Satellite Images Using Hierarchical Feature Vector Matching* in Proc. International Workshop on Image Analysis and Synthesis, pp. 173-180, June 1993

[15] O.D. Faugeras and G. Toscani, *The Calibration Problem for Stereo* in IEEE trans. on Computer Vision, 1986

[16] A. Bauer and G. Paar, *Stereo Reconstruction From Dense Disparity Maps Using the Locus Method* in Proc. 2nd Conference on Optical 3-D Measurement Techniques, pp. 460-466, Zürich, Switzerland, October 1993