ANIT
A System for Perceptual Subsumption and Intelligent Vision Systems

Stuart M Cornell, John E W Mayhew, Sam Harrison
Artificial Intelligence Vision Research Unit,
University of Sheffield, Sheffield, S10 2TN,England

Abstract

This paper documents the details of a development environment called ANIT which provides tools to enable researchers with no knowledge of parallel systems to contribute to and operate whole systems. These tools extend from a simple compilation environment to X based control and display tools for all the modules of the systems. ANIT meets the need to bind *Tina*[10] (AIVRU's software vision system) and future projects together in a way which facilitates the inheritance of existing working code in an evolving system. The target architecture for ANIT is a mixed transputer - Unix Host network incorporating specialised frame-rate vision hardware. To achieve this requirement ANIT has been given several goals, including: (i) Definition and support for a structure for each processing module so as to simplify system modeling; (ii) Definition and provision of a simple but powerful method by which modules communicate, thereby minimizing the knowledge and code needed to produce a communicating module and hence protecting the user from the typical time-consuming overhead in development time associated with solving inter-module communication problems; and (iii) Isolation of the user/programmer from the hardware to prevent dependencies on ever changing architectures, and specifically removing from the C programmer any need to know how to program the transputers even though they comprise the hardware on which his system is executed. the latter facility is supported by IIPC (AIVRU's in-house run-time environment for Transputer-Sun networks) [3] which is wholly integrated with ANIT. We report demonstrations of the value of ANIT/IIPC in several projects involving complex control systems for autonomous vehicles with visual guidance from a 4 degrees of freedom stereo 'eye/head' rig. The value of ANIT for other applications, such as industrial inspection and assembly and intelligent surveilance are discussed.

1 Introduction

The development of complex multi-competence systems, for vision and control applications presents many common problems. The obvious ones are the com-

BMVC 1994 doi:10.5244/C.8.64

petences and algorithms themselves but regularly overlooked are the problems of actually building an integrated system. Combining the ideas and algorithms of different projects and people into one workable and maintainable system is a formidable task. This paper documents the details of a development environment called ANIT aimed at facilitating system development. The name arises from the *Architecture for Navigation and Intelligent Tracking* which was originally developed in Lisp by Mayhew and then repeated by Booth[1]. We describe a new implementation in ANSI C which provides a whole development environment for a dual architecture comprised of Unix Hosts and Transputer networks. In contrast to other data-flow architectures, the configuratuin is entirely in software and imposes no physical constraints on the developers. ANIT provides tools to enable researchers with no knowledge of parallel systems to contribute to and operate whole systems. These tools extend from a simple compilation environment to X based control and display tools for all the modules of the systems. We shall explain the implementation and operation details of the ANIT system, and describe some of the projects already in operation or development.

2 Aims of ANIT

The specific set of requirements for the ANIT software development environment and architecture are as follows:-.

- Define and support a structure for each processing module to facilitate system modeling and design.
- Define and provide a simple but powerful method by which modules communicate minimizing the knowledge and code needed produce a communicating module.
- Isolate the user/programmer from the hardware where possible so as to prevent dependencies on ever changing architectures.
- Provide a transparent Unix / Transputer development environment
- Provide a simple compilation and execution environment but one which does not prevent knowledgeable users from adding specialised modifications.
- Support the debugging of modules by providing different levels of runtime information about processing / communication.
- Provide access to familiar window / button interfaces for all tasks, (this is normally unavailable for transputer tasks).
- Provide tools which allow multiple users to combine systems.
- Provide powerful runtime libraries for subsumption.
- Provide standard sets of utility libraries for the control of specialised hardware such as Marvin's frame rate hardware[11][12][2] and TMAX cards.

3 Current Systems

3.1 Micro-Saccadic Tracker using Camera Rig

A micro-saccadic tracker[5] on AIVRU's stereo camera rig was one of the first systems implemented in ANIT. It was originally implemented in a previous version of ANIT and in doing so many of the ANIT philosophies were tested and extended. The current ANIT implementation has been built on this experience. Porting the tracker to the new ANIT was effected in a short period of time, demonstrating the benefits of the high level of abstraction achieved by the modular approach enforced by ANIT. The new tracker uses the transputer network primarily but the control loop can be run equally well on a dual host system, as can most current ANIT systems.

3.2 Smooth pursuit Tracker using Camera Rig

The above tracker has been implemented to use a smooth-pursuit velocity control paradigm. It uses the main control loop of the micro-saccadic tracker but replaces the head control part with a state feed-back proportional velocity controller. The work has implemented a high frequency (100Hz) controller in ANIT validating ANIT's usability in real-time control.

3.3 Trajectory control for AGV

Following on from the tracker project, a sophisticated trajectory control and docking system was implemented in ANIT which used the tracker system as a basic competence to produce vision data from which to navigate. This building upon other systems is one of the fundamental aims of ANIT and the Trajectory project has been very successful in this endeavor.

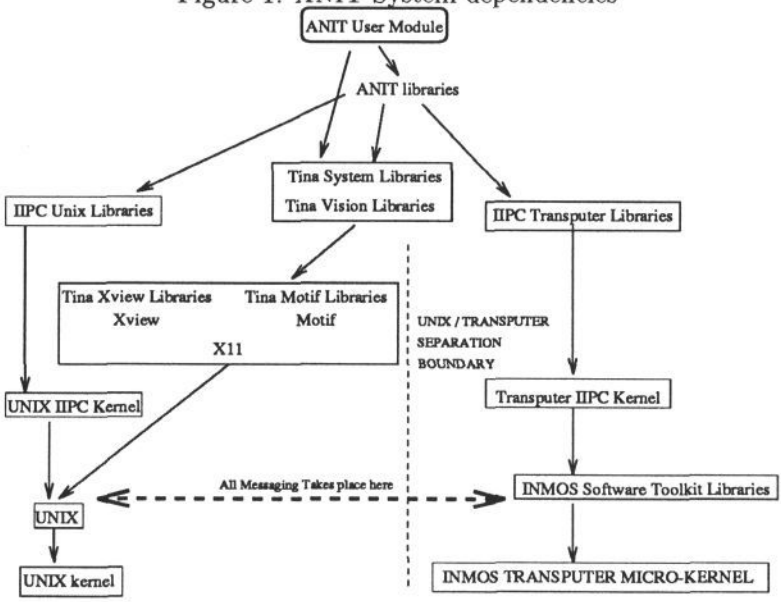
3.4 Sprite - A Self Navigating object

As a preliminary to the acquisition of another mobile robot, a simulation was implemented in Tina. The control architecture was developed under ANIT using the simulation. The exchange of the simulation for modules which control the hardware and read from the laser range finder fitted on the robot has been performed, using the same control loop developed under ANIT. Our experience has been that ANIT has greatly facilitated the development work.

3.5 Stereo Matching under ANIT

An implementation of AIVRU's stereo matching algorithm has been undertaken in ANIT and attempts to utilise as much of the existing Tina code as possible rather than re-writing it for a parallel environment[13]. This implementation will then form the basis of a high level control paradigm for AIVRU's mobile vehicle.

Figure 1: ANIT System dependencies



4 IIPC - Intelligent Inter-Process Communication

IIPC[3] is AIVRU's run-time environment, which supports all of ANIT's messaging facilities. It operates on Transputer Networks and UNIX hosts and is based on the C language. It presents a homogeneous view of the hardware system and provides mechanisms for task naming, synchronous and asynchronous communications, and dynamic task instantiation. IIPC performs all of the low level communication done by ANIT and without it, many of the facilities provided by ANIT would not have been possible. The design of IIPC was driven largely by the principles of ANIT as developed by Mayhew in previous projects[1].

5 The Hardware Environment

Within the last few years AIVRU has constructed and commissioned its own specialised parallel vision engine called *Marvin*[11]. This consists of a large (26) T800 transputer network and some in-house frame-rate video processing boards. These boards include Canny edge detection, convolution , correlation , image rectification and general purpose image processing transputer boards (TMAX cards). They all communicate by means of MAXBUS video interface, and the TMAX cards map the received images straight into transputer memory giving an extremely powerful vision resource. This engine is connected to a large network of Sun workstations and two other transputer networks . One of the targets for the vision system under development is an agile AGV fitted with a 4 degrees of freedom stereo camera rig designed and built also in AIVRU. Other application areas are in industrial

assembly and in intelligent surveilance.

6 The Software Infrastructure

Mapping a software system onto the above hardware environment requires software management. Previously AIVRU has developed successful vision competences to model-match and pick up objects and navigate an autonomous vehicle though a cluttered environment[13] , supported by a large number of image processing algorithms [9][6][8]. These successful systems, however, soon became inoperable because they were coded in isolation and the knowledge required for their use was gradually lost with staff turn-overs. This is a typical experience in laboratories developing computer vision systems world-wide and it represents a huge failure to capitalise on investments. The clear need is for systems designed from the outset to facilitate system evolution by releasing the creators of modules from the inter-module communication problems that so often make it tedious or even impossible to use one person's module in another's system. ANIT has been built with this objective uppermost in it's aims (See section 2). The latest ANIT implementation incorporates those features of AIVRU's *Tina* computer vision system which are of general use so that they are available to all ANIT programmers who might wish to exploit them (See section 13). Projects are also underway to port some of the existing *Tina* applications to a parallel environment (See section 3.5). A full breakdown of system dependencies in the ANIT / IIPC environment is shown also in Figure 1.

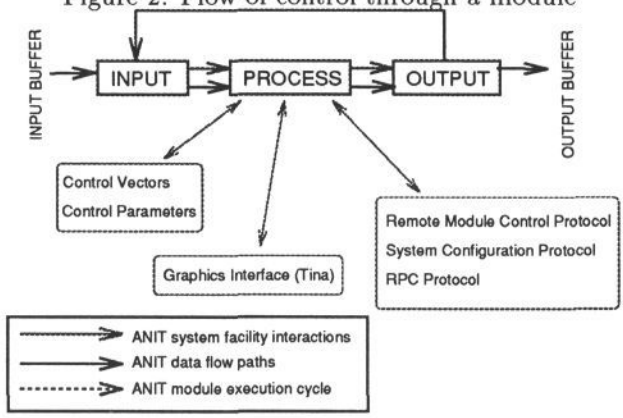
7 Module Parameterization

As part of the ANIT module definition, each module can be parameterised by means of *control vectors* and *control parameters*. In ANIT, a *control vector* is a multi-state named register visible internally and externally to the module. The control vectors are defined by the module and are used to signify operating states, processing modes and decision results. By the use of these a module can both be controlled and indicate current status to other modules. *Control parameters* are similar in the manner of visibility but are used to carry variable parameters such as gains and processing limits, and so take the form of real numbers. ie 3.14. These methods of communication are provided not as an alternative to the data passing wires, but as a complementary, intermittent method. One important use is for the system operator to modify the behavior of a system from the graphical control interface or startup script.

8 Module processing model

At the heart of ANIT is a simple model for each processing module (See Figure 2). This is formed around a simple Input - Processing - Output loop. The ANIT programmer is required to write only those stages which they wish to specialise. This is in the simple cases just the *processing stage*. The default *input* and *output*

Figure 2: Flow of control through a module



stages just collect all inputs and despatch all outputs. The use of this model makes the analysis and debugging of systems much simpler and the simple structure helps system designers decide on the granularity of the modules. Even though it may be necessary to produce a module for a trivial task, the difference in ease of modelling of the system is significant. ANIT provides utility function for the input and output stages which give a powerful interface to IIPC facilities. These utilities implement:

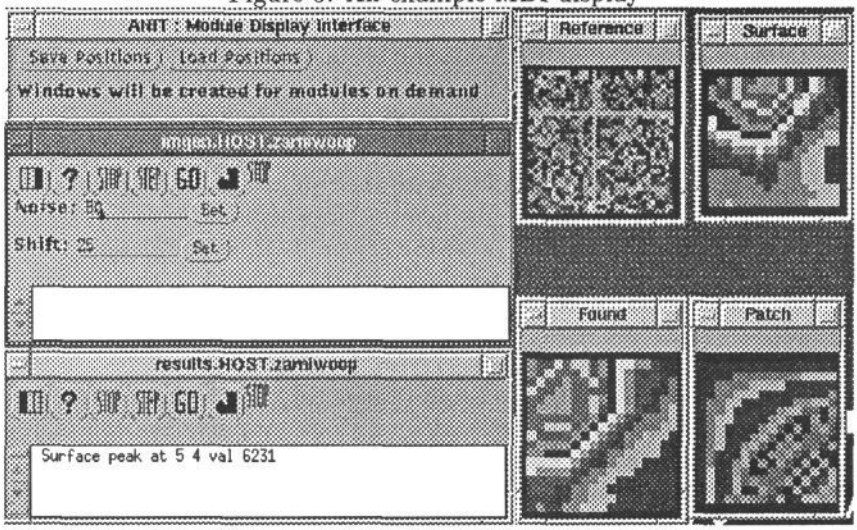
- volatile message queues,
- acknowledged / synchronised output
- timing constraint control.

In addition to the three standard stages of the module, ANIT offers two others. The *once-only stage* is run at placement of the module and contains all the declarations of input and output data structures, and wire declarations. It also contains definitions of control tool facilities and control vectors and parameters by which the module may be monitored and parameterised. The remaining module stage is the *startup stage*. This stage is executed after *once-only* and then again whenever the module is reset. The module can be reset at any stage in the process sequence but the *startup stage* will not be executed out of sequence, ie only following *output* regardless of whenever the reset vector was activated.

9 Data passing philosophy - Wires

Writing message passing programs for architectures such as the INMOS transputer is a task usually for those who have spent years learning the craft. Even with IIPC you must adhere to a strict set of protocols in order to make debugging programs bearable, for when one process receives an incorrect piece of data you must be able to trace back the problem. It could be in the message, the way it was passed, the process who created it and so on. ANIT attempts to minimise this by allowing a declarative style of message passing. The user defines (in a standard part of

Figure 3: An example MDI display



the code) what the data is, where it resides in memory, and (optionally) where it is destined to be sent. Then that data is sent at a fixed point in the processing sequence (the output stage) down a *wire*. All the user need do is alter the data as required in each processing cycle and the data will be sent again next time. In the simple case, there is one declaration of the output data in the once-only stage and then the user never need to modify the *wire* definition again. Similarly, a declaration for input data connections is all that is required, and the data store is filled with any new data at the relevant point in the cycle (the input stage). NB: There are obviously more complex variations on this theme but the principle is the same.

10 Dynamic wiring of modules

The connection of modules together by wires can be defined either by the modules concerned or by any other module, or in the *Anit Description Language (ADL)* at startup time. The connection can be modified at run-time to give dynamic wiring and re-wiring of systems as required in order to connect or disconnect sections. This easy method of defining connections gives the power to optimise the load being placed on particular processors when used in combination with careful task placement. Also, because of the point-to-point communication method it is possible to plan the system communcation lines so as to reduce traffic in critical areas of a processor network.

11 Module Display Interface

One of the major advantages ANIT gives is the ability to use X based control tools for Transputer and UNIX tasks in an identical manner. This is provided

by means of the Module Display Interface, which is in reality an special ANIT module. The ANIT programming libraries contain a set of functions which enable modules to use buttons and on the tools and input text from them (See Figure 3). They also allow the display and manipulation of images through Tina [10]. These facilities will soon include the full graphical facilities of Tina which will enable AIVRU staff to transfer existing UNIX based vision algorithms under Tina to ANIT and the Transputer environment with the minimum effort. Figure 3 shows a toy demo correlation system. The windows shown include tools for the control of two modules imgen and results, (note the control vectors and parameter.) Also shown are four image display tools produced by the modules in the system.

12 Compilation / Runtime facilities

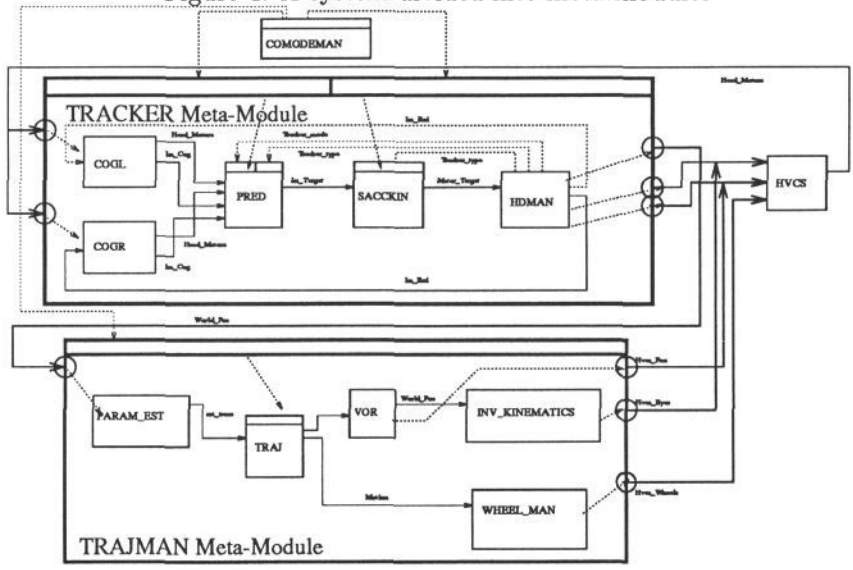
ANIT provides a comprehensive make facility which simplifies the compilation of modules for different architectures and also will create the hardware and tasks description files required by IIPC. Also provided is an easy to use *ANIT Description Language* by which the mapping of modules and their wiring connections can be converted into those descriptions required by IIPC.

13 Advanced data passing - Tina Structs

In conjunction with the Tina facilities described in the Module Display Interface section, ANIT supports the passing of complex data structure trees along its data wires. This is provided through the use of Tina structures, and provided only Tina structures are used, theoretically unlimited complex structures including recursive lists and reconverging trees can be sent. ANIT automatically handles duplicate data pointers reconstructing them at the destination of the wire. All that is required from the programmer is the pointer to the top of the data tree for ANIT to send the entire tree. This facility is currently under development and currently supports Sun and Transputer modules but it is hoped that easy porting to other processor architectures will be included. The data passing procedure is very efficient, it entails no duplication of data, and it may contain some data compression facilities in future. Each Tina data structure contains a common first field which is the *Tina structure id.* This field is initialised to contain a unique identifier for each data type enabling the identification of data structures even from simple `void *` pointers. By using this and a unique type detail string ie "wwflip" which briefly describes the contents of the structure it is possible to produce a description (or serialization) of even the most complex data tree. Once the tree has been detailed (describing all the items locations and sizes in memory) this description can be used to write the tree to a file, print out a listing of its contents, or (and more importantly for us) send it's contents down a data link and reconstruct it at the destination. And the latter is possible under ANIT.

ANIT knows how to handle a Tina serialised tree and can use them as structures for it's communication wires. All that is required from the task is that it defines the output wire as carrying a Tina data type and give the pointer to the top of the tree. This facility is very similar in essence to the XOR system used in MNT[4] which

Figure 4: A system divided into meta-modules



is based upon XDR[14], Sun's external data representation standard. The use of XDR was considered for ANIT but it could not cope with our target data (Tina structures) without the id labeling system described above. Having implemented the Tina id system it was felt more appropriate to use a bespoke system instead of XDR which has many overheads by comparison. However, the use of XDR as a future alternative protocol has not been ruled out due to its great portability and standardization.

14 Future work

14.1 Meta-Modules

In order to better facilitate the decomposition and grouping of modules in a given system. We propose the idea of a *Meta-Module*. A Meta-Module is a collection of modules which together can be treated as one processing entity. Selected facets of this group, such as wires and control vectors form the external connections for the meta module. Once the meta module is defined in the system it is treated as a normal module in every way, but as a Meta-Module it has extra facilities for the control of those modules which it encapsulates. This principle maps well onto the integration of several projects. Although under ANIT, this integration is eased by the processing model, it is advantageous to encapsulate a collection of modules together and thereby further reduce the knowledge required to use them. An example is shown of a proposed grouping of the modules forming one of the current ANIT projects in Figure 4.

References

[1] Booth,C.J.M., Mayhew,J.E.W. A Distributed Architecture for the control of complex tasks. *RIPRREP/1000/51/89 Research Initiative in Pattern Recognition, RSRE, Malvern Worcs (1989)*

[2] Brown,C., Rygol,M., (1990) An environment for the development of large applications in parallel C. *Applications for Transputers 2. IOS Press. Proceedings Transputer Applications*

[3] Brown,C., Harrison,S. and Rygol,M. IIPC-An Intelligent Inter-process Communication Environment for Transputer and Unix Hosts. *Transputer Communications Vol(1), 1993*

[4] Olsson Lars., Winroth, Harald. Object-Oriented Computational Networks for Computer Vision Applications. *Proceedings TOOLS 10 (Technology of Object-Oriented Languages and Systems) pp223-233, Prentice Hall, 1993.*

[5] Mayhew,J.E.W, Zheng,Y., Cornell,S., (1992) The adaptive control of a four-degrees-of-freedom stereo camera head. *The Royal Society*

[6] Pollard,S.B., Mayhew,J.E.W., Frisby,J.P. (1990) Experiments in vehicle control using predictive feed-forward stereo. *Image and Vision Computing 8(1), 63-70*

[7] Pollard,S.B., Porrill,J., (1987) Matching geometrical descriptions in three-space. *Image and Vision Computing 2(5), 73-78*

[8] Pollard,S.B., Mayhew,J.E.W., Frisby,J.P. (1985) PMF: A Stereo Correspondence algorithm using a disparity gradient limit. *Perception 14, 449-470*

[9] Porrill,J., Pollard,S.B., Pridmore,T.P., Bowen,J.B., and Mayhew,J.E.W. (1987) TINA: A 3D Vision system for pick and place. *Proceedings 3rd Alvey Conference 65-72*

[10] Porrill,P., Pollard,S.B., Pridmore,T.P., Bowen,J.B., Mayhew,J.E.W., and Frisby,J.P. 1987 TINA: The Sheffield AIVRU vision system. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence, 2 1138-1144*

[11] Rygol,M., Brown,C. (1989) MARVIN: Multi-processor Architecture for Vision *Applying Transputer Based Parallel Machines, Proceeding oUG-10*

[12] Rygol,M., McLauchlan,P., Courtney,P., Pollard,S.B., Porrill,J., Brown,C., and Mayhew,J.E.W. Parallel 3D Vision for vehicle navigation and control. *Transputing '91. IOS Press ISSN 0925-4986*

[13] Rygol,M. (1991) Exploitation of MIMD Architecture for 3D machine vision *Phd Thesis, University of Sheffield*

[14] Sun Microsystems Inc. External Data Representations: Protocol Specification, 1990.