A Multiprocessor 3D Vision System for Pick and Place

Michael Rygol, Stephen Pollard and Chris Brown

Artificial Intelligence Vision Research Unit,
University of Sheffield,
Sheffield S10 2TN

We describe a 3D vision system implemented upon a locally developed transputer-based hybrid parallel processing engine named MARVIN (Multiprocessor ARchitecture for VIsioN), hosted by a SUN workstation. In addition to the recovery of scene descriptions from edge based binocular stereo, the system incorporates a model matching algorithm which is able to accurately locate modelled objects within such scenes. The competence of this vision system is demonstrated by visually guiding a robot arm to pick up various objects in a cluttered scene with a total processing time of approximately 10 seconds.

INTRODUCTION AND OVERVIEW

The computational complexity of machine vision algorithms is such that their use in industrial environments is often limited when executing on conventional sequential computer architectures. It follows that much potential for improvement exists through the application of parallel processors. However, realising that potential is not entirely straightforward as the type of parallelism is not uniform throughout the processing cycle. At the lower levels, tasks such as edge detection involve only local image operations and offer much potential for *spatial* parallelism. At the topmost level, model matching offers greatest potential for model instance and *featural* parallelism.

The goals of the work described in this paper are twofold. Firstly, to demonstrate that it is possible to develop a large parallel 3D vision system (as opposed to the parallelisation of a single image processing routine often presented elsewhere). Secondly, to demonstrate the suitability of our prototype general purpose vision engine, MARVIN, for exploiting numerous forms of parallelism within a single overall application.

MARVIN has been designed to provide a balance between frame-rate hardware and a general purpose parallel computing resource. The system has been designed to be both general purpose and easy to use.

SYSTEM ARCHITECTURE

MARVIN's 25 processors (T800 transputers) are firm-wired as a regular, fully-connected mesh with 3 rows, 8 columns and an extension for the root transputer. The machine architecture is further described in [1]. A scaled down version is shown in figure 1. Two links from the root transputer are connected to the host machine. Link 0 provides the usual boot path and I/O interface whereas link 1 provides a dedicated communications path to *tditool* (a multi-window tool, running in Sunview) allowing multi-processor console I/O [2].

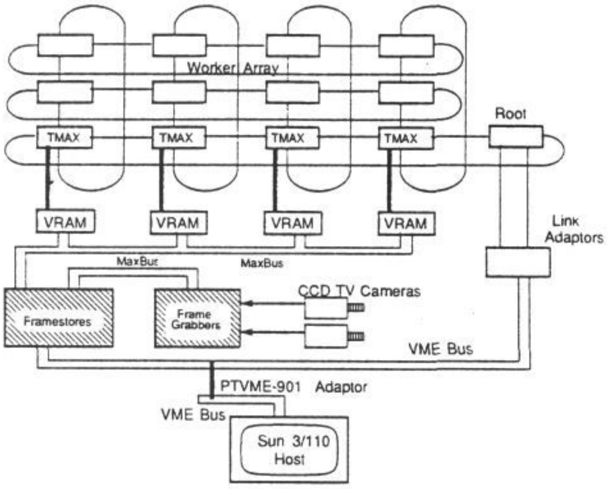

Figure 1. Marvin System Architecture

One of the rows consists of locally developed transputer cards (named TMAX) provided with 4 frame-rate byte-wide bidirectional video busses (the industry standard MAXbus, by Datacube Inc) and circuitry to control the operation of these busses, such as the ability to obtain a region of interest from the video stream. These busses may also be ganged to allow the transfer of wider data streams up to 32 bits. All other processors are standard T800 + 2Mbyte TRAMs (Transputer Modules).

This research was supported by SERC project grant no. GR/E 64497 awarded to J.P. Frisby and J.E.W. Mayhew under the ACME programme in collaboration with GEC Hirst Research Centre, Wembley

BMVC 1990 doi:10.5244/C.4.31

The TMAX cards are instrumental in providing a fast data path through the system for (predominantly) image data, minimising data transit times, a common problem with message passing machines. It is our intention to exploit further the MAXbus facility by adding compatible cards to perform frame rate image processing operations (eg. convolution).

The stereo images (512 pixels square) are simultaneously grabbed from a pair of ccd cameras via two Datacube Digimax framegrabbers into Datacube framestores. The framestores share a number of MAXbus ports with the row of TMAX cards through which synchronised image acquisition can be achieved by the TMAX cards (utilising a shared interrupt signal).

Resident on each TMAX is a server providing network-wide facilities. Any task on any processor may request operations from these servers. Such operations include region-of-interest acquisition, data plotting and low level control operations. Access to these operations is via a library of function calls.

The entire system is programmed in Parallel C and runs within a locally developed run-time environment [2]. The model of parallelism adopted is based upon Communicating Sequential Processes (CSP) where the system comprises a number of sequential processes executing concurrently and communicating via *channels* [3]. Our run-time environment allows all processes throughout the system to communicate with each other via *virtual* channels, allowing (addressed) messages to be exchanged between processes that have no knowledge of the hardware topology.

The root processor performs no vision processing but holds the highest levels of the *control architecture* and runs various servers to provide host facilities to the rest of the network.

Worker processors provide various repertoires of *resources* which are requested to do work by other processes in the control architecture, employing a *client-server* model. The development of this technique has greatly simplified the addition of new competences to the system.

Vision processing is broken down into a number of *tasks* each of which may itself be multi-threaded. (A "thread" is a lightweight process that may share code and data with other threads.) Numerous copies of the vision task bundles are distributed across the network. Each of these tasks receives a control message that contains all of the necessary run-time parameters for that task. This technique allows dynamic changes of operation in the system.

CONTROL ARCHITECTURE

System control is hierarchical and distributed. At any level in the hierarchy, the system comprises a small and manageable number of processes each with a well-defined interface, allowing simplified interaction within the system.

A simplified version of the control architecture is shown in figure 2 where higher levels of the hierarchy are towards the centre. The top levels of the control hierarchy are resident upon the root transputer. A control thread is created on the root processor for each sub-group of processors performing the vision processing allowing asynchronous control, if necessary. A central thread, the highest level in the hierarchy, controls the operation of the vision processes via these threads. Each vision process has no built-in knowledge of where its data comes from or where results are to be sent. The action of the machine is completely fluid, all dataflow being determined by the control architecture at run-time.

Each vision process communicates with its (superior) control thread upon completion with a small reply packet. The controller then uses this information to instruct the task performing the next processing stage. In this way the various tasks are kept independent from each other as much as possible.

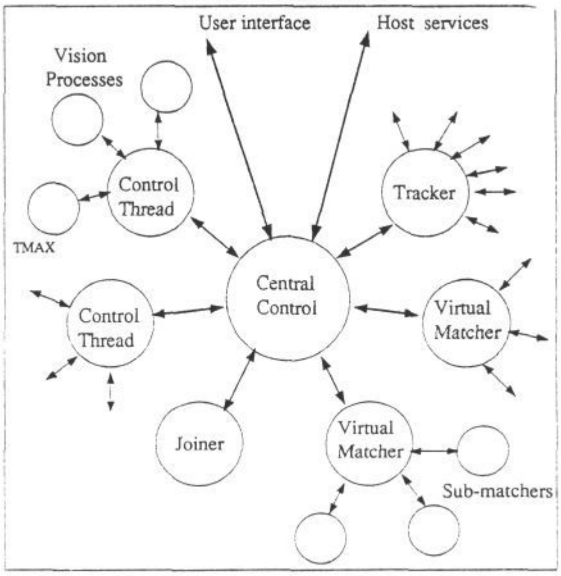

Figure 2 Control Architecture

Some of the tasks require asynchronous control operations. In this case, control information is received by a separate thread running within the task, allowing control information to be asynchronously decoupled from data flow.

The MARVIN Software Infrastructure [2] allows us to ignore the physical communication paths between tasks and the physical interconnections between processors, allowing any *logical* topology to be chosen and changed dynamically.

Figure 2 highlights the principle of the distributed control paradigm. It is easy to see that control of a parallel system is made simple and "open" using this technique as opposed to, say, a completely centralised or data driven organisation. The user may then interface with the system via the top level of control.

RECOVERY OF 3D SCENE GEOMETRY

This vision system is derived from the AIVRU TINA system [4] which employs edge based stereo triangulation as a basis for three dimensional description.

We employ spatial parallelism by dividing both left and right images into 8 horizontal slices approximately 64 rasters wide. A small overlap (2 rasters) between adjacent slices is incorporated to avoid boundary effects and simplify the processes of recombination that occur later. There is a limit to how thin the image slices can be made without adversely effecting the reliability of the stereo matching process. However, potential does exist for further subdivision of images in the horizontal direction with a small increase in the complexity of the stereo matching algorithm.

Each image slice pair is acquired simultaneously into an allotted TMAX, controlled remotely by the control task on the root processor. The pair of transputers vertically adjacent to the TMAX are used to process the left and right image slices in parallel. The size of the right image slice is adjusted to take into account the warping effect of the rectification of edge locations into a parallel camera geometry. This is determined from a copy of the calibration data resident upon each processor (this may be updated dynamically to allow for updated calibration estimates to be incorporated).

Obtaining Edges

Edges are obtained to sub-pixel acuity from grey-level images by a single scale high frequency application of the Canny edge operator [5]. The high frequency operator used here employs a gaussian mask of sigma 1.0. Convolution is computationally expensive but fortunately the two dimensional gaussian smoothing can be achieved through two 1 dimensional convolutions (i.e. first along the rows and then the columns). However, it is our preferred intention, in the longer term, to use specialised convolution boards directly on the MAXBus video stream.

Each Canny process obtains the raw image slice from a TMAX with a simple parameterised function call. The Canny task processing the left image slice packs the resultant edgemap into a data packet and sends it to a collection thread running within the right Canny task. Upon completion, the right Canny task and the collection thread rendezvous and send a reply to the control thread, on the root processor. Following detection, edge strings are formed by linking edge pixels (edgels) into chains of connected components.

Stereo Matching

We use a locally developed algorithm, PMF [4], for stereo matching. In PMF, matches between edges from the left and right images are preferred if they mutually support each other through a disparity gradient constraint and if they satisfy a number of higher level grouping constraints, most notably uniqueness, ordering (along epipolars) and figural continuity.

The PMF task runs on the processor that now holds both edgemaps (the one that held the right image slice). The edge structures are organised so as to make both spatial location and connectivity explicit. To avoid major data transfer and recomputation the PMF control packet (sent from the root) simply contains a pointer to the edge structures in the memory shared by the threads.

Geometrical Elements

As well as being matched, edge strings are processed to recover descriptions of the two dimensional geometrical elements they may represent. This process is currently limited to straight line descriptions though in previous implementations we have also recovered circular descriptions, and are currently developing methods to identify ellipses. The algorithm uses a recursive fit and segment strategy. Segmentation points are included when the underlying edge string deviates from the current line fit. Robustness of the system (and its speed) relies upon the fact that a heuristic search strategy is used to identify those regions of strings/segmented sub-strings that are most amenable to straight line fit. The actual fit is computed by orthogonal regression.

Given descriptions of the two dimensional geometry (in a single view) and the results of the application of the stereo algorithm to the underlying edge strings, it is possible to recover three dimensional geometrical descriptions. Disparity values can be obtained along the 2D geometrical descriptors for each matched edge point. A second stage of 2D fitting (in arc length against disparity) computes the fit in disparity space. Finally, disparity data is projected into the world using transformations based upon the camera calibration. The sequence of operations taking place on each pair of processors is shown in figure 3.

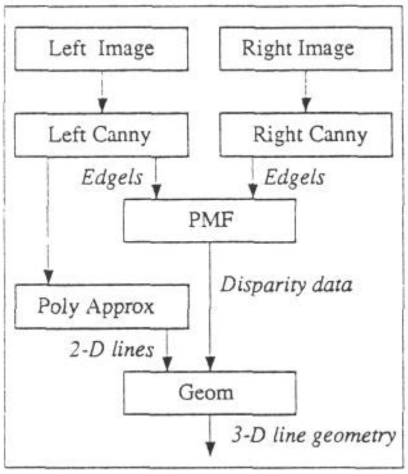

Figure 3. Data flow for recovery of scene geometry for each image pair slice

At this point in the processing, the 3D geometry from the current scene is spatially distributed across a number of processors. This data needs to be integrated into one data set as if it came from a single processor. A Joiner task (figure 5) communicates with all of the 3D geometry tasks, receiving both 2D and 3D information. These descriptions are optimally combined where valid 2D connectivity is identified between image slices. Upon completion, the Joiner returns the integrated geometry to the main control task resident on the root processor. The controller then forwards this information to a number (defined at run-time) of model matcher tasks distributed throughout the network.

Figure 4 shows an example of the 3D geometry recovered from a scene, projected over a ground plane.

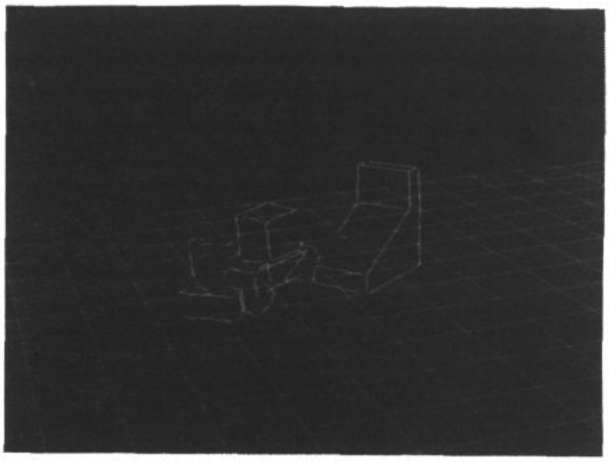

Figure 4. 3D Geometry with ground plane

One important consideration in this parallel vision system is that the computation of descriptions higher in the processing chain is dependent upon large amounts of previously computed data. For example the 3D data structure is dependent upon both 2D polygonal approximations and the matched edges. Accordingly, the use of the traditional processor farm is inappropriate as the amount of data flow required would make it unusable.

MODEL MATCHING

The model matcher is able to give accurately the position and rotation of modelled objects (defined in terms of their 3D geometrical primitives) from the geometry recovered by the earlier processing stages.

The adopted strategy [6] is to base initial matching hypotheses on congruencies identified between 3D scene descriptions and a chosen subset of features from the model. Following hypotheses, potential matches are ranked on the basis of the extent to which further support exists for the three dimensional transformation they implicitly represent (between model and scene). The algorithm exploits ideas from several sources: the use of a partial pairwise geometrical relationships table to represent object model and scene description from Grimson and Lozano-Perez [7], the least squares computation of transformations by exploiting the quaternion representation for rotations from Faugeraus et al [8], and the use of focus features from Bolles et al [9].

Whilst exhaustive search for maximal cliques of consistent scene and model descriptions is avoided, the algorithm still requires a considerable amount of searching to be performed. Furthermore, the computational expense of the algorithm (which increases roughly linearly with the scene complexity) is replicated for each modelled object when implemented on a sequential machine.

The most obvious way to exploit the architecture of MARVIN in the model matching phase is to run multiple instances of the model matcher on different processors, thereby searching the same data for different models. This limits the parallelism to the number of models being searched for and is still too computationally time-consuming. A further degree of parallelism is obtained by employing a multi-level control architecture within the matching process itself. The model matching process is decomposed into separate searches for feature sets from the modelled object in the spirit of *characteristic views* [10]. A characteristic view in this scenario is deemed to consist of a set of geometrical features of the object that are consistent with a range of closely related viewpoints of the object. This, in effect, allows the distribution of the search tree over a number of processors.

To realise this in a consistent manner, a *virtual matcher* is used. The virtual matcher receives a control message describing the name of the model to become its responsibility and a list of processors available for use (allocated at run-time, typically 6) which have a resident matching task which will be instructed to search the scene geometry for a particular characteristic view of the object. The virtual matcher obtains all relevant information about the model (from files on the host machine) and the current geometry (from the Joiner) and distributes this information to its subordinates. The virtual matcher retains a precomputed grasp position, specifying a pick-up position in the coordinate frame of the model description. Each subordinate matcher attempts to locate a *subset* of the model features.

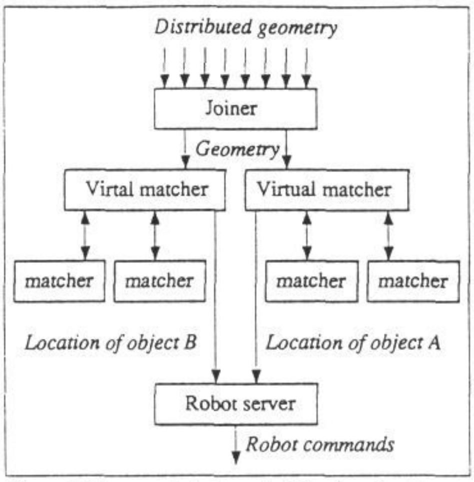

Figure 5 Geometry joining and object location for two objects (A & B) each with two subordinate matchers

Upon completion, the virtual matcher chooses the best match obtained from all of its subordinates and feeds the object location and grasp position (now transformed into the world coordinate frame) up to the next higher level of the control hierarchy. A virtual matcher is used for each model to be located in the scene with the resources of the entire network being shared between them.

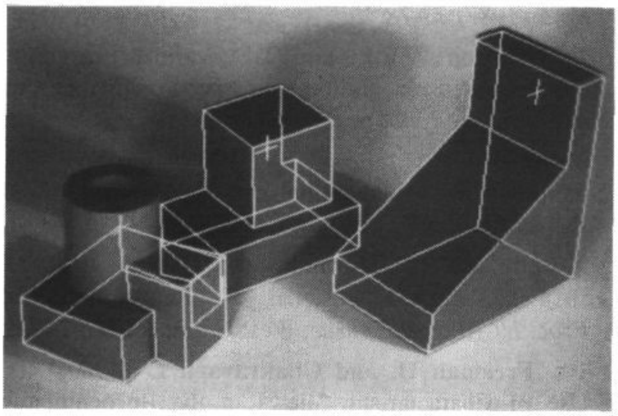

Figure 6. Matched models

This technique allows a search to be made for multiple objects (with no extra time penalty) in a consistent, flexible and highly efficient manner (figure 5). Figure 6 shows the reprojection of three matched models onto the original image along with their grasp positions.

CONTROLLING THE ROBOT ARM

A UMI robot controlled by an IBM PC is used to perform the pick and place. The IBM PC is in turn connected to a SUN 3 workstation over a serial line. MARVIN may talk to any SUN in our network by means of the UNIX socket mechanism. Facilities provided by our run-time environment allow any task on any transputer to open and communicate with sockets to the outside world [2].

Each virtual model matcher returns a rotation and position of the relevant object. This information is used to transform a precomputed grasp position from the model coordinate frame to the world coordinate frame.

This information is sent over a socket to a server controlling the robot (figure 5) on a remote machine according to an agreed protocol, specifying the name of the model, where to pick it up and what to do with it. MARVIN need have no concern with solving the inverse kinematics, camera to robot transformations and path planning in order to pick the objects from the workspace, these issues are computed in different areas of the *computer* network. An example of the robot picking up a widget is shown in figure 7.

The *hand-eye* calibration (the transformation between camera space and robot space) is obtained by using TINA to locate a flag held in the gripper of the robot arm. This is repeated numerous times in various positions of the robot workspace. A least-squares method is used to obtain the best transformation.

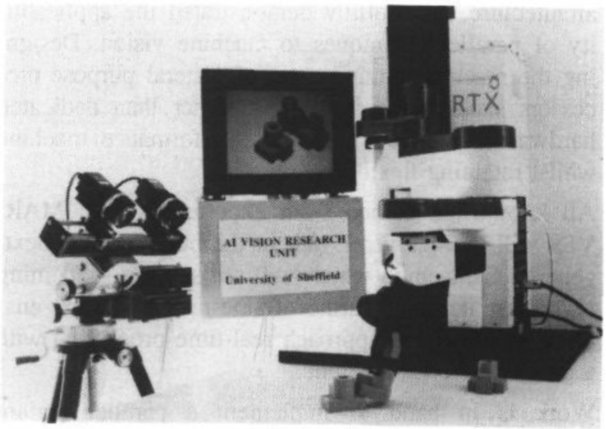

Figure 7. Robot picking up a widget

CONCLUSIONS

A multiprocessor transputer-based vision system has been described. We have employed two different techniques to utilise the parallel hardware. The recovery of the scene geometry employs *spatial* parallelism by decomposing the image pairs into horizontal image slices. This is a natural approach as depth data is obtained by matching elements between left and right images. The requirement to transfer image data quickly into the heart of the network is overcome by means of the MAXbus and TMAX cards, making the architecture of MARVIN well suited to exploiting spatial parallelism.

The model matching process is parallelised on a *featural* basis, using a hierarchical control strategy. Our run-time environment provides a suitable framework for integrating processing modules by allowing processors to provide various resources to be utilised by any other requesting processes.

Using MARVIN we have a system that in a former incarnation took well over one hour to to locate *one* object, now runs in around 10 seconds, locating *numerous* objects, making it far more practicable to perform interactive "what-if" experiments and thus bringing this level of visual competence ever closer to the industrial domain.

A more detailed breakdown of processing timings for a typical scene is shown in table 1. The two vision processes which have benefited most from being parallelised are the Canny and model matching stages, the two most costly tasks on a sequential machine.

Over the entire sequence of operations from grabbing the image to obtaining four matched models we achieve an average performance per processor of over 80%. This figure relates the estimated time taken by one processor to the time taken by 24 processors.

Building a parallel vision system on a multi-processor architecture has usefully demonstrated the applicability of parallel techniques to machine vision. Designing the machine around a set of general purpose processors such as the transputer, rather than dedicated hardware has given us a high performance machine whilst retaining flexibility.

All knowledge gained from experiences with MARVIN will be highly relevant in the building of a next-generation machine which will utilise the forthcoming H1 transputer and further frame-rate hardware enabling us to start to approach real-time processing with a general purpose machine.

Work is in hand to implement a parallel feature tracker on MARVIN which will follow (at near real-time) features identified as being those from a modelled object. This beacon tracking is to provide some of the information required to enable our in-house vehicle to navigate through an unknown environment.

Vision Process	Time (ms)
Canny	5500
PMF	1000
2D Geometry	100
3D Geometry	200
Geometry joining	100
Model Matching	1600
System Overheads	1000
Total	~10000

Table 1. Processing times

REFERENCES

1. **Brown C. and Rygol M.** (1989), "Marvin : Multiprocessor Architecture for Vision", *Proceedings of the 10th Occam User Group Technical Meeting*

2. **Brown C. and Rygol M.** (1990), "An Environment for the Development of Large Applications in Parallel C", *Transputer Applications '90*

3. **Hoare C.A.R.** (1985), Communicating Sequential Processes, *Prentice-Hall International.*

4. **Porrill J, Pollard S B, Pridmore T P, Bowen J, Mayhew J E W, and Frisby J P** (1987) "TINA: The Sheffield AIVRU vision system", *IJCAI 9, Milan 1138-1144.*

5. **Canny J** (1986), "A computational approach to edge detection", *Trans. Patt. Anal. & Mach. Intell, 679-698, PAMI-8.*

6. **Pollard S B, Porrill J, Mayhew J E W and Frisby J P** (1986) "Matching geometrical descriptions in three space", *Image and Vision Computing, Vol 2, No 5 (1987) 73-78.*

7. **Grimson W.E.L. and T. Lozano-Perez** (1984), "Model based recognition from sparse range or tactile data", *Int. J. Robotics Res. 3(3): 3-35.*

8. **Faugeras O.D. and M. Hebert** (1985), "The representation, recognition and positioning of 3D shapes from range data", *Int. J. Robotics Res*

9. **Bolles R.C., P. Horaud and M.J. Hannah** (1983), "3DPO: A three dimensional part orientation system", *Proc. IJCAI 8, Karlshrue, West Germany, 116-120.*

10. **Freeman H. and Chakravarti I.** (1980), "The Use of Characteristic Views in the Recognition of Three-Dimensional Objects", *Pattern Recognition in Practice, pp 277-288, Gelsema and Kanal (Eds), North-Holland.*