A Vision System for an Autonomous Underwater Vehicle

Ian Fitzgerald

October 1999

Supervisors:
Mr Samer Abdallah
Dr David Wettergreen
Dr Alex Zelinsky

A THESIS SUBMITTED FOR THE DEGREE OF BACHELOR OF ENGINEERING
AT THE AUSTRALIAN NATIONAL UNIVERSITY

DEPARTMENT OF ENGINEERING
RESEARCH SCHOOL OF INFORMATION SCIENCES AND ENGINEERING
THE AUSTRALIAN NATIONAL UNIVERSITY

Acknowledgements

This work could not have been completed without the assistance of a number of particularly helpful people. I would like to thank my project supervisors Samer Abdallah and David Wettergreen for their help with general project goal definitions and solutions to various problems that I have encountered during the course of the year. Thanks also to Alex Zelinsky for allowing undergraduates to use the mobile and underwater robotics laboratory and the computer equipment therein. Chris Gaskett and Harley Truong have also been helpful in suggesting various software debugging and hardware fabrication issues, respectively.

Most of the results obtained for this thesis could not have been obtained so accurately without the help of Martin Stonebridge, Bruce Mascord, Ron Cruikshank and Tom Rhymes. Thanks to Martin for his suggestions on calibration target fabrication and measurement. Bruce provided me with an introductory course in using the milling machine at RSPhysSE, and also provided necessary equipment for measuring the calibration target so accurately. Thanks also to Ron and Tom for allowing me the use of their workshops milling machine for the entire day.

I would also like to thank the entire Kambara group, and my undergraduate colleagues who have invested so much of their time into their respective projects. They have been great company over the past nine months. Finally a big thanks to my family for their support and for helping me out in any way they could.

Abstract

This thesis details the design and implementation of a fully integrated vision system that is intended for use on the Autonomous Underwater Vehicle, Kambara. The vision system comprises of three cameras; two of them implement stereo vision, while the other can be manoeuvred for observation or other general purposes.

This years project goal was to implement a system, which would use an automated calibration routine to determine the vision systems camera parameters. Using this information, a feature tracking and range estimation module was to be designed, which would direct the pan-tilt head towards the target.

A video digitising device driver was ported from Linux and enhanced to perform under a VxWorks environment, obtaining video data at up to 45Hz at a resolution of 320x240 pixels. Calibration was performed in laboratory and then in underwater conditions. Underwater calibration showed an increase in focal length by approximately 30%, and a significant reduction in lens distortion effects.

Using the calibration parameters, partially optimised outdoor tracking was performed at up to 20Hz, while simultaneously locating the 3D position (x,y,z) coordinate of the feature with respect to the camera reference frame. Position estimates were accurate to 15% via direct use of calibration parameters, and as close as 3% if the baseline magnitude was specified directly. This 3% error range was deduced to be consistent over the entire field of view, based on experimental results.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

The ocean exists as one of our most important natural resources, providing industry with both important minerals, and everlasting tidal energy. While currently under-utilised, it is certain that we will eventually turn to the ocean for its vast supply of natural resources. However, once industry moves to the ocean, problems arise. We have already designed colossal engineering masterpieces that ride on the ocean surface or in some cases below, but the task of maintenance is expensive. This is because the areas to be inspected and maintained are predominantly underwater.

The Australian National University (ANU) is currently developing an experimental Autonomous Underwater Vehicle (AUV) which could provide a convenient, cost effective solution to this problem. The AUV, named Kambara, is fitted with equipment required for computer vision that make underwater exploration/inspection and other underwater tasks significantly easier. Also, since the vehicle is to be autonomous, there will be no need to waste human effort performing manual vessel inspections when Kambara could do it automatically.

1.1 Kambara

Kambara is the open frame submersible robot being developed at ANU by the Systems Engineering Department of the Research School of Information Sciences and Engineering (RSISE). It consists of an Aluminium frame, 1.5 metres long, 1.3 metres wide and 0.9 metres high, with displaced volume of approximately 110 litres. Vehicle movement is actuated by 5 thrusters, allowing Kambara to travel at a maximum velocity of 1 metre per second, and a maximum turn rate of 60° per second. The mass of the frame, enclosures and thrusters is 66.5kg, with payload mass (including batteries and computer equipment) of 43.5kg.

Kambara is one of two submersibles that were assembled at the University of Sydney. These two submersibles form the basis for research work at both the University of Sydney (Oberon) and here at the ANU (Kambara, seen in Figure 1). Kambara and Oberon are very similar in appearance, however their project objectives differ substantially. Oberon is focussed more on terrain mapping, using sonar and other more conventional methods, is powered off-board and runs under Windows NT. Kambara on the other hand is controlled by a system that uses reinforcement learning to achieve its tasks, and is guided by a combination of computer vision, and sensor data.

There are two watertight enclosures on-board Kambara, mounted along its plane of symmetry. The lower enclosure houses the battery pack (six 12V sealed lead-acid batteries) which has a total capacity of 1200W. Within the lower enclosure also resides power distribution and charging sensors, along with depth, temperature and leakage sensors. A flexible coupling connects the upper and lower enclosures.

Figure 1: Kambara

The upper enclosure contains the sensitive computer and sensor equipment. This equipment includes:

1. CompactPCI 740 PowerPC processor (233MHz) running VxWorks for vision, filtering and communication.
2. Motorola 68332 processor for generating PWM signals to drive the thrusters.
3. "Industry Pack" (IP) modules for serial communication, digital IO and analog to digital conversion.
4. A sensor suite for state estimation.
5. The vision system components.

1.2 Vision System Components

The vision system comprises of a Sony pan-tilt-zoom EVI-D30 camera (pan-tilt-zoom commands communicated using the VISCA protocol via an RS-232 compatible serial interface). A PXC200 Imagenation frame-grabber is used for video digitising. It is equipped with a 4-input multiplexer to allow input selection.

Also, crucial to the vision system are the two Pulnix TMC-73M CCD cameras with watertight enclosures. These are attached to the Kambara frame on either side of the upper enclosure. The Pulnix cameras implement stereo vision, hence depth perception is possible. Cabling completes the summary of vision system hardware. [1]

1.3 Kambara Software Architecture

The vision system is only a small part of the Kambara software suite. Figure 2 shows the entire system when viewed from the top level. The issuing of control information is summarised simply. Commands are given from off-board routines (green). Commands are structured in a non-restrictive sense, such as "go to this target", rather than “travel forward 3m then turn left 90 degrees”. Such commands allow the on-board learning algorithm to determine the most appropriate solution to the problem. Commands can be issued via an off-board user interface, then once a command is issued; the vehicle manager (purple) relays instructions to the on-board software modules (blue).

The on-board modules communicate as follows: The Feature Tracker uses visual sensing algorithms to track features of interest, and uses the relative motion to help guide the Swim Generator. The Swim Generator (now referred to as the Vehicle Neurocontroller) performs the mapping of motion information to control signals, which can be interpreted by the Thruster Controller. The Peripheral Controller drives the other on-board devices (eg. cameras, and possibly later, scientific equipment). The Sensor Sampler collects sensor information and updates the controllers and State Estimator. The State Estimator produces estimates of vehicle position by filtering sensor data. Then the Telemetry Router sends state information and image data off-board.

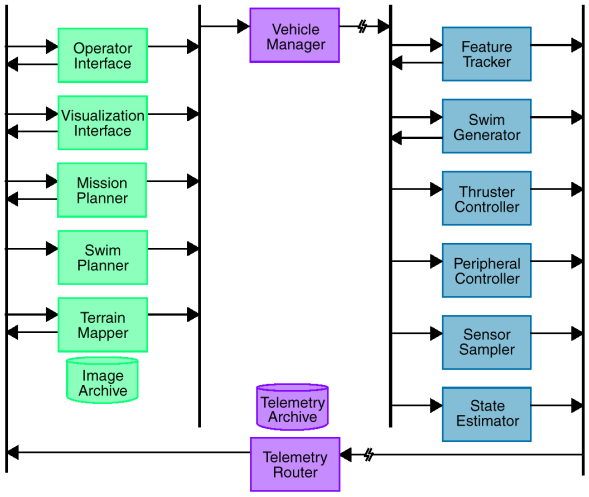


Figure 2: Kambara Software System [2]

The off-board Visualisation Interface provides visual interpretation of state information, while the Operator Interface implements a scheme to allow various modes of control – including Tele-operation. The Swim Planner interprets vehicle telemetry and uses this data to adjust the behaviour of Kambara. A Terrain Mapper would interpret visual data and render image maps of the underwater environment. Finally the Mission Planner sequences course changes to produce complex trajectories to autonomously navigate the vehicle to goal locations and carry out complete missions.

Note that the box labelled "Feature Tracker" represents the majority of the "vision system". The “Peripheral Controller” is a distributed module, part of which is implemented by the vision system also. This partial implementation refers to pan-tilt-zoom camera control. So the vision systems top-level role can be stated as to “provide a 3D estimation of the position of a feature in a pair of images”.

The vision system implementation started in 1998; hence the foundations for feature tracking and range estimation have already been laid. Before goals for this year are defined, it is informative to review what has been achieved previously.

1.4 Last Years Work

Previously, the Kambara Vision System project was focussed on developing the essential project foundations. Hardware was acquired and manufactured and software was written and tested for

feature tracking and range estimation. Most notably to this years interests, the effects of some key vision system parameters were investigated. [1]

1.4.1 Key Results and Implications

The key results (and implications of these results) that were derived from last year's project are listed in Table 1.

Result No.	***Observed Effect***	***Result***
1	Refractive Distortion	Refractive distortion without spherical lenses is insignificant
2	Length of stereo Baseline	Wider baseline gives better resolution for estimation over larger distances (2m or greater), but smaller baseline better for close up (closer than 0.6m).
3	Range Estimation Accuracy	Range estimation accurate to 10% (within 2m), and within the 15° angle range. Doubling the 15° angle range results in an extra 10% error
4	Stereo-Rig-To-Target Orientation	Range estimation with targets at greater than 15° from the visual axis of the stereo rig, yield errors above 10%
5	Edge Detection	Edge detection degrades performance significantly, and loses a large amount of colour information

Table 1: Key Results identified in Previous Project

1.4.1.1 Result 1: Refractive Distortion

Spherical lenses were recommended as the camera mount lens of choice, as refractive effects would be minimised. After analysing the effects, it was decided that there was no need for these lenses to be made, as distortions were minimal. This is a cheaper and faster approach to solving the problem. Note also, that using a robust calibration technique could compensate for the effect of refractive distortion.

1.4.1.2 Result 2: Length of Stereo Baseline

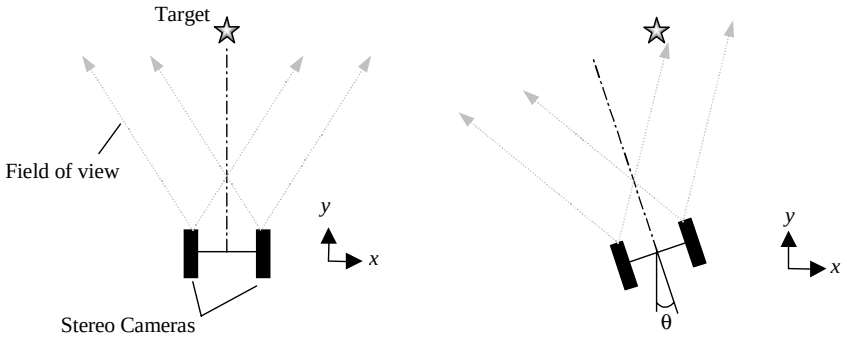
There is a trade-off to be made with respect to camera baseline selection. As the baseline is increased, tracking of targets at a farther distance becomes more accurate, but close up images are impossible to track since it is outside the stereo field of view. A baseline of 200mm gave accurate results for a range beginning at 300mm, whereas a 460mm baseline was suitable starting from a distance of 600mm. These results are important when it comes to deciding where to mount the cameras on Kambara. The distance at which Kambara will usually be tracking from is so far unknown and will best be found via experimentation.

1.4.1.3 Result 3: Range Estimation Accuracy

Within the 15° window as described above, and within 2m, range estimates were within the 10% window of error that was deemed acceptable. If the angle was doubled, the estimate error was also doubled to 20%. This further indicates the need to account for lens distortion. It may also have been attributed to refractive effects.

1.4.1.4 Result 4: Stereo-Rig-to-Target Orientation

Range estimation with targets at greater than 15° from the visual axis of the stereo rig, yield errors above 10%. The situation is described in Figure 3. This is largely due to the fact that lens distortion was not accounted for in the model used. Hence for larger field of view capability, this lens distortion must be accounted for.



As θ exceeds 15°, range estimate error become greater than 10%.

Figure 3: Stereo-Rig-To-Target Orientation Problem. [1]

1.4.1.5 Result 5: Edge Detection

The use of edge detection prior to correlation was largely ruled out as being too computationally expensive. This depends largely on the algorithm implementation. Correlation was performed last year using plain image extracts, which can cause problems that are not present when edge detection is used.

The problem arises that when a target passes between two areas of different background colour. Say for instance a fish that swims in front of a large rock. If the template includes background detail, the correlation level will be significantly affected. It is also highly probably that around 50% of the template will be background (not all objects are square!). When the background intensity accounts for anywhere near 50%, plain intensity correlation will not do.

1.4.2 Summary of Last Years Work

This work has given valuable insight into vision system hardware arrangement – optimal baseline, and optimal operating region. However the operating region is quite narrow (30° for less than 10% error), and most results were gathered when a target was located in the centre of the image. This was in order to develop some valid models, and collect data that did not depend on lens distortion. However, it is desirable to increase this region of accurate range estimation, to much larger angles, especially as the previous model doubled its error as the window was widened an extra 15°. So an important area to focus on this year is in improving the camera model such that lens distortion is taken into account. Edge detection was also largely written off, but as discussed, still may provide an important function.

Overall, these developments in 1998 left us with a portable module for stereo range estimation. This included template updating routines and range estimation routines (for parallel and converging cameras). For practical implementation on Kambara, there are a number of system modifications that need to be made. First, the image acquisition card needs to have a driver under VxWorks. The next level of complexity arrives when you wish to upgrade the range estimation to work for arbitrary (but reasonable, ie. facing the correct direction) camera orientations. This is to remove the need for camera alignment to be accurately performed and measured each time Kambara is deployed. Finally, integration of the pan-tilt camera tracking routine into the independent vision system application should be performed.

1.5 Project Objective

The ultimate goal of this project is to implement an application that can automatically calibrate the system cameras, such that it is possible to determine the lens parameters and the relative location and orientation of each camera in the vision system. Then, using this knowledge, the system should track a selectable target with the Sony pan-tilt-zoom camera, over a period of time. The AUV could then follow this object (using its thrusters and pan-tilt camera) until it is lost, or no longer of interest. This goal can be decomposed into a number of sub-tasks. Each sub-task needs to be completed before the next level is attempted, so this list runs in chronological order.

1. Finalise low level drivers.
2. Mount cameras on Kambara.
3. Calibrate camera parameters, and relative orientations.
4. Perform feature tracking such that position estimation is possible.
5. Obtain accurate (within 10%) position estimates for a wider field of view, for arbitrary but reasonable, camera orientations.
6. Perform pan-tilt mapping to point the camera at the feature of interest.

The project also aims to integrate the modules into an independent software package. Ideally, user interface and functional modules should be sufficiently separated to allow portability.

1.6 Thesis Outline

The vision system will be explored from the basic level of obtaining images through to pointing the pan-tilt camera at a target as it is tracked. Each chapter represents a particular element of the vision system and they are all self-contained modules that will be investigated individually.

First image acquisition will discussed, outlining the implementation and operation of the PXC200 frame-grabber. Other camera accessories will then be identified, which includes the VISCA driver and Sony camera mount. In order to use the vision system to make the required position estimates, certain information must be known about internal and external camera parameters. To solve this problem, an automated camera calibration technique has been implemented, and this will be discussed in the next chapter. The visual correspondence and position estimation modules work hand in hand to implement real-time position estimation, which will conclude the technical part of this report.

Appendix A provides a glossary of terms used in this thesis.
Appendix B lists the vision system camera specifications.
Appendix C defines how to implement memory mapping under VxWorks.
Appendix D goes into more depth on some subtle PXC200 device driver issues.
Appendix E details basic range estimation for parallel cameras.
Appendix F provides more details on selection of angles for the Sony camera mount.
Appendix G lists the key results from Tsai's calibration algorithm.
Appendix H provides a programmer reference for using the list structure.
Appendix I defines the algorithm used for position estimation.
Appendix J discusses some tracking optimisations that have been implemented so far.
Appendix K shows the raw calibration data.
Appendix L contains the original project proposal.

Chapter 2
The Vision System

The Vision System (Feature Tracker Module) is best described at its top level by looking at its required input/output relationship, thus defining the system interface. Once the system interface is defined, the internal structure can be designed.

Input is received from the Vehicle Manager (currently assumed to be directed via a user interface) to tell the system what to do. Response to these inputs is in the form of some vehicle action. One action is to perform visual sensing of the environment to implement feature tracking, and orient the pan-tilt head. Another is to provide the Neurocontroller with information about targets current position. Thus the flow of information can be summarised by Figure 4:

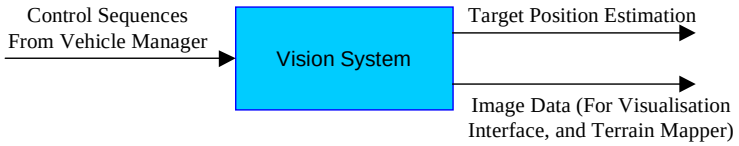


Figure 4: Vision System Interface

There is scope to add input from the State Estimation module, which could be used to implement a target locater module that would reliably and efficiently determine the image region in which the target should be located. This would increase the speed of the tracking algorithms substantially. Note however that if tracking frame rate is high enough, it is still possible to reduce the search window without relying on data from the State Estimation module.

2.1 Data Flow

The vision system is responsible for a range of operations. It follows the processing of data all the way from observing the target (a real world object), to determining some action to take based on the calculated position of this object. The data flow can be summarised as shown in Figure 5.

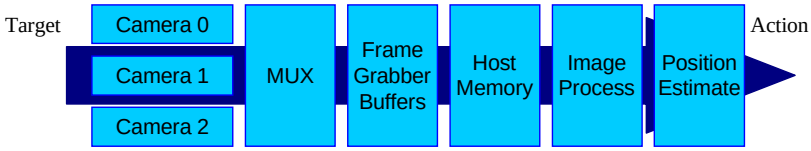


Figure 5: Vision System Data Flow

After image processing, the data is also sent to the User Interface Module, which in turn relays its information to the User.

2.2 Vision System Architecture

The vision system can be broken down into eight modules. Each module manages data (objects/structures) relevant to its function, or acts as a coordination module. They are explained at a high level, in Table 2.

Name	***Function***	***Comment***
TwoView	Handles display, Window Management, Application execution. Extends CView, a standard Windows User Interface (UI) C++ class.	*Platform dependant*
AppInterface	Maintains bitmap information for the user interface, twoView.	*Platform dependant*
VsystemInterface	Maintains calibration *objects* and grabber instances. Provides user/module interface primitives.	Platform independent
Stereo	Maintains calibration *parameters* and implements position estimation and pan-tilt mapping routines.	Platform independent
Track	Implements different tracking algorithms, all returning the target location within the image.	Platform independent
EviCamera	Implements camera control primitives.	Platform independent
GrabberBase	Interface to frame-grabber. Allows image acquisition.	Platform independent
Calib	Provides algorithms to process and perform calibration on input images.	Platform independent

Table 2: Vision System Modules

2.3 Module Dependencies

The modules have been identified and explained at a high level. It is also instructive to observe how the modules interact. This system hierarchy has been designed as shown in Figure 6.

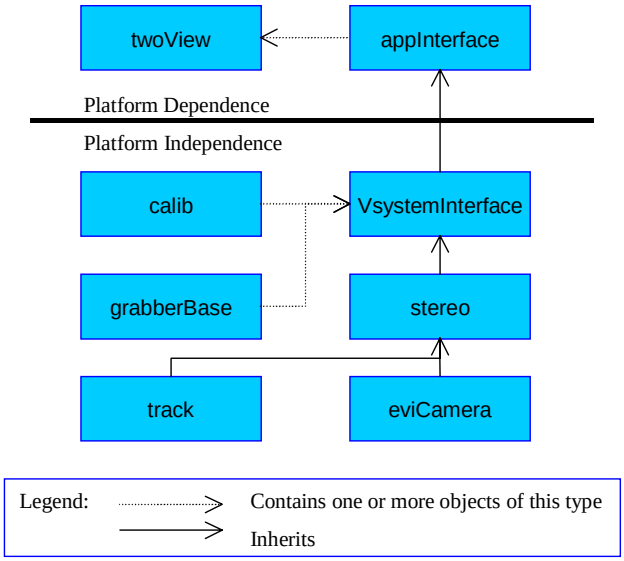


Figure 6: Module Dependencies

For graphical development work on any platform, a user interface (front-end) is required. Unfortunately user interfaces are inherently platform dependent (eg. “twoView”), hence the most useful interface layer to provide is one which implements a number of commonly used primitives. This is the role of “VsystemInterface”, and it provides primitives to:

- Select a template
- Start tracking a feature
- Stop tracking a feature
- Update the current template
- Move the pan-tilt head
- Calibrate Kambara’s cameras.
- Obtain a set of images from Kambara’s cameras.
- Handle certain events.

Note that for development on platforms other than Windows, the top two modules (twoView and appInterface) will need to be ported to the new environment. The module dependencies have now been summarised; each will now be examined in more detail.

Chapter 3
Image Acquisition

The vision system equipment is capable of capturing real world image data and storing it for use, however this information is not readily available to software applications. That is where the need for a device driver to manage the frame-grabber arises. A device driver is a software entity whose sole purpose is to provide user programs with a means to access the data derived from a, usually physical, device.

3.1 Requirements

The PXC200 device driver had to be designed to meet a number of functional requirements. This set of minimum requirements ensures that the vision system will be able to perform the functions required of it by other Kambara software entities. The requirements are listed in Table 3, and appear in order of importance (top being essential, bottom being desirable).

Id	***Function***	***Measure***	***Required***
1	Provide image data to user applications	True/False	True
2	Provide data in real-time	Frame rate (Hz)	>30 Hz
3	Allows three cameras to be used via MUX	True/False	True
4	Provide means of ensuring image integrity	True/False	True
5	Minimise frame loss due to MUX swap	Number of frames	Minimise
6	Allow client server architecture	True/False	True
7	Allow use on other platforms	Platform	Intel[1]

Table 3: Device Driver Requirements

3.2 Architecture

The device driver software architecture has been adapted from an existing Linux device driver, and is illustrated in Figure 7. As shown, there are a number of distinct layers in this application. The two most prominent layers are the application and PXC200 layers. The latter refers to the on-board processor on the PXC200, whose operation can be varied by programming it with different register settings, and is the base upon which the entire application is built.

The top layer is intended to be an interface to user programs. It is written in C, and contains function definitions that emulate those given in the Imagenation PXC200 library written for the Microsoft Windows environment. This layer is not essential - the same functions could be implemented from an application level using I/O control functions (otherwise known as 'ioctl' calls

[1] ***VxWorks*** on an Intel based computer. Under windows the standard drivers can be used.

directly to the driver), however it does implement some useful high level features. These features include client-server primitives, and also the ability to modify image acquisition parameters while grabbing.

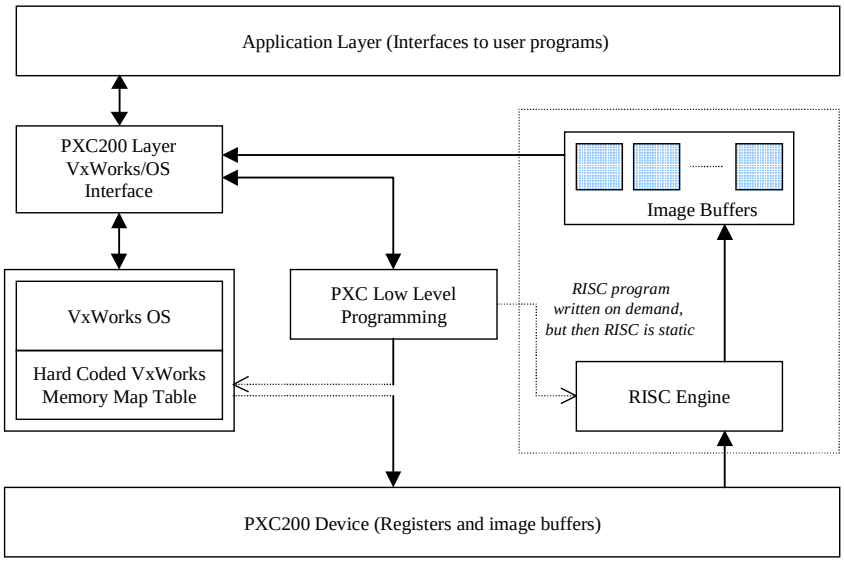


Figure 7: Device Driver Architecture

The motivation for client-server ability is not plain to see. It is possible to assume that image acquisition would be prompted by a signal during Kambara boot-up, and would then continue indefinitely. A client-server model would implement a way of coordinating, comparing and monitoring image acquisition results.

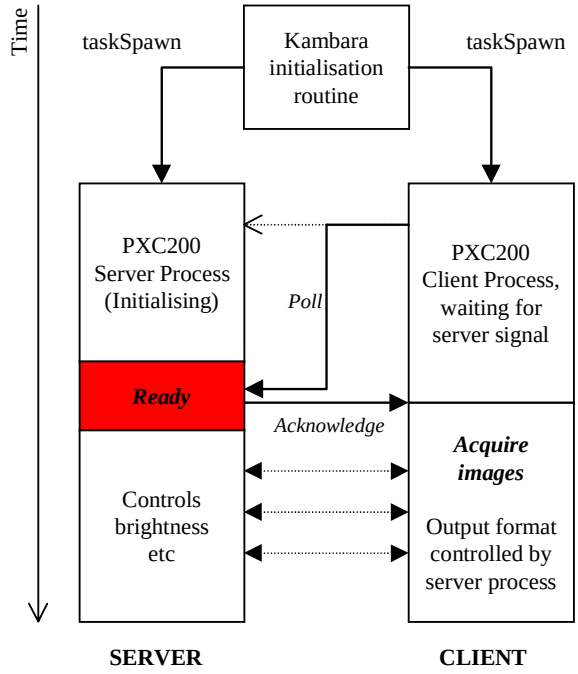


Figure 8: Client Server Implementation

For instance, if one of the Pulnix camera aperture settings had been changed, one image would appear significantly brighter than the other would. This could cause problems in correlation and feature tracking. In an attempt to avoid this problem, a control loop (the server) could be instructed to observe the average image intensities coming from both cameras. If the difference was too large, the server could make contrast and brightness adjustments.

Meanwhile the client would continue to acquire images on demand, not knowing that the server is actually monitoring and adjusting various settings at the same time. Also not that due to mutual exclusion semaphores, there would be no instance where the server would interrupt a client image read. This client server architecture is summarised in Figure 8.

3.3 Implementation

There are a number of implementation issues that must be satisfied in order for the device driver to be useful. These issues are now discussed.

3.3.1 Platform Independence

The Kambara team has access to a number of different operating systems. VxWorks runs on the Pentium II 300 development platform, whereas the Kambara uses the 233MHz PowerPC. Both systems handle interrupts slightly differently, and differ in terms of endian compatibility. Intel uses little endian, whereas PowerPC uses big. This incompatibility can be fixed through byte swapping. The final driver code allows modification to suit the platform depending on the constant PPC, which can be defined at compile time.

3.3.2 Register Programming

Before images can be acquired, the PXC200 device has to be programmed with appropriate data. This includes the address (or addresses) of the memory region to which data will be written. The mechanism by which registers are programmed is through memory mapping. Memory mapping is described in Appendix C.

3.3.3 Image Acquisition

Image acquisition is certainly the most important functionality implemented by the device driver. However at the lower level, it is quite a complex issue. The particular card to be used is to be attached to compact PCI port (and under development, a standard PCI port). Once the driver registers are programmed, and the driver installed, image data is transferred from the PXC200 FIFO buffers to host memory via the PCI bus. The speed of this process lies in the fact that once set up correctly, data is sent via DMA (Direct Memory Access), rather than via a standard memory copying, or memory mapping. The PCI device is programmed such that it has permission to write to a certain region of memory, and then begins doing so when prompted. Image data is then transferred to this known location *without* intervention from, the operating system. This is the main difference between memory mapping and DMA.

It is up to the manufacturer to decide just how the data is transferred from the frame grabber buffers to host memory. In the case of the Imagenation card, the use of one or two RISC (Reduced Instruction Set Computer) processes is employed. There is usually one RISC program for the even and one for the odd **video fields**. The fields are then interleaved to produce the image. The PXC200 uses a BT848 chip, which executes this RISC, process (RISC programming instructions are

available via [13]). They include SKIP, JUMP, WRITE and SYNC. The commands are identified by a short bit sequence which, and each expect a certain number of arguments to follow. The bit streams are written to host memory, the start address is programmed on the PXC (into the BT848), and then RISC execution is started.

Note that each RISC program is *static*. Simply changing the value of a register or variable cannot change the RISC program. The RISC program is essentially hard coded into host memory by the device driver as a set of bit sequences that indicate image width, height and bits per pixel. If, after the RISC program has been running, these parameters change (resolution or images format change: 24- to 32-bit for instance), the RISC program will not be valid and requires halting, de-allocating and rewriting. Hence changing resolution should not be done often when high speed is required. It is possible however, to change between different program stages, eg. the transition between calibration and tracking where the emphasis shifts from accuracy to speed. Figure 9 gives a graphical account of the data flow associated with an image acquisition.

The two RISC processes are represented by the single RISC box shown in Figure 9. This is merely to aid in diagram simplicity. Memory mapping and DMA transfer is illustrated also. Note that the BT848 can be programmed to acquire images in different (4) resolutions. High-resolution (640x480) requires the use of two video fields, but low (320x240 or under) does not necessarily need this extra speed. It was decided to eradicate the second field when capturing in lower than 640x480.

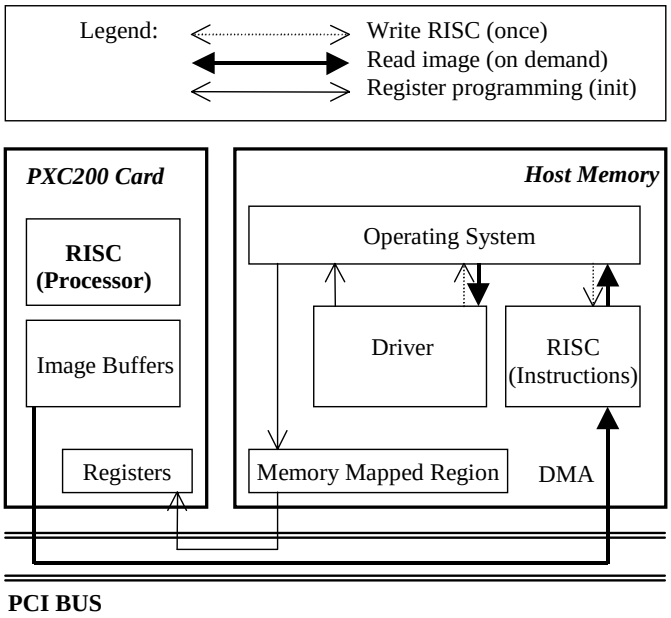


Figure 9: Image Acquisition

3.3.4 Interlace

The use of two fields to interleave the video signal has been described. Note that true synchronisation between these fields is difficult and for fast moving objects, you will notice a slight time lag between the images depicted by the odd and even fields. This results in a problem known as interlacing. Interlacing creates problems for image processing, due to the acquired image not being a true depiction of the image scene. To remedy this, interlace has to be avoided, and hence 640x480 is not a viable option. Since 320x240 does not require the second RISC field – and speed

is reasonable[2] - it is the most desirable resolution to use for image processing. Using this resolution (and no second frame), fast moving images do not appear interlaced, but blurred. There is nothing that can be done about this blurring effect, but it is an improvement over interlace – especially considering that the image will be smoothed using Gaussian operators[3] anyway.

Interlace is the most evident problem facing the design of the device driver. There are other more subtle issues associated with using this hardware to acquire still frames. These issues need to be addressed if programming the PXC200, and are outlined in Appendix D.

3.3.5 Speed

Requirement 2 indicates that maximising frame rate is of a high priority. Without high-speed image acquisition, real-time image processing would not be possible, no matter how good the processing algorithms may be. 30Hz was a nominal minimum requirement that was selected as a ballpark speed indication. Speed maximisation is the ultimate goal.

We have noted that an image resolution of no greater than 320x240 will be used hence speed measurements for this size image will be examined. The frame grabber acquisition routine is kept as simple as possible, and essentially consists of one 'memcpy' command. Image acquisition speed is thus dependent on the processor speed. Our VxWorks development platform was an Intel, Pentium II at 300MHz, whereas Kambara will be running on a PowerPC at 233MHz. The UNIX command, 'timexN' was used to repeatedly measure image acquisition times, until the uncertainty in timing was less than 2%. Results are listed in Table 4.

Resolution	***Speed***	***Accuracy***
640x480	22Hz	±2%
320x240	45Hz	±2%

Table 4: Image Acquisition Speed

3.3.6 Multiplexer Latency

The third most important requirement is that we are able to use the frame grabber to gather image data from three cameras, which means that a multiplexer (MUX) must be used. The multiplexer allow four inputs to the frame grabber, and the manufacturer assured us that we would only drop one frame per MUX swap.

Testing using the Windows driver indicates that MUX swapping does cause a problem. Under Windows the frame grabber device driver does not block (cease acquiring images) until the MUX has made its transition. This is one problem that has been avoided with the VxWorks device driver. It blocks until a fixed number of interrupts have been generated that signal that a complete image has been acquired. Currently the number is fixed at 3.

This accounts for the case where an image request is made half way through an image transfer. When the request is made, the transfer finishes and accounts for 1 interrupt count. Imagenation have advised that 1 frame loss would be incurred while swapping frames, so the next indicates the completion of the frame which is partially derived from one MUX and partially from another. Accounting for any timing errors, in an effort to ensure data integrity, one more interrupt should be

[2] It would be possible to use a single field in 640x480, but image acquisition would be too slow.

[3] It is common to remove image noise by convolving the image with a Gaussian kernel, which has a 'smoothing' effect on the image.

awaited before continuing. This problem has been overcome under windows simply by inserting a "Sleep(50)" command (pause for 50 clock ticks).

3.3.7 Visualization

Development and verification that images are being acquired accurately is useful, but not essential (once the debugging stage is over). The image visualisation technique used involved writing "Portable Picture Map" (PPM) files. This is the simplest of colour image file formats, and requires no compression. Hence it is significantly slower than some formats. This trade-off is justified because high-speed video is to be transferred via a cable link to the surface, rather than dependence on PPM image transfer.

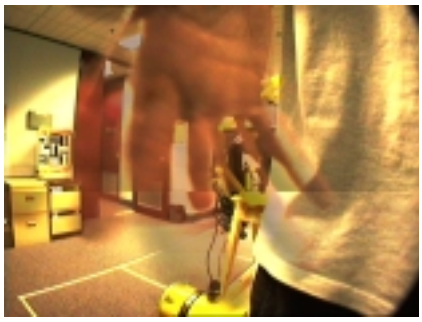


Figure 10: Image Overlap Effect

3.4 Verification

The final product can be assessed based on the requirements listed in Table 3.

Function	***Status***
Provide image data to user applications	True
Provide data in real-time	45 Hz @ 320x240
Allows three cameras to be used via MUX	True
Provide means of ensuring image integrity	True
Minimise frame loss due to MUX swap	Interrupt count of 3 (Between 2-3 frames)
Allow client server architecture	True
Allow use on *other* platforms	VxWorks: Intel and PowerPC

Table 5: Device Driver Requirements Verification

3.5 Image Acquisition: Conclusion

The identified requirements have all been met. The only aspect that might require future attention is the issue of frame loss due to MUX swapping. Otherwise, Kambara has an effective device driver, capable of acquiring images at 45±0.9Hz at a resolution of 320x240, and faster at lower resolutions.

Chapter 4
Camera Accessories

This chapter describes the process associated with the specifications of the Sony camera mount and pan-tilt-zoom device driver. The Sony camera is not important from the viewpoint of image processing. It provides the means to get a clear picture of what Kambara is looking at, and will also offer the possibility of visual servoing using camera position information.

4.1 Camera Mount

The Sony pan-tilt-zoom camera is to be mounted within the upper enclosure on the Kambara, it has to be held rigidly and should be placed in a consistent position each time the vision system is in use. (Actually, the calibration system should have the potential to eradicate this requirement for repeatability, but that functionality is not yet in use). The design characteristics of the mount are made particularly simple due to the camera's manoeuvrability, shown in Figure 11.

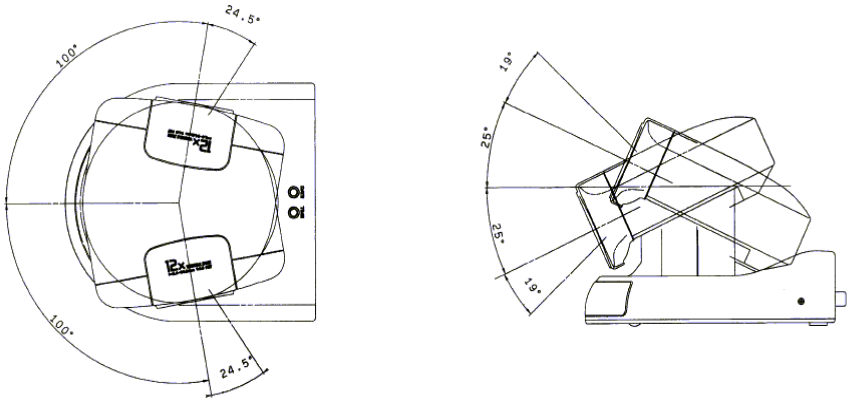


Figure 11: Camera Maneuverability

The mount was eventually attached to a simple drawer mechanism in the upper enclosure, fixed to point downwards at an angle of 36±5°. This was under the assumption that the majority of observation tasks would be below Kambara (eg. coral, fish). For a detailed account of camera mount considerations, refer to Appendix F.

Vibrations may cause a slight problem in receiving a clear image from the Sony camera. When Kambara is powered up, and the thrusters and other electronics are all running, this may be a problem. Since Kambara has not been fully tested in this respect it is hard to judge the magnitude of this problem. If it does seem to yield bad images, then a damping layer might need to be added onto the mount, eg. a thin rubber strip.

4.2 Pan-Tilt-Zoom Device Driver

The device driver had to be capable of sending packets via the computers serial port, formatted to the VISCA specification (available from [14]). There were two options here – write our own or find an existing driver. Sony EVI series cameras are quite common and the chances of finding a driver on the Internet were high.

Obviously, the choice was simple – we would use the existing solution. It was platform independent – able to be used under Windows, UNIX and Linux (as well as some others). Its architecture is very simple which makes the module easy to port between the different platforms. The architecture can be described in Figure 12.

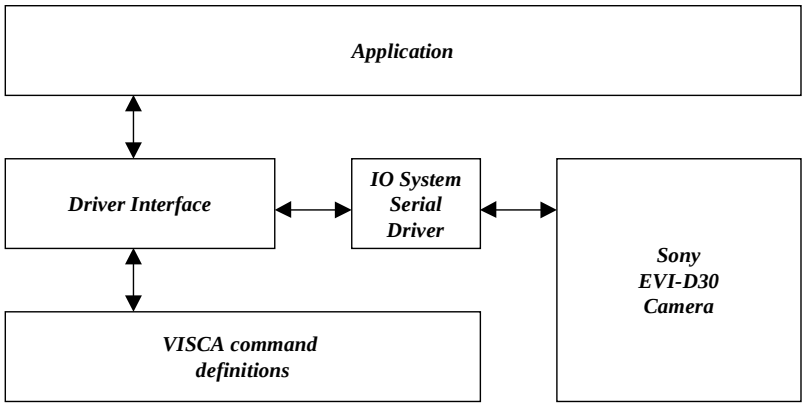


Figure 12: VISCA Driver Architecture

The only problem with this driver module was that support was not included for VxWorks. This needed to be added. It was quite a simple addition due to the structure of the driver. Note that it does not conform to the VxWorks device driver standards – it is not registered by the IO system. Rather, it is simply included by client programs and creates and RS-232 connection to the camera via a specified serial port (in our case, on the Intel PC, via the standard serial device "/tyCo/1").

4.2.1 Potential Driver Problems

While the driver is simple in architecture and design, and is multi-platform compatible, it relies on a strict client server relationship between the driver and the camera. Most commands are specified in terms of the VISCA packet length, the length of the VISCA packet acknowledgment, and the VISCA command sequence (up to 15 characters). Some commands have a 'wait' option while others do not.

Without an option to wait, the command is issued to the RS-232 connection, and execution is paused until the correct number of acknowledgments has been returned. For some commands, an inquiry is required before an adjustment is made. This is unfortunate as on a single processor, program execution blocks until the response is received, which slows things down considerably.

Thus, for better performance, a separate process dedicated to sending/receiving data to/from the camera should be utilised. Currently this is not the case, but it is considered to be a simple enough task to implement, using pipes (a standard construct under VxWorks).

4.3 Sony Camera Accessories: Conclusion

The camera mount has been bent into shape and fitted. After some modifications (to support it more securely within the upper enclosure) it appears to be rigid enough to support the camera. The only unresolved issue is the effect of vibrations. However, it is assumed that this problem will be minimal.

The VISCA driver was a simple port to VxWorks. The only issue of concern here is whether the standard "/tyCo/x" serial drivers will be applicable to our case. If not, a separate serial driver may need to be written. Potential problems have been identified, and remedies suggested. Once the vision system functions as a single, stable entity under windows, it will be ready for porting to VxWorks, and this is when these issues will become more apparent.

Chapter 5
Camera Calibration

The Vision System for Kambara relies heavily on knowing certain information about the camera's that it uses when estimating target range and direction. This is why a calibration routine is required. With a suite of cameras, calibration can also be used to determine each camera position with respect to the others; this places less importance on accurately measuring the camera separations and rotations, which can be inaccurate.

Camera calibration is one of the first steps required in any classical vision system (with some more recently developed systems eradicating the need for specific calibration). Since this is such a common task in the field of computer vision, many algorithms have been developed in order to solve the problem. One of the most widely documented algorithms for camera calibration was written by Roger Tsai, and has been used extensively in this project. [3]

The calibration parameters that can be derived through using Tsai's algorithm are:

Intrinsic Parameters

- f, the effective focal length of the camera
- κ, the first order radial lens distortion coefficient
- C_x, C_y, the coordinates of the centre of radial lens distortion
- s_x, horizontal scale factor that accounts for hardware timing errors

Extrinsic Parameters

- R_x, R_y, R_z, rotation angles (roll, pitch and yaw) for the transform between world and camera coordinate frames. These are also given as individual rotation matrix elements: r_1 through r_9.
- T_x, T_y, T_z, translational components for the transform between the world and camera coordinate frames.

Previous calibration attempts were empirical and involved aligning the cameras such that an object appeared on the same pixel row on each camera. As noted from the results of previous work, if the object were further than 2m away, and outside of the 30° cone centred along the optical axis of the stereo rig, range estimation results were greater than 10% off the desired amount. The error has been attributed to lens distortion, and partially refractive distortion, so hopefully both may be accounted for by implementing Tsai's calibration model.

5.1 Calibration Theory

There are two levels of theory to consider when using Tsai's calibration technique. The first and most important is how the camera model parameters can be used to make position estimations. Secondly, there are the details of the calibration method itself, which are quite complex. Tsai's

calibration package is available via [9]. The inputs and outputs to this calibration algorithm are shown in Figure 13.

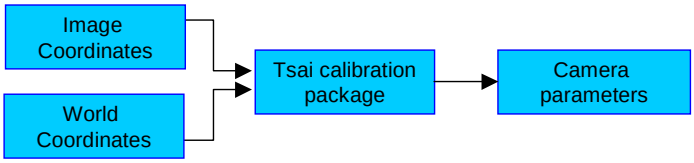


Figure 13: Tsai's Calibration Interface

Since the calibration package can be viewed as a black box inside the vision system, it will not be explained in depth, but the method will be outlined briefly. The more relevant derivation involves relating the world coordinates to image coordinates via Tsai's camera model.

5.1.1 Tsai's Camera Model

Tsai's camera model is based on the pinhole camera model[4], with some important differences. The pinhole model assumes a thin lens that provides no distortion is being used, but in reality this model is not strictly correct. The thin lens diverts incident light rays that are parallel to the optical axis, such that they intersect at f, the lens focal point

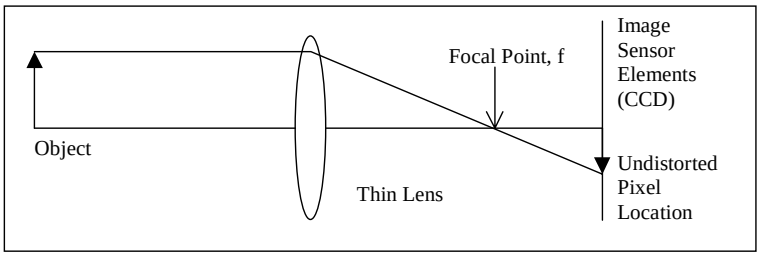


Figure 14: Pinhole Camera Model

This is where Tsai's model is superior; it takes this radial distortion into account. When lens distortion is taken into account, distorted pixel locations can be mapped back to the undistorted value. This is important for position estimation, as an error of one-pixel results in over 3.4mm error in range. [1]

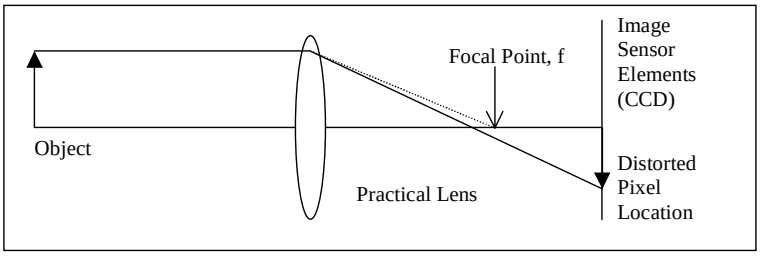


Figure 15: Effect of a practical lens

Once the camera parameters are determined, Tsai's model accounts for this distortion, and in theory, can eliminate this range error by accounting for lens distortion.

[4] The term "pinhole" refers to the analogy of the focal point being a hole in an opaque sheet, which is then held in front of the image plane.

Analysing Tsai's model is important for two reasons. First, expressions need to be derived in terms of calibration parameters that relate the world coordinates of and object, to coordinates system used within the image observed in memory. This is the motivation behind camera calibration. Second, these relationships can then be rearranged, an extra camera added, then we can relate the coordinates of two images, to world coordinates in which both images are located. Tsai's camera model is depicted in Figure 16. The camera model also assumes that we already know the camera parameters. For the derivation of Tsai's governing equations, refer to Appendix G.

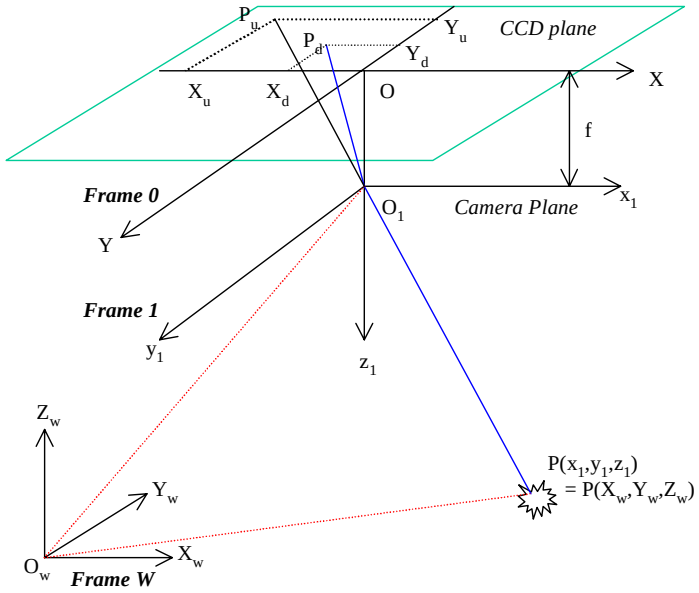


Figure 16: Tsai's Camera Model (Rear Projection)

This model calculates extrinsic parameters with respect to a world coordinate frame, W. This is particularly useful if more than one camera is being used. If a suite of cameras is calibrated using images taken at the same (or approximately the same) time, the extrinsic parameters will provide a relative description of where each camera is in free space (with respect to W). World coordinates can be related to camera coordinates via (1) and hence other camera coordinate systems that are defined in terms of the same world coordinate frame.

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = R \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} + T \qquad \text{……..} \qquad \textbf{(1)}$$

This removes the need for accurate alignment and measurement of the hardware when installed on Kambara. The accuracy in measurement is required only for the calibration target.

5.1.2 Calibration Target

The calibration algorithm works by inspecting the correspondence between a given set of measured data points that are known in advance, and a set of data points that are observed in an image. Once the data points are sorted in the same order, methods such as singular value decomposition can be used to solve the over parameterised system of data points.

So the unknown is the location of the calibration points within the image. Many techniques can be used to determine these point locations. For simplicity and efficiency, it has been decided that the target will consist of two planes (to allow non-coplanar calibration). An array of boxes is printed on each plane. The target background is black, while the foreground is white, which creates a high contrast region such that edge detection and other routines will find it easier to detect these region boundaries. The corner of each box is a calibration point. Thus once all of the boxes are located, a line fitting and intersection routine can determine the intersection points (to sub-pixel accuracy).

The accuracy of the calibration target is important. A rule of thumb is that the calibration target should be measured accurately to a tolerance of one order of magnitude (or more), better than the position estimation tolerance required. If the vision system measurements were to be accurate to the nearest centimetre, the calibration target should be accurate to at least the nearest millimetre. Also, note that Tsai's algorithm works best as the number of points is increased. This is why so many boxes were placed on the target.

The calibration software needs to be able to pick out the calibration target from its surrounding, while not knowing anything about its distance and orientation from the camera. Once some information about target geometry is known, different methods can be used to extract the target from a given image, with some reliability. The specifications given to “Inkline” for printing the calibration pattern are shown in Figure 17.

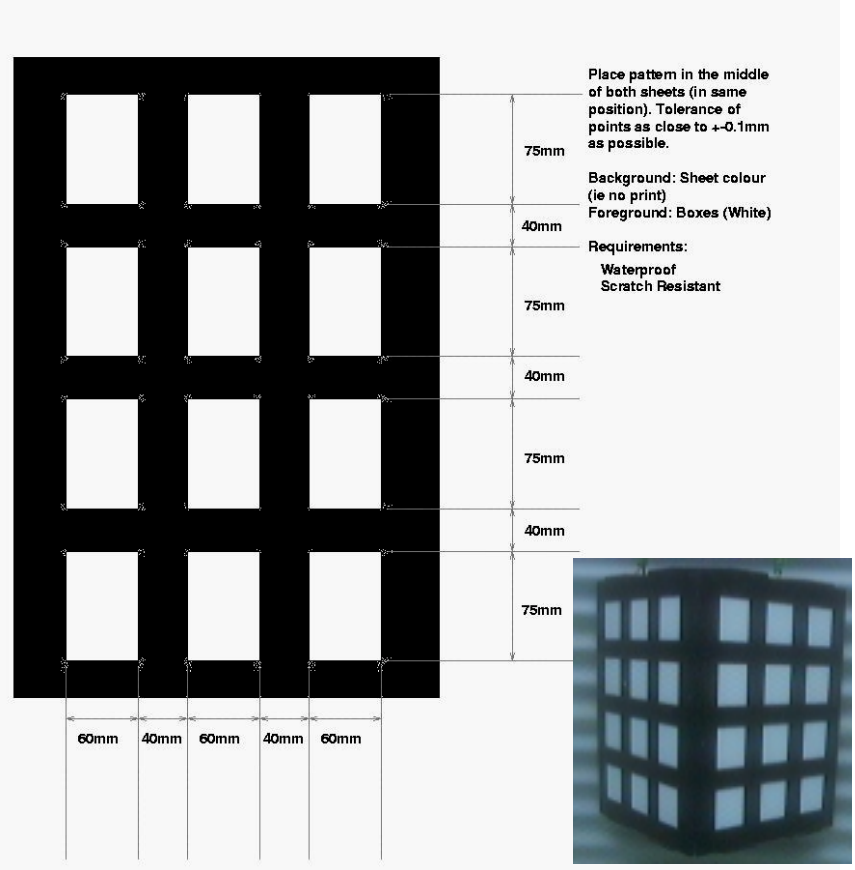


Figure 17: Specifications for Target Printing (and eventual target)

What is known about this target, is that the height of the boxes is greater than the width. We can also make assumptions as to how the calibration target is held – at what angle or how close. One requirement is that the angle window it is held at, should not be too narrow, and about 1-3 metres away (far enough such that all cameras can see the target simultaneously, but close enough so that the boxes do not appear too small.

There is a problem with the current calibration target layout. Each plane is made out of Delrin (black homo-acetyl polymer [5]), which has quite a reflective surface. Thus it is advisable to hold the target at a downward angle to avoid reflection off the sun.

5.2 Calibration Software

The calibration software is possibly the most complex part of the entire vision system. Hence, satisfying a stringent set of calibration requirements is essential.

5.2.1 Requirements

There have been no formal requirements laid down by the Kambara team, as to how the calibration is to perform. We do know we want our position estimations to be (at least) within 10% of the correct value, and ±2-3cm is desirable. Speed is also not a problem – calibration is not done in real time, but does require someone to hold a target before the Kambara for a period of time. Held still for periods of $1/10^{th}$ of a second, but generally held in front of Kambara until the system calibrates, which can be up to 2 minutes. It may also prevent the rest of the system from initialising, so speed is desirable. Accuracy and reliability are the key requirements, but a full set of loose requirements are given in Table 6 (in order of importance):

Priority	***Requirement***	***Measure***
1	Accuracy in target measurement	±0.01mm or better.
2	Reliability: Should know if there is something wrong with the calculated values	Calibration routine does not exit until parameters correspond (bounding values) to ground truth
3	Speed: as quick as possible for accurate results	Full calibration in less than 2 minutes.

Table 6: Calibration Requirements

5.2.2 Module Architecture

The modules contained in the line fitting and edge detection part of the calibration system have been written externally to this project, and hence will not be covered in great detail. It is however important to know how they tie into the system as a whole.

Edge detection and line fitting are two of the most common image-processing tools used, and hence there are a number of different algorithms and, numerous packages available via the Internet, which perform this function. The solution that was finally located, is a small part of a complete publicly available image processing software system, TargetJR. It is accessible via [8].

The entire TargetJR system is written in C++, and highly dependant on lower level components in the hierarchy. The required components were extracted, and a universal base class ("IanBase") was reverse-engineered. This base class allowed the modules to work without the reliance on the 200+ source files it otherwise required.

The calibration module uses "CannyOX" and "FitLines" [11] to generate and further process **edgel chains**. First, edgel chains are generated via "CannyOX", and then some custom processing is done to filter outliers. Next lines are fitted to the remaining edgel chains, and these lines are then extended until they intersect with their closest neighbour. Line fitting uses orthogonal regression, which in turn requires the use of a singular value decomposition (SVD) routine. [12] The architecture is described in Figure 18.

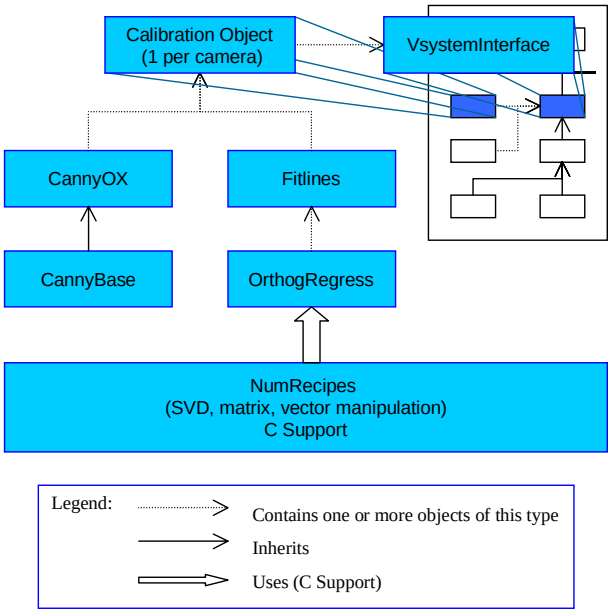


Figure 18: Calibration Module Heirarchy

"IanBase" is used by all vision system modules and contains definitions for Image, Edge, and many other base classes used during calibration and general image processing. It also implements a LIFO list class that is essential for successful integration of the "TargetJR" objects. See Appendix H for a specification of this LIFO list class (which also has the capacity to be converted into a FIFO list structure). The key idea in using these modules is that it is a list that contains the important information. Both "CannyOX" and "FitLines" can be viewed as "Black Boxes" that perform list manipulation.

5.2.3 Components of the Calibration Software

The calibration routine involves a number of processing stages. For complete calibration success, all three cameras need to be simultaneously calibrated. The top-level coordination is done in "twoView", acquiring images and calibrating the system if required. It is then up to the calibration routine to determine the parameters for each camera. If one calibration stage fails, for one of the cameras, the entire process has to be repeated. The calibration stages are shown in Figure 19. On the right, are calibration stages, on the left is the simple coordination algorithm used when images are acquired. The red boxes indicate the use of existing packages; blue are my custom solutions.

5.2.4 Edge detection and Line Fitting

These tasks are achieved through the use of the TargetJR modules. There is a stage in between that is of ultimate importance – outlier rejection. Edge detection can return a list of edges that are

noticeable within an image, and line fitting can accurately fit lines to these edges, but what is required, is a knowledge of which lines correspond to a box edge. This is one of the hardest tasks to be performed reliably. A pre-processing and outlier rejection scheme has been devised to identify which lines are actually parts of a box.

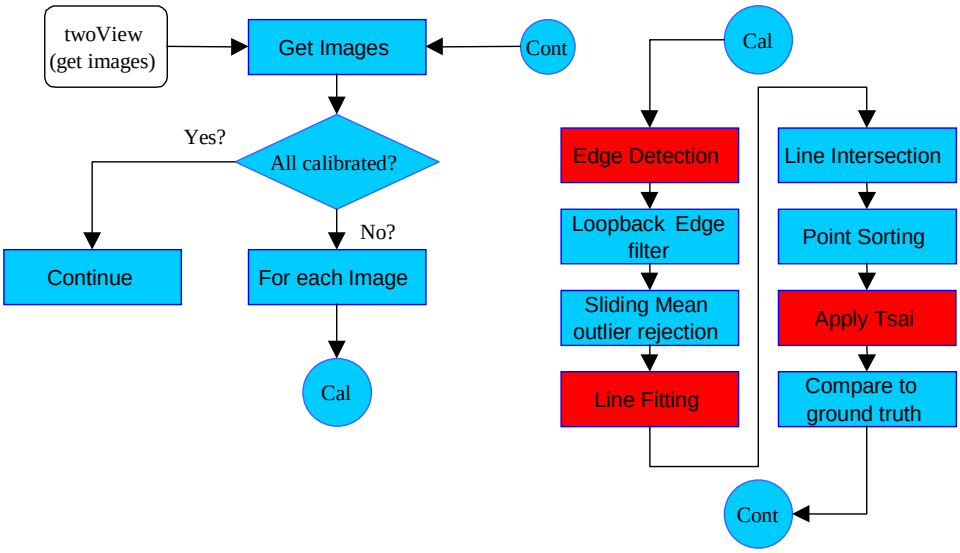


Figure 19: Calibration Steps

5.2.5 Loop-back Edge Filter

Outlier detection has to be done, but it would be nice to have some means of removing edgel chains that are certain to be outliers beforehand. For instance, the edge detection routine returns an edgel list. Each element in the list corresponds to a chain of edge points, which may correspond to the perimeter of a calibration box. So since a calibration box is a closed loop edgel chain, the list could be filtered to remove all edgel chains that did not finish near where they started. Also, edgel chain length thresholding was used, to specify that only edgel chains containing N or more pixels would be accepted. This significantly reduces the amount of work needed to be done by the outlier rejection scheme (two of which were developed).

5.2.6 Sliding Mean

This is the solution to the outlier rejection problem. It takes place between the edge detection (and edge filtering) and line fitting routines, and thus deals with a list of edgel chains. It models outliers as white noise - it expects an even distribution of outliers across the entire image. Note that even if this is not the case, the method is still robust.

The key idea is that there are N boxes that are assumed to be within the image (in the case of the calibration target being used, N = 24. It is expected that the calibration boxes will constitute the majority of the closed loop edgel chains within the list. The next thing to do is reduce the amount of filtering required. Thus, instead of dealing with a list of edgel chains, a list of points was derived. The list of points approximates the centroid location of each closed loop edgel chain (assuming symmetry). The x and y coordinates of this new list can then be averaged, yielding a start point.

An incremental approach is used to remove outliers until the required number of points is present. The x and y means are evaluated, then the Euclidean distance from the mean location to every point

is calculated, with the centroid of largest distance being removed from the list. This process is repeated until the correct number of points is present.

5.2.6.1 Implementation

The success of this method is heavily dependent on the environment under which the images are being observed. In practice, calibration would occur underwater, where vision range is vastly reduced. Under these situations, it is quite reasonable to expect this method to work, and work reliably. However, in a lab environment, the situation is worse (which is not often the case). Objects in the lab are quite often square, for example computer screens, books, boxes, shelves etc. Thus the number of outliers that are likely to be experienced in the lab are significantly higher than will be present in practice.

But this does give a good environment for developing in sub-optimal conditions. The sliding mean routine has been devised such that it writes an output file once per camera. This output file has been written such that a Matlab script could parse and display the results. An example of a filtering result is given in Figure 20. The red circles are candidate centroids, the blue asterisk is the initial mean (x, y) coordinate, and the green dots trace the path of the mean as the number of outliers is decreased. The red asterisks correspond to where the algorithm believes the calibration boxes are.

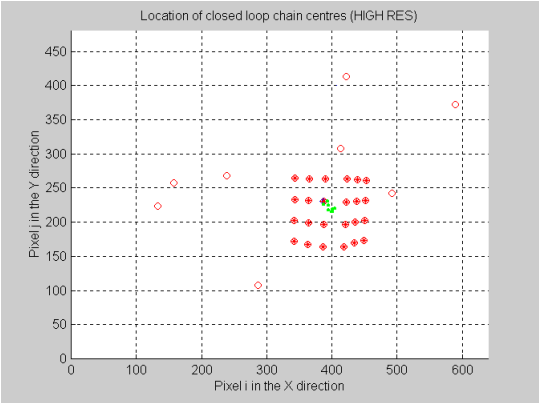


Figure 20: Expected Result

As shown, the algorithm worked fine, even in the presence of outliers. Actually, the number of outliers does not make too much of a large difference, the major problem is the location of the outliers. This is inherently going to be a problem when using a method such as this. The situation is described in Figure 21. Blue indicates the result of the sliding mean filter, red indicates an outlier, and the filled circles correspond to the actual box centroids we are hoping to isolate. The green star is the final mean point and the maximum Euclidean distance from the mean to any accepted point is shown by the circle (call this the "forbidden region").

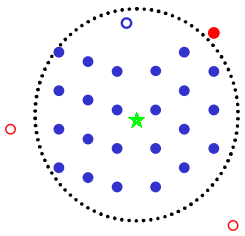


Figure 21: Pathological Outlier

As can be seen, if an outlier is positioned in an awkward position, the algorithm fails. This has been simulated in the laboratory. See Figure 22. Note that it was not the extra number of outliers in this case that caused the problem, just the outlier that is located within the forbidden region.

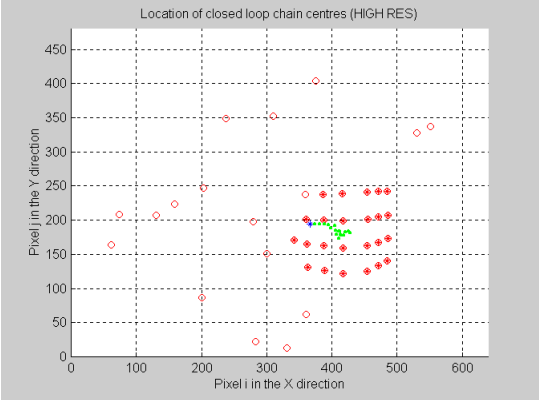


Figure 22: Effect of Close Outlier

Another problem is when a box is made invisible due to the sun reflecting off the calibration target. In this case outliers are not the problem, but there is a lack of a point (not 24 that are detected), and hence one of the outliers will be assumed to be part of the target.

5.2.6.2 Remedies

To reduce the effect of pathological outliers, it is possible to introduce the use of the convex hull. The convex hull is a polygon that encloses N points, whose area (or volume) is minimized. An example of the convex hull (dotted region) in the pathological case described in Figure 21 is shown in Figure 23.

It is clear that the convex hull has succeeded where the sliding mean method had failed. There is still a pathological case however, when the outlier is inside "Region 1". So the convex hull is obviously a better solution (better outlier rejection), but is still prone to outliers.

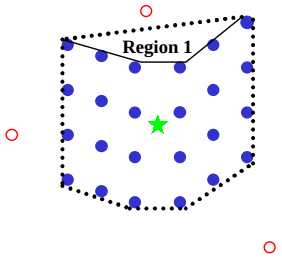


Figure 23: Convex Hull

The difficulty with implementing a convex hull algorithm lies in the fact that you do not strip off outliers. Instead, all N point permutations would have to be considered in calculating each convex hull, then finding the points that contributed to the minimum area. It was deemed that the extra effort (and significantly reduced calibration speed) was not warranted *at this time*. This is because the problem can be remedies simply by holding up a black circular piece of Perspex behind the target which will look similar to the target. In this case pathological outliers should not be a

problem. Also, underwater results may show that using the perspex works adequately, and the convex hull method may not even be necessary.

5.3 Line Intersection

Line intersections are extrapolated once lines have been fitted to the edgel chains (after filtering via the Sliding Mean method). The line intersection part applies the formula for a straight line, $y = mx + b$, to find the intersection point. The more complex part of the algorithm is the line intersection control loop. It is responsible for figuring out exactly which lines to intersect. The pseudo-code in Code Fragment 1 details the algorithm behind the line intersection control loop. It uses a method of gradients; hence there is the possibility of infinite or badly scaled gradients. However after running in the order of 300 trials, this event never occurred. For completeness, support for this case would be desirable, but since the values are double precision floating point, the probability of the denominator (x offset) being 0 is negligible for developmental purposes.

This section of code assumes the presence of an intersection routine. It merely uses the equation of a straight line and simultaneous equations to find the intersection point. Vertical lines may cause problems with infinite gradients, but no problem has been encountered yet. If this does turn out to be a problem, perhaps a quaternion notation may provide a solution.

```
For each line that has been fitted (reflist)
        Push the line into a new duplicate list (duplist)
For each item in reflist
        For each item in duplist
                If reflist item is not the same element as in duplist (i.e. the same line)
                        If the lines have endpoints sufficiently close (threshold)
                                Intersect the two lines and return the coordinate
                        If a coordinate was found (i.e. there was a match within threshold)
                                Push the corner point into a new list of points
                        End
                End
        End
End
Return the point list
```

Code Fragment 1: Line Intersection Control Loop

5.4 Point Sorting

In using Tsai's algorithm, the order of world and image coordinates must be the same; hence there is one further stage of processing before Tsai's algorithm can be applied. This is the sorting of image points to match the order in which world points are entered. World coordinates of the calibration target are hard coded however, so you can select any ordering scheme you like.

The difficulty is that in developing a sorting algorithm, the starting point is needed. So for instance, the top left corner could be selected as the starting position, and sorting will continue left-to-right, top-to-bottom. The problem of locating the top left corner had been assumed to be trivial, however implementation was quite a painful process due to top left corner location schemes did not work under all target orientations. It was decided that in order to advance development speed, a "trial and error" approach was used, where points were sorted in a variety of ways, then the results were later

assessed. The calibration target is to be held approximately horizontal. It is assumed that it will be held at a maximum of ±30° from horizontal.

The three methods for finding the top left point will be discussed after the general algorithm has been described. Point sorting can be summarised by Code Fragment 2.

```
Set all ID's to –1
        Initialize an empty list for storing the sorted points
        Set pointcounter to 0
        set closest to INIT
        While the sorted point list length is smaller than required
                Designate a refpoint as the result of attempting to find the top left point
                Mark this point
                        (set ID to pointcounter, remove from the list, increment pointcounter)
                While closest not equal to NOTFOUND
                        Push refpoint into the sorted list
                        Find closest point to refpoint (exclude self) within some angle range
                        (such that we trace horizontally, not vertically)
                        If a close point was found
                                Mark this point (as before)
                                Set this point as the new refpoint
                        End
                End
        End
```

Code Fragment 2: Point Sorting

5.4.1 Point Sorting Strategies

Since the orientation is not always the same, hence there were three conditions to be tested for when finding the top left point. They are shown in Table 7. The MAXTHETA method is the most reliable in general, and has been used in various calibration experiments.

Strategy	***Scheme Label***	***Description***
1	MAXTHETA	Find the maximum angle between every point and the origin (bottom left corner (0,0))
2	TOPLEFTBOTTOMRIGHT	Assume the target is at an angle, which places the left side facing the top left corner
3	BOTTOMLEFTTOPRIGHT	Assume the target is at an angle, which places the left side facing the bottom left corner

Table 7: Top Left Location Strategies

The schemes do not work every time, and each one has its downfall. Assuming that the target is at an angle (ie. method 2 and 3) works only if the point locations are arranged as expected (top-left-to-bottom-right etc). Also, method 1 works well in a number of cases, but has its disadvantages also. The problem with method 1 is illustrated in Figure 24. The red points indicate error. Of course, there are four points per box to sort - this diagram is only used as in indication of error.

Since all methods have their downfall, a method of trial and error can be used. One of the methods will always yield a correct answer under the stated assumptions. The points are sorted 3 times, and Tsai's algorithm is applied. The results are then compared to ground truth – results obtained when it is known that the correct sorting and outlier rejection was performed.

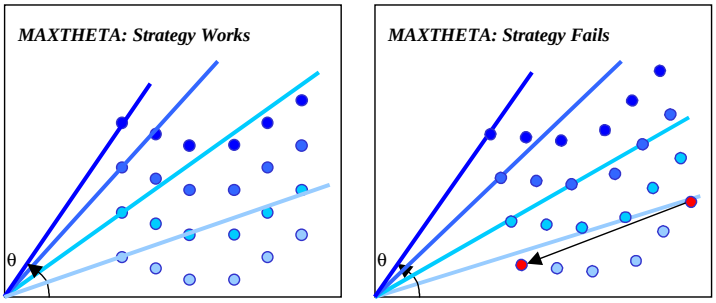


Figure 24: Failure of Strategy 1

There is certainly scope for improvement in the point sorting section of the program, but results have generally been acceptable. In the worst case, it could be assumed that the target is always at an angle, under the top-left-bottom-to-right assumption. Then it would work every time.

Point sorting success can be seen in Figure 25. Point ID's have been incrementally added to each point as they are sorted. The colour of the point is an indication of its ID (the "mod" or "%" operator has been used). Red replaces yellow at the top left corner, otherwise the order (left to right, top to bottom) is ***yellow, green, blue, cyan, magenta***. The colours then cycle through this list in order, as the point ID increases.

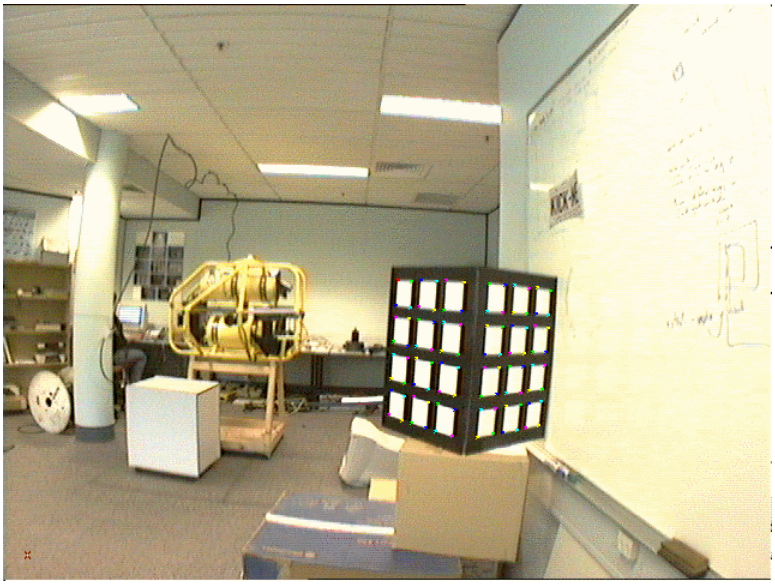


Figure 25: Sorted Points

Having an algorithm for point sorting, the input data is ready to be entered into Tsai's algorithm, and results obtained.

5.5 Applying Tsai's Algorithm

Tsai's algorithm can then be applied, using the package. The result is then compared to ground truth to see if the results yield nonsense values for focal length, or possibly the centre of radial distortion. Otherwise the calibration parameters are assumed to be correct.

Results have been gathered for a variety of circumstances. There are two main levels of calibration. The simpler is single camera calibration, in which the parameters for that camera are derived and returned. The second is system calibration, where all three cameras are calibrated based on the images they acquired at, or as close to, a certain instant in time. In this way, the translation vectors and rotation matrices returned by Tsai's algorithm can be used to find the relative camera positions.

The experimental process used was to observe the calibration target from a number of different positions. Intrinsic parameters for individual cameras are generated frequently. It takes a minute or two for full system calibration – where the *relative* extrinsic parameters are also catered for. This longer time for system calibration is largely due to complications with the line fitting and line intersection routines. These complications are:

- Line Fitting does not always return consistent line lengths. On three sides of the box, it fits an entire line, but on the other edge, the best it can do is a 3-pixel line fit.
- Because of the strange nature of the line fitting result, line intersections cannot intersect between the line endpoint's that have the closest Euclidean distance (this will often return the wrong point).
- To cater for this, a thresholding method has to be used, which may sometimes make the wrong decision. If the target is greater than 5m away, points that should be outside this threshold are now closer.

Also, to perform camera calibration, 7 or more calibration points are required in order to produce a non-trivial solution. [10] Therefore in theory, all of the points on the calibration pattern need not be located, so long as there are at least 7. The complication is the points have to be ordered to correspond to the world coordinate list. This requires some knowledge of calibration target topology, which is difficult to model, and has been deemed outside the scope of this project. With such a mechanism, calibration failure as captured in Figure 26, may be avoided.

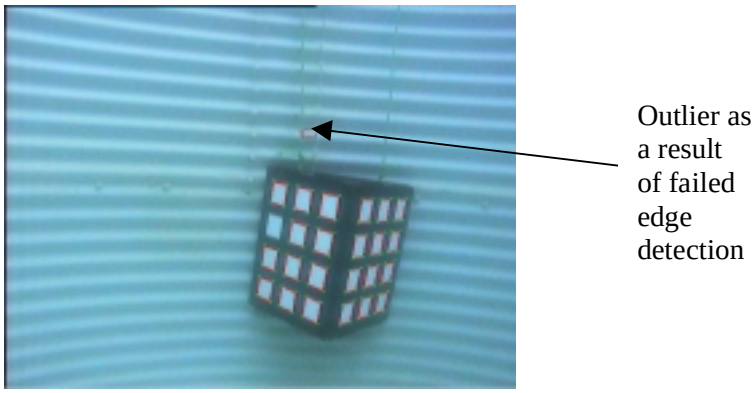


Figure 26: Calibration Failure

5.5.1 Test Images

Video of the target as viewed underwater and in the laboratory, has been collected. Two sets of calibrated images are shown in Figure 27 and Figure 28. Note that the underwater test environment limited us to only two video inputs. This is because we only had access to composite video feeds with only two composite inputs, and also two S-Video inputs (which were not useful in this case). Line fitting, corner detection and point sorting results are displayed in both image sets.

Also, lens distortion is evident in the centre image. This is due to imperfections in the perspex dome and will contribute to errors in calibration parameters for the pan-tilt camera. A result of which will be incorrect focal length and lens distortion parameters, and hence extrinsic parameters. This will not affect depth calculations, but will affect how accurately the pan tilt camera can be directed at the target.

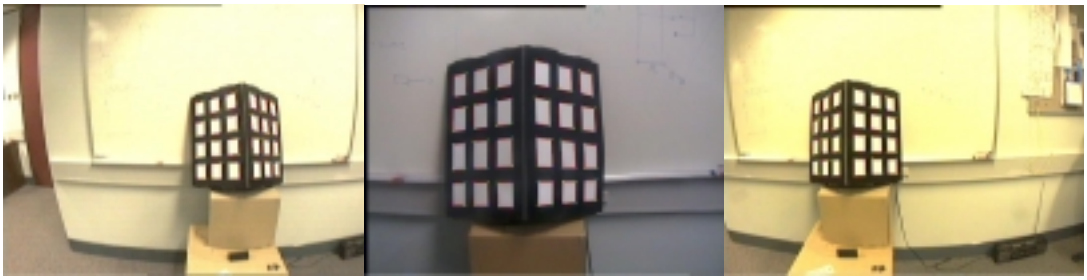


Figure 27: Laboratory Calibration Image Set

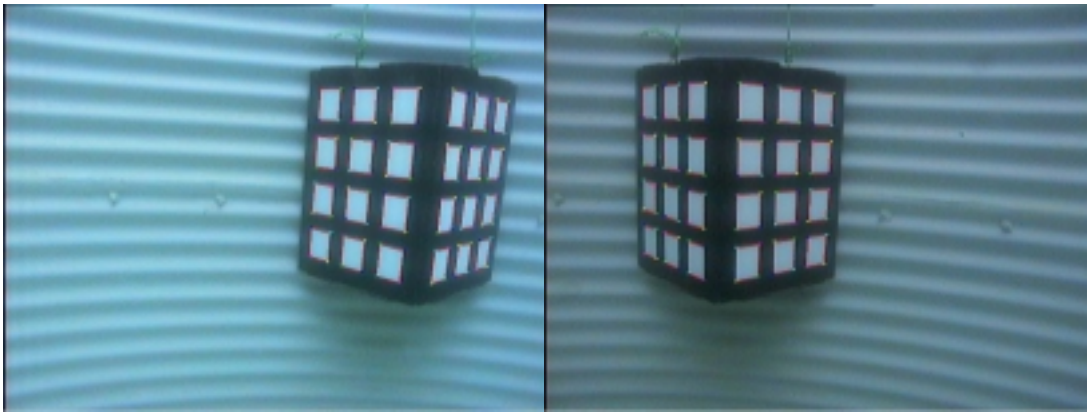


Figure 28: Underwater Calibration Image Set

Frames from the recorded video have been digitised and used during the calibration process. In the results data set, between seven and twelve calibrations have been performed on each camera, with approximately 5 full calibration successes. This has allowed computation of intrinsic parameters (and one extrinsic parameter) under varying situations to a specified level of confidence.

5.5.2 Intrinsic Parameters

Intrinsic parameters were calibrated in the pool, and in the laboratory. The results were calculated to a confidence of 95%. Results from calibration tests in the laboratory are tabulated in Table 8. Percentage error is indicated also which is calculated as *2* tolerance/mean * 100%*. First the results for calibration in the laboratory. These can be directly compared to the Pulnix camera parameters given in Appendix B.

Camera 0 (Left)	*Focal Length (mm)*	*Distortion ($1/mm^2$)*	*Dist.Centre X (pixels)*	*Dist.Centre Y (pixels)*	*Scale Factor (s_x)*
Mean	2.846366	0.037709	337.796844	240.503407	1.030092
Tolerance (±)	0.0152	0.0026	6.2077	1.0110	0.0011
Percentage Error	1.069%	13.536%	3.675%	0.841%	0.212%

Camera 1 (Centre)	*Focal Length (mm)*	*Distortion ($1/mm^2$)*	*Dist.Centre X (pixels)*	*Dist.Centre Y (pixels)*	*Scale Factor (s_x)*
Mean	6.258867	0.019131	329.743668	255.606562	1.293961
Tolerance (±)	0.474978	0.006799	16.558347	15.988919	0.013251
Percentage Error	15.178%	71.075%	10.043%	12.511%	2.048%

Camera 2 (Right)	*Focal Length (mm)*	*Distortion ($1/mm^2$)*	*Dist.Centre X (pixels)*	*Dist.Centre Y (pixels)*	*Scale Factor (s_x)*
Mean	2.882766	0.035011	292.139091	232.177868	1.029990
Tolerance (±)	0.014699	0.002773	5.650331	2.972539	0.001907
Percentage Error	1.020%	15.843%	3.868%	2.561%	0.370%

Table 8: Intrinsic Camera Parameters (Laboratory)

The only ground truth we have for comparison is the camera and lens specifications given in Table 12 in Appendix B. This verifies that 2.8mm is an accurate result – unfortunately tolerances are unknown. For the Sony camera, the values vary due to dome distortion, but also seem reasonable. Parameters exhibited by camera 1 display significant error due to abnormal distortion due to the perspex dome. Parameters calculated for camera 0 and 2, show significantly better results, but further minimising error in focal length and lens distortion would improve the performance of any position estimation routines. Calibration of the scale factor is also important. “Even a one-percent difference can cause three- to five-pixels error for a full resolution frame”. [3] So calculation of this parameter has been performed to a satisfactory level. Intrinsic parameters obtained from underwater calibration are tabulated in Table 9.

Camera 0 (Left)	*Focal Length (mm)*	*Distortion ($1/mm^2$)*	*Dist.Centre X (pixels)*	*Dist.Centre Y (pixels)*	*Scale Factor (s_x)*
Mean	3.808516	0.007852	340.722355	241.853594	1.031134
Tolerance (±)	0.046373	0.002017	5.182598	3.204416	0.000928
Percentage Error	2.435%	51.378%	3.042%	2.650%	0.180%

Camera 2 (Right)	*Focal Length (mm)*	*Distortion ($1/mm^2$)*	*Dist.Centre X (pixels)*	*Dist.Centre Y (pixels)*	*Scale Factor (s_x)*
Mean	3.814817	0.007799	300.620906	233.137488	1.030247
Tolerance (±)	0.024588	0.001565	7.697915	4.135454	0.001279
Percentage Error	1.289%	40.127%	5.121%	3.548%	0.248%

Table 9: Intrinsic Camera Parameters (Underwater)

The comparative results between parameters calibrated in wet and dry conditions can be seen graphically in Figure 29. The centres of distortion were consistent between the different conditions, note however the offset centre of distortion on camera 2. This lens property would yield especially strange position estimation results if lens distortion were not incorporated into the model. Focal lengths varied by approximately 30% when calibrated underwater. This is expected and is due to refractive effects. [1]

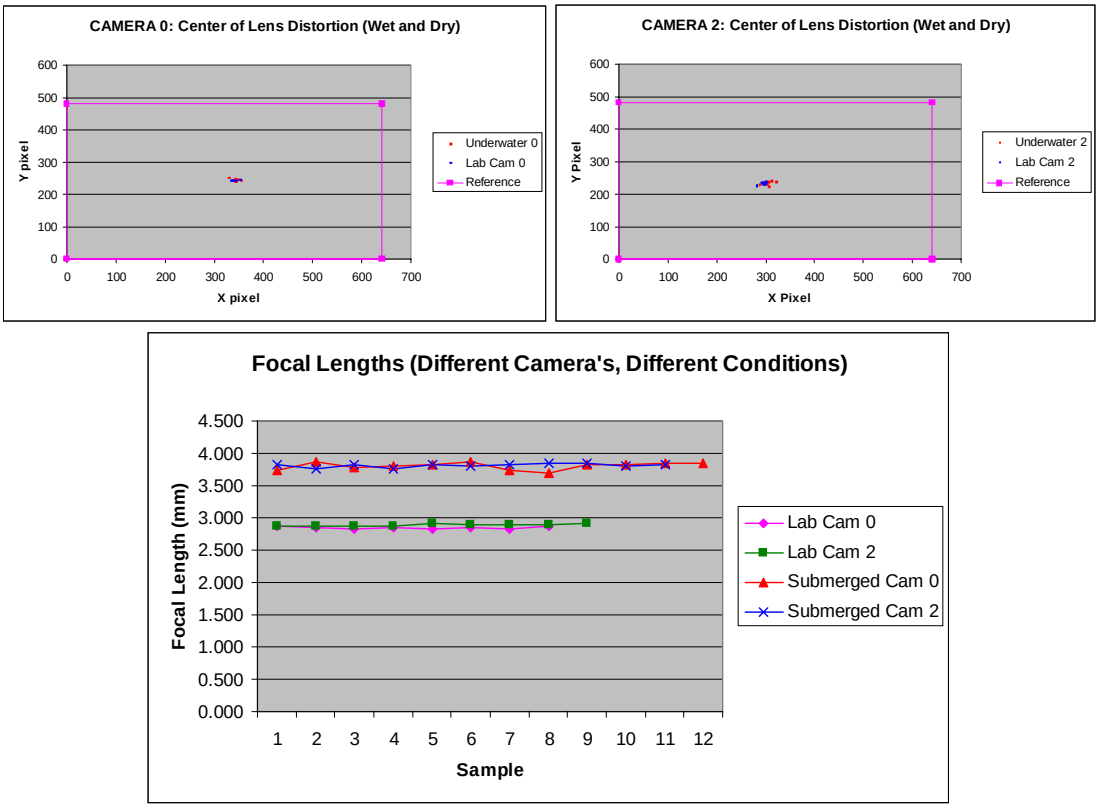


Figure 29: (Top left) Camera 0 Centre of Lens Distortion, (Top Right) Camera 2 Centre of Lens Distortion, (Bottom) Camera 0 and 2 Focal lengths

Lens distortion values were also noted to vary considerably between different environments. Again refractive effects on the water-glass-air transition is the cause of this difference. Note that in an underwater environment the effect of radial distortion significantly *decreases*. This has implications when it comes to tracking – the template will not vary much in appearance between the two images (compared to dry results). Therefore development in a lab environment will actually ensure that the algorithm will work underwater.

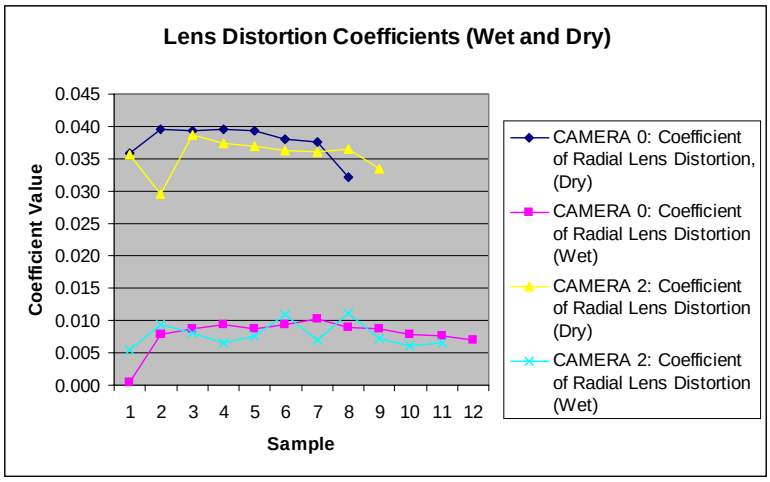


Figure 30: Coefficient of Radial Lens Distortion in different Environments

5.5.3 Extrinsic Parameters

System calibration has been tested in a laboratory environment. In general, results such as the translation vectors have appeared to be consistent. Unfortunately it is difficult to compare extrinsic parameters which have been obtained from different positions and orientations. Under these circumstances there are no parameters that will bear resemblance. The only calculation that can indicate that the figures are close, is the calculated baseline. Thus baseline calculations have been performed on camera 0 and 2 (left and right). The baseline is calculated in millimetres, and the results are calculated for the 5 cases below:

Environment	*Attempt 1*	*Attempt 2*	*Attempt 3*	*Attempt 4*	*Attempt 5*	***Mean***
Laboratory	412.68575	426.85729	426.85729	423.06713	486.92141	**435.2777**
Underwater	476.83541	562.03297	469.20370	443.91612	431.39693	**476.6770**

Table 10: Baseline Derived from Extrinsic Parameters (mm)

The baseline was measured with a tape measure to be 450mm. Thus we observe a 5% undershoot in the laboratory, and 6% overshoot underwater.

5.6 Camera Calibration: Conclusion

Camera calibration has been performed successfully, allowing the determination of intrinsic and extrinsic parameters. The cameras have been characterised in the laboratory and underwater environments, with the result that the focal length is increased by 1mm underwater, while the distortion coefficient has been largely reduced.

When compared to lens specifications and assumptions about centre of lens distortion, it seems that the calibration intrinsic parameters are reasonable. The expected focal length is equal to that given in the specifications but to more significant figures. The centre of lens distortion for camera 0 is approximately in the middle of the lens, while for camera 2, it if offset towards pixel 300, rather than 320 in the X direction. Extrinsic parameters such as translation vectors all seem to be approximately correct when observed by sight.

One significant result is that baseline calculations yield at least 5% error to the real value. This can be attributed to error propagation from the intrinsic camera parameter estimation, and possible inaccuracies in corner detection. This will be significant in position estimation results.

Chapter 6
Visual Correspondence

Humans are able to solve the problem of visual correspondence merely by "inspection". However, computers find it significantly more difficult to determine visual relationships. This is since computation is heavily based on correlation/convolution like algorithms, which are inherently slow.

Visual correspondence itself, is concerned with the location a template (a sub-region of an image) within another image. However, this other image does not necessarily view the object at the same orientation or from the same position. The centre of the template is used as the reference point, and the correlation peak indicates where the template centre is most likely to be. The correlation peak can be a maxima or minima, depending on the correlation scheme.

The next level of complication comes with tracking. Tracking can be defined as the task of computing the visual correspondence between N images in real time. The difficulty with tracking is achieving real time speeds, while maintaining accuracy.

A major issue that needs addressing is image noise. Pixel jitter (fluctuations in image acquisition due to hardware timing errors) is one source of noise. In an underwater environment, there is likely to be debris suspended in the water, which may scatter light. In order to reduce the effect of noise, Gaussian filters can be applied to the image, which has a blurring effect.

This years focus has been on integrating the system and using automated calibration parameters to generate position estimations. This requires a tracking algorithm so depth perception can be achieved, but places minimal constraints on the tracking efficiency. Thus the detail in which visual correspondence has been examined, will be minimal. Template updating was ignored, but a number of different routines have been implemented in an attempt to improve localisation and correlation certainty. These algorithms and a number of simple optimisations for real-time tracking will be explored.

6.1 Considerations

The problem to be addressed is "how do we evaluate the level of visual correspondence between two images?" There is the problem of optimising these algorithms to work in real time, however efficient optimisations have been omitted from this year’s work in attempt to limit the project scope. Image correspondence algorithms will be the major focus.

The vision system implements a number of methods for computing visual correspondence. The techniques can be split up into two major groups:

1. Correspondence without Image Transformation
2. Correspondence with prior Image Transformation

In this context, image transformation is more than Gaussian blurring alone. It involves either mapping the image to a new "image space", or performing edge detection. First, implementation of correspondence without transformations will be investigated. Then some attempts will be made to improve these results using image transforms. First, the background theory will be explained.

6.2 Correlation

Correlation is the simplest way to determine image correspondence, has been used in all of the implemented algorithms. Another way of computing correspondence is via frequency domain analysis. This has not been attempted this year, due to time restrictions. So we will focus on correlation.

Correlation is used to locate the centre of the template in an image. When applied to an image, the correlation operator produces a correlation map, a two dimensional array of individual numbers. The process of producing a correlation map can be outlined in Figure 31.

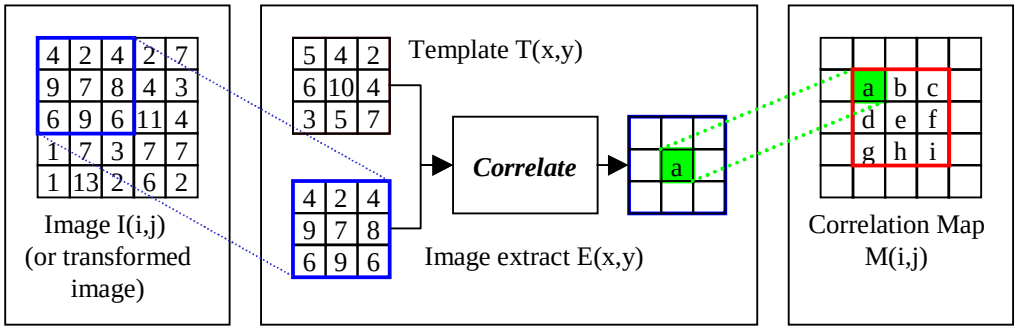


Figure 31: Correlation (Grey scale)

Here the blue box is translated over the image, *I(i,j)*, moving from left to right, then top to bottom (other implementations are possible). For each position, the numbers inside the box (pixel intensities or otherwise) are extracted (into *E(x,y)*) and compared to those in the template, (*T(x,y)*). Assuming an odd template width and height, we can compare *T* and *E* using the correlator, thus producing a number which represents the relative level of correspondence between the images. As the blue box is moved over other image regions, the same algorithm is applied, thus producing the correlation map. Simple adjustments (eg. ignoring some of the image and template pixels) can be used to apply this simple algorithm to non-odd template dimensions. This mechanism is in some way for proceeding visual correspondence techniques; the only difference is in the implementation of the box that performs correlation.

6.3 Correlation Implementation

There were two correlation methods that were implemented this year. These were Normalised Correlation and Sum of Absolute Differences. Normalised Correlation uses Equation (2) to find each value in the correlation map, *R(u,v)*, for each image coordinate.

$$R(u,v) = \frac{\sum_{y=-j}^{j}\sum_{x=-i}^{i} I(u+x,v+y).T(i+x,\, j+y)}{\sqrt{\sum_{y=-j}^{j}\sum_{x=-i}^{i} I^2(u+x,v+y).\sum_{y=-j}^{j}\sum_{x=-i}^{i} T^2(i+x,\, j+y)}} \qquad \text{…….. (2)}$$

Where *I(i,j)* is the image and *T(i,j)* is the template being located. After applying this formula to every pixel in *I*, the correlation map will contain values ranging between zero and one. A one indicates a perfect match, while zero corresponds to no match. Thus maximisation of *R(u,v)* yields the template centre location. Of course, this value can be mapped to range from 0-255 (to generate a grey scale correlation map) or inverted such that 0 is the maximum location.

Simpler than the method of normalised correlation, sum of absolute differences uses equation (3) to find *R(u,v)*.

$$R(u,v) = \sum_{y=-j}^{j}\sum_{x=-i}^{i} \left| I(u+x,v+y) - W(i+x,\, j+y) \right| \qquad \text{…….. (3)}$$

According to this particular implementation of the algorithm, the centre of the template is most likely to be where *R(u,v)* is minimised. This method is both simpler and faster than normalised correlation, due to the removal of extra square root and division operators, but produces a relatively unpredictable range of *R(u,v)* values - one correlation might return a peak at 40,000, and then next at 32,000. This makes automatic thresholding adjustment for template removal, [1] more difficult.

6.4 Visualisation

In order to gain an empirical understanding of the effectiveness of each tracking routine, a correlation map generator for matlab was generated. This produced a three dimensional plot of the correlation level at each (x, y) image coordinate. The test image shown in Figure 32 was obtained. This is a useful test image because there are similar objects lying next to each other. The template that will be used in each of the tests is  (here the template is smoothed). Also note that this template is altered if an image transformation technique is applied. The corresponding image location is highlighted in red.

Figure 32: Raw Image

The matlab correlation map between the template and image shown above is given in Figure 33. The method used in this correlation was the sum of absolute differences. The red peak indicates the maximum correlation value, and hence the templates position.

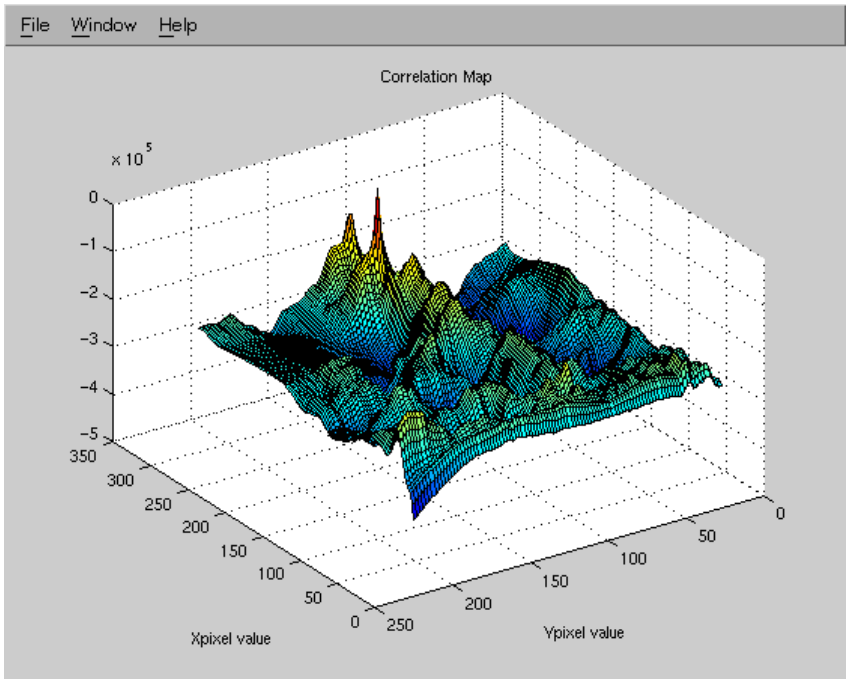


Figure 33: Matlab Correlation Map

This correlation map does provide a lot of detail, however it takes a considerable amount of time to generate the map to this resolution (2 minutes per test). Instead, an option was added to transform the above correlation map into a Grey-scale image. These will be examined from here on.

6.4.1 No Image Transformation

Without image transformation, the correlation techniques described will compute the correlation map. The problem with this comparison is that the correlation peak is very sharp, giving rise to good localisation, but also is intolerant to template variations. For example a fish that turns to swim in another direction will appear different. One way to reduce the effect of template variations is to apply a Gaussian operator. Much like correlation, the Gaussian kernel is convolved with the image to achieve a blurring effect. [1].

Grey-scale correlation maps of the observed scene were obtained after the application of a Gaussian operator. The results can be seen in Figure 34 for sum of absolute differences and Figure 35 for normalised correlation.

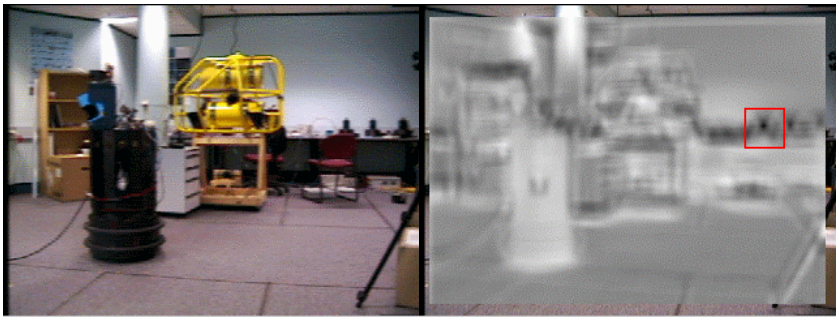


Figure 34: Correlation map for sum of absolute differences

Figure 35: Correlation map for normalised correlation

Note that the normalised correlation map intensities were cubed in order to enhance the visibility of the maximum correlation point. These methods are effective in locating the template in a simple manner, but are affected by background colours. This situation is outlined simply in Figure 36.

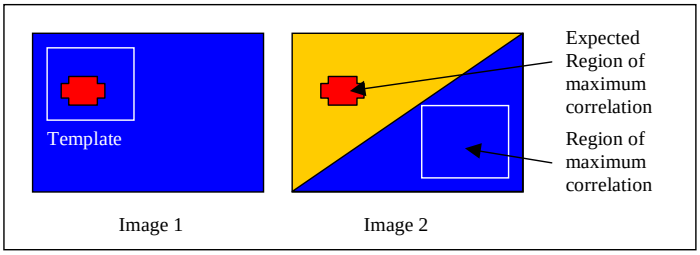


Figure 36: Background Dominance

The template is selected in Image 1, and the object of interest is indicated in red. However, the blue background dominates the template, it accounts for over 50% of the template colour. If correspondence is then calculated between this template and an image obtained in an environment such as that shown in Image 2, poor results will be obtained. There are a number of methods that can be used to get around this problem, and in general, it requires the image to be transformed in some way.

6.4.2 With Image Transformation

There have been two image transformation mechanisms implemented in the vision system. These involve correlations using images transformed via the sign of difference of Gaussian operator, and using the rank transform. Both methods result in a situation similar to that shown in Figure 37. Here background colour transitions are the only information correlated.

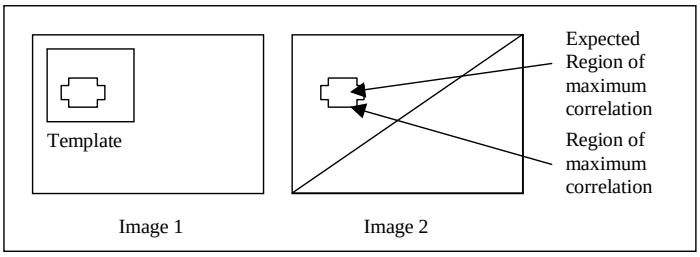


Figure 37: Removal of Background Dominance

This can be done by taking the image derivative [1], or by another means. The sign of difference of Gaussian method involves convolving the image with two Gaussian operators with different variance. Subtracting the two convolutions is a simple way to detect the image edges. Note that in this implementation, the sign of the difference of Gaussians indicates whether the pixel is mapped to black or white, hence the method is called sign of difference of Gaussians.

Another method for performing a similar type of transformation is via the Rank transform. It looks at relative pixel differences rather than edge location when performing the image transformation. The method can be illustrated in Figure 38.

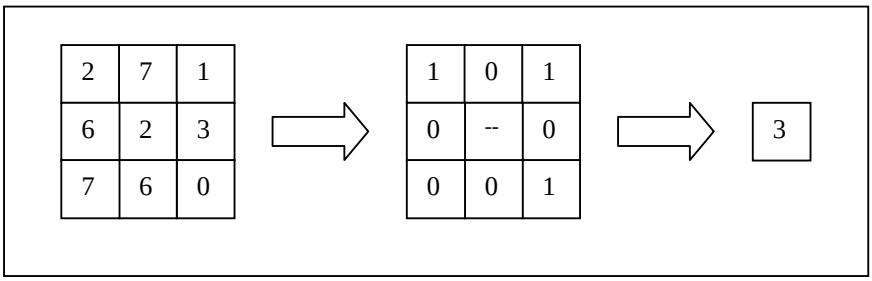


Figure 38: Rank Transform

This method performs a mapping in a similar way to correlation. It looks at the surrounding pixels, decides which are less than or equal to the current pixel (centre), then counts how many pixels had a lower intensity. This count of lower intensities is the new pixel value. This method is advantageous because it is tolerant to fluctuations in pixel values because it uses relative pixel ordering rather than absolute. The method can be stated succinctly in Equation (4), (5) [7].

$$R(u,v) = \sum \xi(I,P,P') \quad \text{..... (4)}$$

$$\xi(I,P,P') = \begin{cases} 1; I(P) < I(P') \\ 0; otherwise \end{cases} \quad \text{..... (5)}$$

Here I is the image extract, P is the centre pixel and P' represents the surrounding pixels. After applying the rank transform to all pixel locations, a transformed image results, from which a correlation map can be generated. The only drawback is that there are a smaller number (8 in this case) of new intensity values. This can be increased by using a larger grid size; instead of 3x3, use a 5x5 grid.

Results for various correlation tests can be seen below. Figure 38 shows the Rank transform (using a 3x3 grid), with normalised correlation applied to locate the template (also Rank transformed). Figure 40 shows the result of the sign of difference of Gaussians, coupled with normalised correlation and Figure 41 shows sign of difference of Gaussians, coupled with sum of absolute differences.

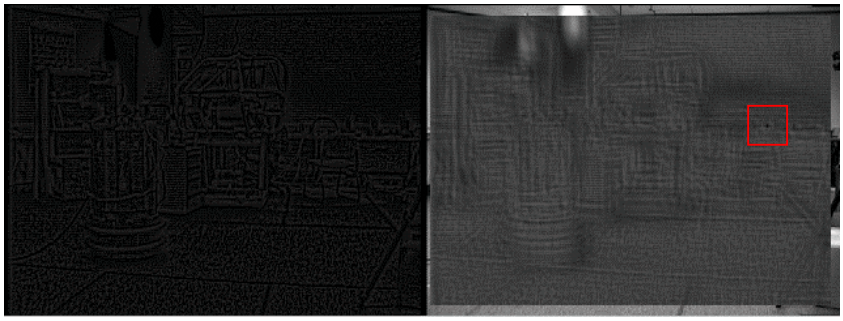


Figure 39: (Left) Transformed Image, (Right) Correlation map

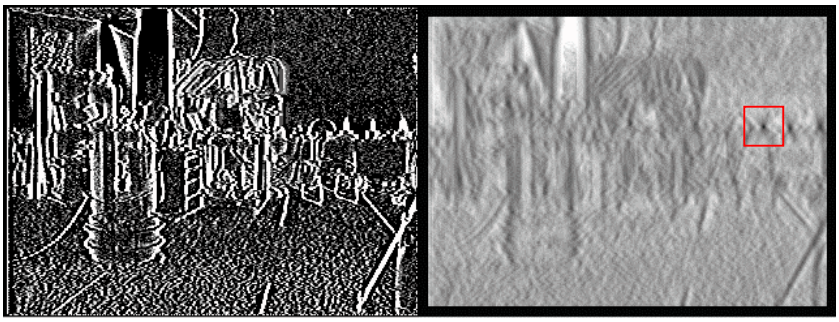


Figure 40: Sign of Difference of Gaussians, with Normalised Correlation

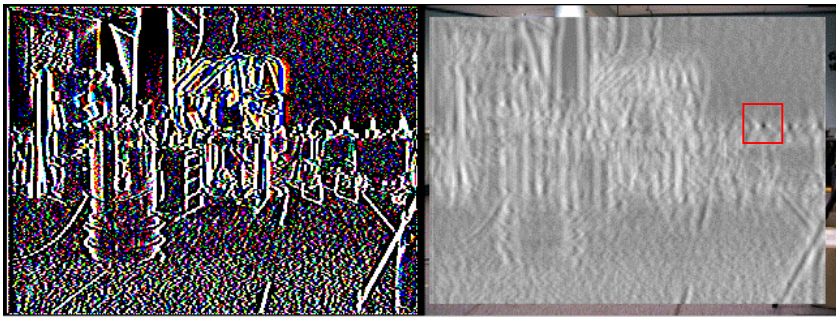


Figure 41: Sign of Difference of Gaussians, with Sum of Absolute Differences

These empirical results show a significant improvement in intensity distribution throughout the correlation map, with the exception of the target point, which is still well defined. For localisation purposes, these techniques would seem to offer significant advantages since they reduce the radius of the correlation peak. More importantly, these methods reject bad matches far better than without prior transformation. There were also a number of optimisation schemes implemented. These are detailed in Appendix J.

6.5 Timing Results

While the implemented optimisations are simple, they have allowed tracking to be useful at a frame-rate of up to 20Hz. While this is still slower than the benchmark of 30Hz, many further optimisations should still be possible. Note also, that this refers to the time taken to execute the correlation algorithm only. To give a relative idea on the speed of various algorithms, their respective execution time can be measured and compared. Table 11 shows these relative speeds. (Using a template that is 14x18 pixels in size, operating on a 320x240 image). Here the following abbreviations have been used: "sum of absolute differences" (SOAD), "normalised correlation" (NORC) and "sign of difference of Gaussian's" (SDOG).

Method	***Trial 1 (sec)***	***Trial 2 (sec)***	***Trial 3 (sec)***	***Mean***
Without Transformations				
SOAD	6.97	7.20	7.14	7.10s, 0.14Hz
SOAD-Window	0.11	0.11	0.11	0.11s, 9.09Hz
SOAD Window Sparse	0.05	0.06	0.05	0.05s, 20.0Hz
SOAD Epipole	1.93	1.87	1.98	1.93s, 0.52Hz
SOAD Epipole Sparse	0.16	0.17	0.17	0.17s, 5.88Hz
NORC	*4.50*	*4.45*	*4.50*	*4.48s, 0.22Hz*
With Transformations				
SDOG + NORC	*6.21*	*6.48*	*6.32*	*6.34s, 0.16Hz*
Rank + NORC	*4.95*	*4.94*	*4.83*	*4.91s, 0.20Hz*

Table 11: Relative Correltion Speeds

Here italics indicates correlations performed in grey scale (normalised correlation is simpler when grey scale is used). Hence grey scale correlation cuts the correlation time in three. This is why the results for normalised correlation are quicker than for sum of absolute differences, which uses **24-bit colour**. Results are divided according to whether they involve transformations or not. In every case, transformations take considerably longer, however no windowing technique was implemented for these methods, hence it can be expected that improvements to a similar scale to SOAD Window Sparse could be achieved. It seems clear that windowing techniques offer the best results. However the weakness in method is that for a fast moving target, this assumption that the target is moving slowly relative to the frame rate, may not always be valid. Hence a position predictor module which intelligently estimates the next target location based on AUV state, may be of use.

6.6 Visual Correspondence: Conclusion

Various methods for calculating the region of maximum correspondence have been investigated. Observations have shown that better localisation can be achieved through the use of image transformations prior to correlation, however this significantly slows down the frame rate. Depending on the level of optimisation used, these methods may still be appropriate – sparse windowing techniques show promise, as do epipolar correlations in performing the secondary stereo correlation.

Normalised correlation provides slightly better localisation at the cost of losing colour information and increasing computational loads. For raw speed, the sum of absolute differences technique yields satisfactory results and does not remove colour information, while achieving acceptable efficiency. For a template size of 14x18 pixels, tracking speed ranging from 0.14 to 20Hz can be achieved within a 320x240 image, while only using simple optimisation schemes.

Chapter 7 Position Estimation

Position estimation is another of the major objectives of the vision system. The other major objective is to achieve tracking at frame-rate. Position estimation is tied closely to both camera calibration and tracking. Camera calibration informs the system of the required system parameters, and tracking offers a solution to the problem of visual correspondence between left and right images in real-time. Once a template has been located[5] in both images, position estimation can continue.

The task of position estimation can be defined as determining a vector from a known point in space, to the object of interest. In this case, the object of interest is the target that is being tracked, and the known point in space is the position of the image plane in either (arbitrarily choose the left) camera.

7.1 Position Estimation Theory

It is widely known that depth perception is difficult to achieve with only a static mono-view camera system, unless you are observing an object of known dimension (eg. a calibration target). This section looks at implementing stereo vision with arbitrary but reasonable camera orientation, as found on Kambara.

Equations used for Tsai's camera model were introduced in Appendix G. They were specified to give an insight into how the world coordinates can be related to image coordinates. This is information that is required for the calibration process. In this section, we use Tsai's camera model to derive world coordinates from image coordinates, ie. work the other way, and introduce a method of incorporating data from another image to derive the unknown ***z*** component. This derivation is given in Appendix I. Its objective is to find a function that relates the observed pixel coordinates in image 0 and 2, to world (x,y,z) coordinates. This algorithm has been implemented in the “stereo” module.

$$\begin{aligned} x &= f(X_{f0}, Y_{f0}, X_{f2}, Y_{f2}) \\ y &= g(X_{f0}, Y_{f0}, X_{f2}, Y_{f2}) \qquad \text{.....} \quad \textbf{(6)} \\ z &= h(X_{f0}, Y_{f0}, X_{f2}, Y_{f2}) \end{aligned}$$

7.2 Results

A number of experiments were conducted during the development of the position estimation module. The first was to see if the tracking and position estimation would integrate successfully

[5] 'Located' meaning the correlation map has been analyzed, and conclusions drawn about template/object position.

and produce a sensible range of estimates. Since the tracking module was not speed optimised, care was taken to avoid template loss (moving a distinguishable target slowly over a number of boxes).

7.2.1 Three Dimensional Position

The position estimation algorithm was implemented and one of the first tests was to obtain a trace of an object position in three dimensions. The results were promising in that they looked to be within the 10% error range when verified roughly with a tape measure, however they exhibited some behaviours that were quite alarming – suggesting an implementation error. When viewed from above and side on (see Figure 42), it is evident that since the object was moved only at approximate right angles to the camera, that the Z estimation was not at all accurate for varying heights.

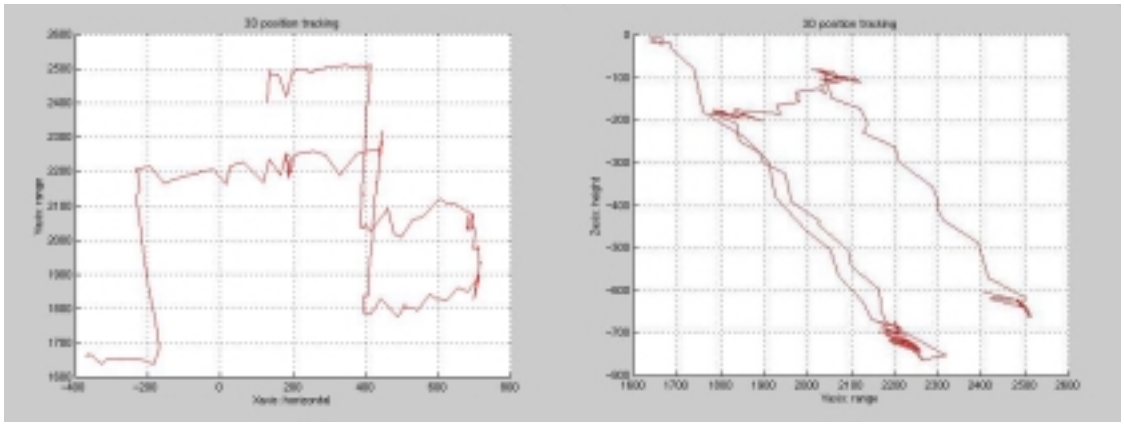


Figure 42: (Left) 3D top-view, (Right) 3D side-view (measurements in mm)

It was assumed that the reason for this angular depth estimate was due to the angle at which the cameras are mounted on Kambara. This is shown in Figure 43. The next step was proving this assumption.

Figure 43: Stereo camera mount angles

7.2.2 Vertical Characterisation

A test was performed with Kambara a fixed distance from a wall. The *z* position of a target was estimated at varying object altitude. The result of this preliminary test is indicated in red, in Figure 44.

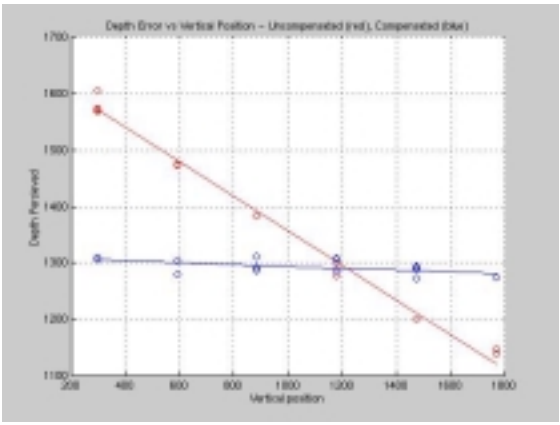


Figure 44: (Red) Depth versus Vertical position, (Blue) After compensation (mm)

The relationship between the depth and height could be approximated as linear. Fitting a line to this data allowed the calculation of the camera angle (angle of the master camera), and by applying a rotation matrix to the position estimate, the blue line in Figure 44 was obtained by performing the same test. Thus this depth error was certainly due to the angle at which the cameras were mounted. This result has two major implications:

- There is no problem if 3D position estimates are required, relative to the frame.
- Otherwise a rotation matrix has to be applied to the data, using a measured camera angle (relatively difficult to measure).

Either way, the next question is whether the estimates lie within the 10% error range that has been specified.

7.2.3 Range Estimation Results

An experiment was designed that would require location of 75 points in space, which covered the field of view of the stereo rig. This corresponds to an angle of ±26° at a distance of 1.18m. The experiment extremities can be seen in Figure 45[6].

Figure 45: Experiment extremities

[6] Note that these are extremities for the stereo rig. Ie. The images on the left-hand side of Figure 45 are from the *right* camera, and those on the right, are from the *left* camera.

An attempt was made to place the target in the corners to see the maximum effect of radial distortion. The red box indicates the target that was used for calculating the depth estimates.

There were three vertical height levels used - over which five depths (for five horizontal positions (total = 3x5x5 points) were measured. The position estimation module then made its calculations of the depth, and the results were compared. Figure 46 shows the range of depth values as the target moved in and out from one side of the image to the other. There are three important observations to be made:

- The position estimates are generally all just outside the 10% error bars.
- The estimates are consistently shorter than the measured data.
- The estimate errors are approximately consistent for varying heights and horizontal offsets.

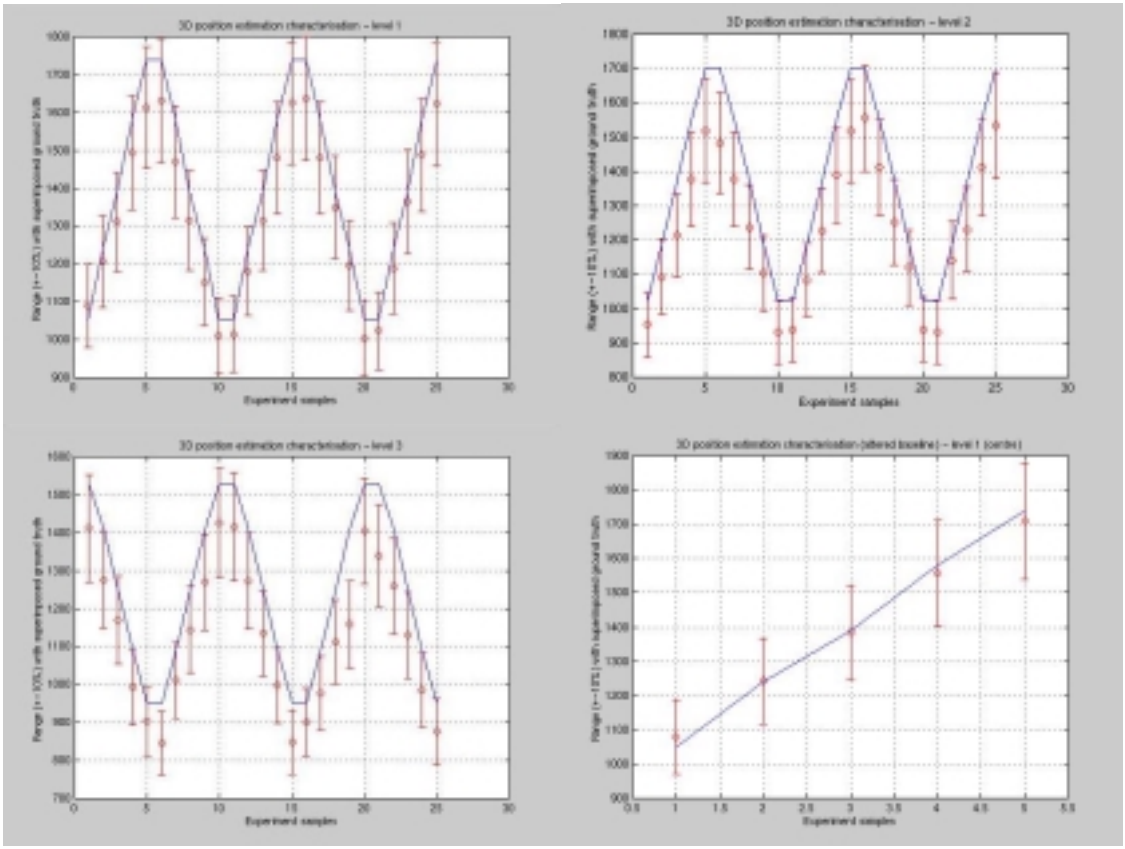


Figure 46: (Top Left) Top level depth estimates, (Top Right) Middle level depth estimates, (Bottom left) Lower level depth estimates, (Bottom Right) Compensated depth estimates

Also recall that extrinsic parameter calculations yielded baseline calculations to within ±5%. The baseline is directly proportional to target range; hence this error propagates to the range estimates. This is most easily shown in the case of parallel cameras (See Appendix E).

$$\frac{\text{baseline}}{\text{Range to object}} = \frac{\text{Pixel disparity} \times \text{pixel dimension}}{\text{focal length}} \quad \ldots\ldots\ \textbf{(7)}$$

Camera baseline is a simple parameter to measure, so the actual baseline was then hardcoded into the stereo module, producing the results in Figure 46 (Bottom right). These results appear accurate (mean error 1.33%, maximum error of 2.76%) to within 3%, rather than the required 10%. Also, since the estimate errors were approximately constant throughout the experiment, we can assume that this baseline adjustment will have a similar effect on all other depth estimates.

7.3 Pan-Tilt Orientation

The final step is to map the orientation of the pan-tilt head to view the object of interest. The algorithm is simple and involves the following steps:

1. Determine the relationship between the camera pan-step and 1°.
2. Map the position vector to be in the pan-tilt camera's reference frame via calibration extrinsic parameters.
3. Figure out the required pan and tilt angles.
4. Issue the command to the pan-tilt camera via the VISCA driver.

Having achieved pan-tilt object tracking, the project goals have all been met.

7.4 Position Estimation: Conclusion

A stereo camera rig, whose relative orientations have not been accurately measured, has been shown to be capable of estimating range to an accuracy of approximately 15%. If the camera baseline is measured separately, this error can be reduced to less than 5% (for estimations between 1 and 2 metres), and this error appears to be consistent over the entire field of view.

It is clear that position estimation that accounts for lens distortion appears to yield more accurate depth measurements than using parallel cameras and the pinhole camera model. Note that the approximate magnitude distance between the camera focal points is required in order to obtain this higher degree of accuracy. Using the measured baseline, the accuracy requirements for the position estimation module have been satisfied. At the same time, pan-tilt camera orientation was fixed on the target that was being tracked.

Chapter 8 Conclusion and Further Work

Implementation of image acquisition, camera control, various tracking routines and the position estimator module allowed 3D target following to be performed in real time, and hence the project goals were achieved. A video digitising device driver was successfully ported from Linux with to perform under a VxWorks environment, obtaining video data at up to 45Hz at a resolution of 320x240 pixels.

Laboratory tracking was successfully performed at 20Hz with minimal optimisations, with no template updating.

Calibration was performed in laboratory and in underwater conditions. Underwater calibration showed an increase in focal length by approximately 30%, and a significant reduction in lens distortion effects. Using the laboratory calibration results, position estimation was implemented. The extrinsic camera parameters were calculated via the calibrated intrinsic parameters, propagated the calibration error through to the extrinsic parameters. This extrinsic parameter error was evident by observing the baseline calculation, and noting the 5% error. Once the correct baseline was used, depth estimate error dropped from approximately 15% to under 3%. This easily satisfies the accuracy requirements for the position estimation module. This result also improves on the previous year's estimates, as currently 3% error is possible for full field of view tracking (±26° in the documented experiment), where lens distortion error had previously diverged estimate error past 10% once the field of view became larger than ±15°. Mapping of the pan-tilt head to this position vector was performed successfully – hence the full set of project goals was satisfied.

While the goals for this year have been satisfied, there is still scope for improvement. For instance, full system calibration currently takes approximately 2 minutes, based on experience, and there are some strange results from the line fitting routine. Once line fitting can be performed consistently with a more reasonable minimum line-fit length, calibration accuracy – ie. Baseline calculations may become closer to the expected value.

In 1998, template-updating routines were implemented to follow a target as it changed shape in its environment. Currently there is scope for the addition of a template updating mechanism in the vision system, but it has not been implemented. Further speed optimisations, perhaps by using logical (bit-wise) rather than arithmetic operations within the correlation functions would also be highly desirable to satisfy the minimum tracking requirement of 30Hz.

Underwater position estimate tests would also be desirable, in order to see if the scaled focal length and lens distortion parameters account fully for the air-glass-water refraction transition.

References

1. Reynolds.J. (1998). Autonomous Underwater Vehicle: Vision System. Department of Engineering Thesis, ANU, 1999.
2. Gaskett.C, Wettergreen.D, Zelinsky.A. (1998) Development of a Visually-Guided Autonomous Underwater Vehicle, Oceans 98.
3. Tsai.R.Y. (1987). A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras and Lenses, IEEE Journal of Robotics and Automation, VOL RA-3, No 4. August.
4. Ballantyne, Todd, Acuity Imaging, Personal Communication.
5. RS Industrial catalogue, 1998.
6. Correlation and Hough Transform, Commands and C Functions, http://www.bath.ac.uk/BUCS/Software/image_analysis/visilog/html/refguide/chap11.html#HDR2
7. Woodfill.J, Zabih.R. A Non-Parametric Approach to Visual Correspondence, IEEE, PAMI.
8. TargetJR Homepage: www.targetjr.org
9. Willson.R. Tsai calibration package online: http://www.cs.cmu.edu/afs/cs.cmu.edu/user/rgw/www/TsaiCode.html
10. Trucco.E, Verri.A. (1998). Introductory Techniques for 3-D Computer Vision. Prentice-Hall: New Jersey.
11. Abdallah.S. PhD thesis.
12. Press.W, Teukolsky.A, Vetterling.W, Flannery.B. (1996). Numerical Recipes in C. Cambridge University Press: Melbourne.
13. Rockwell Online: www.rockwell.com
14. Sony Online: www.sony.com

Appendices

Appendix A: Glossary

24-bit colour: An image format where the pixel intensities are stored in 3 channels. One byte is for the red intensity, one for the green and one for blue. When mixed, the RGB combination can produce all the colours in the spectrum (to a finite degree since we only have one byte per colour channel.

Baseline: The separation between camera optical axes

Disparity: The horizontal pixel difference in target locations between the two cameras.

Edgel Chain: A linked list of points that form an edge.

Epipolar Line: Represents the possible point locations of the target in one image, given the point location and relative camera orientation of the other image.

Focal Point (Focal length): The point at which rays at infinity are focused to once distorted by the camera lens. The focal length, is the length in metres from the image plane to the focal point, along the optical axis.

Optical Axis: Line normal to the camera lens, piercing the CCD at its centre (or centre of radial lens distortion).

Pixel Dimension: The size of the CCD sensor elements (given by the manufacturer usually).

Radial lens distortion: Imperfections in lenses occur when rays at infinity to deviate from intersecting the optical axis at the focal point.

Target Range (Range to object): The distance from the cameras to the object of interest (no horizontal or vertical distance components)

Video Fields: Usually two fields are used to convey video information. Half resolution video data is acquired by the two fields, then interlaced producing a full resolution image.

Appendix B: Camera Specifications

Camera	Pulnix TMC-73	Sony EVI-D30
Video outputs (NTSC)	S-Video (12 pin Hirose plug), Composite (RCA)	S-Video (4 Pin mini DIN) Composite (RCA)
Lens	CS Mount attachment included. Focal Length: 2.8mm	F = 5.4 to 64.8mm, F1.8 to F2.7, horiz angle 4.4° to 44.8°
CCD	1/3” colour CCD	1/3” colour CCD
Power Requirements	12V DC, 190mA	12 – 14 V DC, 850mA
Dimensions	40 × 85 mm	142 × 109 × 164 mm
Mass	120g	1200g
Additional Features		Pan tilt unit – range 100° horizontal, 25° vertical Visca control language

Table 12: Camera Specifications

Appendix C: Memory Mapping in VxWorks

Memory mapping is a technique whereby software writes to memory addresses (eg. pointers) which have been remapped to some other location. So it is possible to write directly to PCI register configuration space directly as you would write to a pointer in memory. The operating sets aside a certain amount of space that is to be mapped. The address of the first memory slot that has been allocated is referred to as the base address. Then, knowing the register offsets (given in device white paper), specific values can be written to the desired device register.

In some systems, memory mapping can be performed dynamically by the operating system. Unfortunately our implementation of VxWorks does not include this feature but instead, all memory mapping can be statically defined when building the VxWorks kernel. The base address, amount of space is defined as part of the 'sysPhysMemDesc' structure located in the file 'sysLib.c'. See Code Fragment 3.

However, the memory space is allocated in a certain way – so there is no freedom to set up just any address to map from. Operating system level PCI routines are required to locate the device instance on the PCI bus, from which the base-address can be recovered. Once this is done, the memory mapping entry can be made in the kernel, the system rebooted and device driver installed.

```
….
/* PXC200 Imagenation Frame Grabber 0 */
{
      (void *) 0xfecfd000,       /* Physical Address              */
      (void *) 0xfecfd000,       /* Virtual Address        */
      0x1000,                    /* 4 Kbytes (Register Space      */
      VM_STATE_MASK_VALID | VM_STATE_MASK_WRITABLE |
      VM_STATE_MASK_CACHEABLE,
      VM_STATE_VALID | VM_STATE_WRITABLE | VM_STATE_CACHEABLE_NOT
},
….
```

Code Fragment 3: Virtual Memory Definition

Appendix D: PXC200 Driver: Data Integrity Assurance

Interlace and MUX transitions are the two most evident issues to be addressed with respect to acquiring images, which were useful for image processing. There are two other problems that can occur, which are considerably more complicated.

The first problem is quite unusual, and can be attributed to logic timing within the digital circuitry. As mentioned above, images are transported via DMA from PXC200 to host using the RISC engine. The RISC engine is enabled on the rising edge of RISC_ENABLE (GPIO_DMA_CTL register). The BT848 RISC processor encounters trouble unless RISC_ENABLE is set within a window of time. [4] If not, a series of FBUS errors are triggered which result in only a partial image being retrieved. It is difficult to tune the program to be timed accurately with respect to a clock on-board the PXC200, thus another method was desirable. It was found that the probability of starting the RISC engine outside this window was low, but still high enough to warrant a solution.

The approach to this problem uses the philosophy of assuming that it will start successfully. If the driver detects a flood of FBUS errors however, we know that the RISC engine did not trigger at the appropriate time, and we can then stop and restart. Once the processor is going, there is usually little problem. The same course of action is taken in the event of a RISC jam – an event that occurred during development, but not since the code was stabilised. It serves as a watchdog to make sure if it *does* stop for no apparent reason; it will be restarted as soon as possible.

The second and most frustrating problem was image overlap. This is due to inconsistencies between the speed of PCI/DMA transfer, and 'memcpy' software routines. The solution is described in Figure 47.

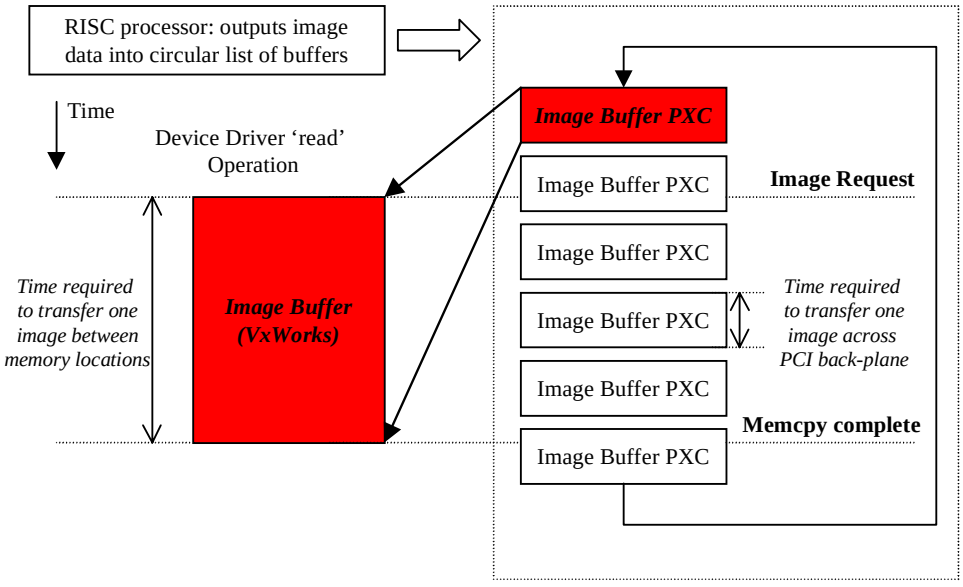


Figure 47: Image Overlap Solution

Performing DMA transfers to, and performing memcpy's from, one memory location, results in multiple DMA transfers occurring in the time it takes for memcpy image to be performed. Instead, the RISC programs were modified to perform DMA to a circular list of image buffers that allows the CPU (specific to its clock speed), to read an image without DMA overlap.

The idea is simple. An interrupt is triggered by the RISC engine on completion of an image transfer, then a counter is incremented (circularly). The counter indicates the buffer to which is currently being written. When a read is called from an application, the counter can be decremented by one, which will point to the most recently written image buffer. Obviously a knowledge of the image buffer locations is necessary – this information is stored when the RISC programs are written.

The only limitation is memory space. Enough image buffers are required such that the time it takes to copy one image from memory (using memcpy), is less than it takes for the PXC200 to send N-1 images into the circular list of buffers. N can be determined empirically, and works for all resolutions at 9. The smaller the resolution, the more buffers are required. Without this mechanism, partial images are the result. See Figure 10.

Appendix E: Parallel Camera Stereo Vision

With two cameras, we can implement a simple range estimation technique, but it requires accurate camera alignment and a number of assumptions to be made.

Assumptions:

- No radial lens distortion.
- The cameras are aligned to be perfectly parallel (parallel optical axes). This also implies that the "vertical disparity" between images is zero (ie. their optical axes intersect the same horizontal datum).
- We know the camera baseline, pixel dimension and focal length accurately.

A perfect parallel camera situation is illustrated in Figure 48. Range estimation uses geometric information from Figure 48 to estimate range via the use of similar triangles (See Figure 49).

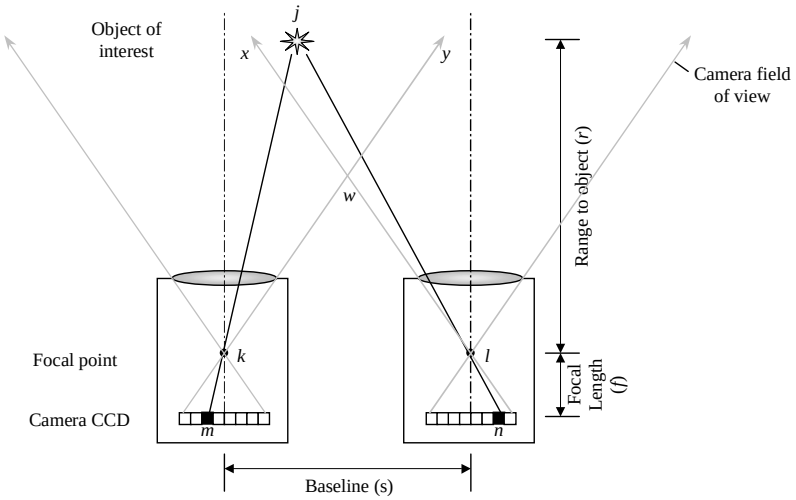


Figure 48: Stereo Camera Arrangement (Parallel) [1]

Once visual correspondence is achieved, the corresponding disparity can be calculated. From Figure 49, we can derive an equation to derive the target range distance.

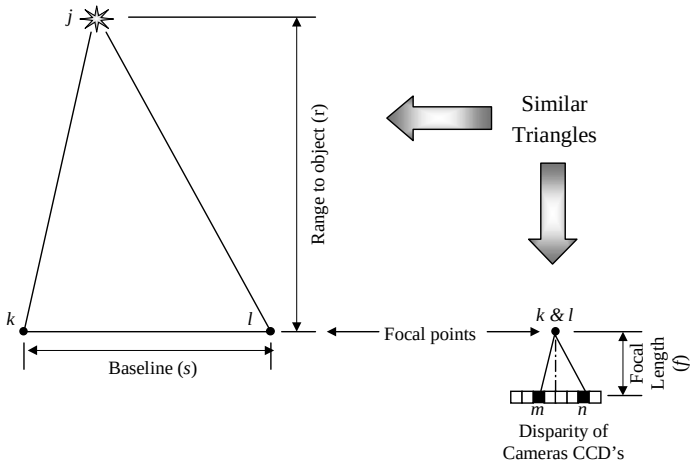


Figure 49: Range Estimation

So the target range can now be calculated via Equation 1:

$$\frac{\text{baseline}}{\text{Range to object}} = \frac{\text{Pixel disparity} \times \text{pixel dimension}}{\text{focal length}} \quad \text{.....} \quad \textbf{(8)}$$

$$\therefore \text{Range to object}(r) = \frac{\text{baseline}(s) \times \text{focal length}(f)}{\text{Pixel disparity} \times \text{pixel dimension}} \quad \text{.....} \quad \textbf{(9)}$$

Equation 1: Range Estimation [1]

This will yield the correct result if the assumptions made earlier are valid. If, however, some of these details are not accurately known, or the cameras are not perfectly aligned, a more sophisticated model is required. This was the motivation for developing position estimation with arbitrary camera orientations.

Appendix F: Camera Mount Considerations

The camera is capable of turning its centre of view ±100° from home in pan, and ±25° in tilt. At minimum zoom, the field of view is ±19° vertically, and ±24.5° horizontally (with respect to its centre of view). Hence there was no requirement for the mount to be adjustable, so long as it was positioned appropriately. It had also been decided that the electronic components would be mounted in the upper enclosure as part of a single super-structure that could be easily inserted and removed. Thus the camera mount was to interface with this super-structure (call it the hardware tray) which is similar to a deep, narrow drawer.

Initially it was thought that the camera would be suitably mounted if its centre of view were aligned along the upper enclosures centre line. Upon further consideration, it was decided that the majority of the observations made by the AUV would be at a downward angle (tracking a pipe or looking at coral for example). An arbitrary figure was chosen such that it would be possible to see whatever was 80±5° below the centre-line of the upper enclosure. The camera is also able to see 124.5° from its home position in pan, and should be positioned as far forward within the end-cap to utilize this wide range. This can be easily achieved by trial and error. Thus the dominant parameter was the angle at which the camera would be mounted. See Figure 50:

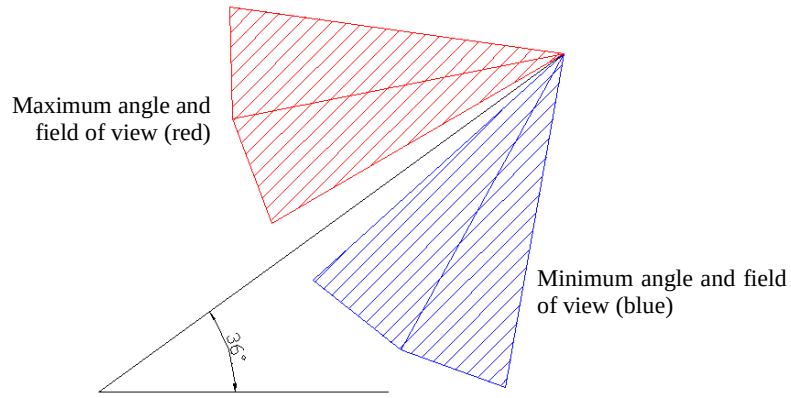


Figure 50: Camera mount angle

So the mount was to be mounted at 36° from horizontal, give or take 5° (we only used an arbitrary starting value). The red cross hatching indicates the field of view at minimum zoom, maximum tilt (positive tilt), and the blue hatching indicates the field of view at minimum zoom, minimum tilt (negative tilt). Once we bent a piece of perforated Aluminium into our hardware tray, it was decided that a first attempt at making the mount could be done simply by bending another piece of perforated Aluminium to give the required 36° angle as shown above. The mount and hardware tray could then be attached by inserting a number of bolts through the perforation holes, and securing with Nylon locking nuts (or otherwise).

Eventual Solution

The dimensions of the mount were not of great concern, except for the trade-off between weight and adequate support (and fitting inside the structure). The dimensions selected were

100 ± 10mm (mount width), 180 ± 10mm (camera base section) and 320 ± 10mm (tray attachment piece). The Aluminium thickness was 1.3mm. See Figure 51 for a diagram. To prevent the mount from vibrating, we effectively clamped the mount in place using threaded cross members that were inserted through the sides of the hardware tray.

When attaching the mount to the hardware tray, an attempt was made to mount it as far forward into the dome as possible, and also to mount it as low as possible within the enclosure (via spacers). Low mounting was desirable due to imperfections in the dome at its centre point. The Sony camera base also had to be modified by adding a number of countersunk screws arranged to slot into the perforations in the Aluminium. When secured with lock nuts, this prevented the camera from moving on the mount.

There has been no need for any worked drawings of the camera mount due to its simplicity. The camera mount was made out of 'scrap' and it had no strict dimensional requirement, just that the camera could attach to it, and it would fit in the upper enclosure. The final dimensions are given in Figure 51 (all dimensions in mm):

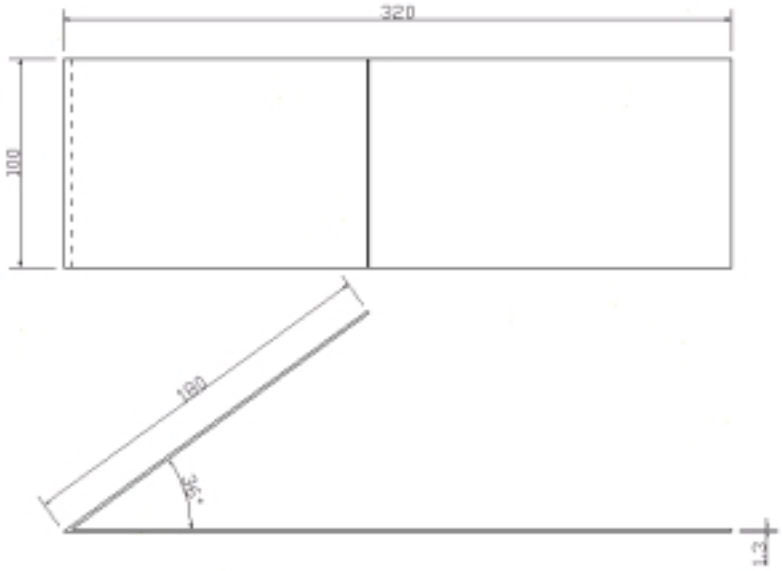


Figure 51: Camera Mount Specifications

The material used was 1.3mm thick, perforated Aluminium. The perforations were 3mm in diameter, centres separated by 5mm.

Once the camera was positioned in the upper enclosure, with dome attached, a video test was eventually performed. At minimum zoom, the camera can see between the two uprights at the front of Kambara – this was essentially a perfect result. Once the camera is moved in different directions, the uprights occlude targets, but this is acceptable, as no image processing will be done using this camera. Having a clear field of view in the home position is useful because the calibration routine will most likely be executed in this position. Hence it is possible to fill the cameras view with the calibration target (rather than filling it partially with the calibration target and partially with Kambara framework).

Appendix G: Derivation of Tsai's Governing Equations

Where point O is the centre of a camera's CCD. We can attach a reference frame to the centre of the CCD (Frame 0). At a distance f, from the CCD plane, we can define a camera reference frame,
(Frame 1), whose z axis pierces the image plane at its centre, O. This coordinate system is indexed by (x_1,y_1,z_1). Frame W, (X_w,Y_w,Z_w) is arbitrarily located somewhere in free space. The translation and rotation transformation matrix that maps Frame W to Frame 1 is known (determined via calibration). Note that if we know the world coordinates, we also know the coordinates with respect to the camera, since the vector O_wO_1 is known.

Assuming a perfect pinhole camera, we can find the location of (Xu,Yu) using Figure 52.

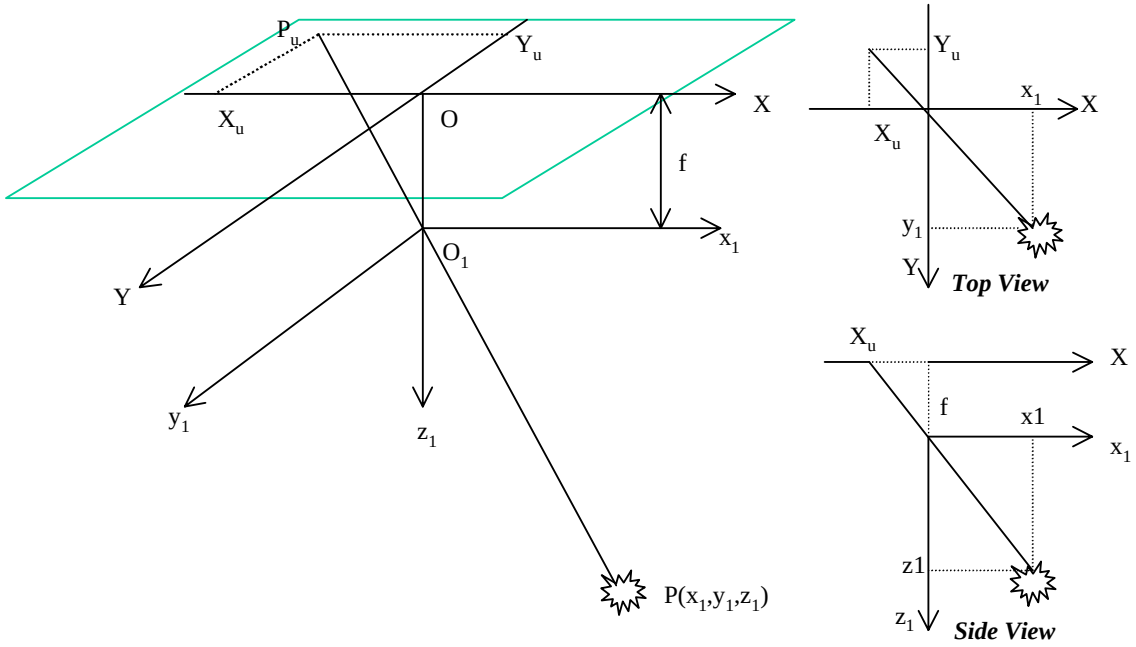


Figure 52: Pinhole Result

Via similar triangles (looking at the Top View and Side View respectively), it can be seen that:

$$\frac{X_u}{Y_u} = \frac{x_1}{y_1}, \text{ and } \frac{X_u}{f} = \frac{x_1}{z_1} \quad \ldots\ldots\ldots\ldots\ldots\ldots \textbf{(10)}$$

$$X_u = f.\frac{x_1}{z_1}, \text{ and } Y_u = f.\frac{y_1}{z_1} \quad \ldots\ldots\ldots\ldots\ldots\ldots \textbf{(11)}$$

Having determines the undistorted image parameters from the geometry of Figure 52 we can now apply the radial distortion model. We model radial distortion (assuming only first order distortion) by letting:

$$X_d + D_x = X_u \text{ and } Y_d + D_y = Y_u \quad \textbf{(12)}$$

$$D_x = X_d(\kappa_1.r^2) \text{ and } D_y = Y_d(\kappa_1.r^2) \quad \textbf{(13)}$$

Where:

$$r = \sqrt{X_d^2 + Y_d^2} \quad \textbf{(14)}$$

Next we relate these distorted coordinates to the computer memory buffer (or frame) locations. This refers to pixel values.

$$X_f = s_x d'_x{}^{-1} X_d + C_x \quad \textbf{(15)}$$

$$Y_f = d_y{}^{-1} Y_d + C_y \quad \textbf{(16)}$$

Where:

- (X_f, Y_f) are the pixel coordinates
- (C_x, C_y) are the pixels at which radial distortion is centred
- d'_x is $d_x(N_{cx}/N_{fx})$
- d_x is the centre to centre distance between pixels on the CCD (given in camera hardware specifications)
- N_{cx} is the number of CCD sensor elements in the x direction (given in camera hardware specifications)
- N_{fx} is the number of pixels in a line as sampled by the computer
- s_x is an uncertainty parameter that accounts for variations in hardware timing.

Then X and Y are defined as:

$$X = \frac{X_d.N_{fx}}{d_x.N_{cx}} \quad \textbf{(17)}$$

$$Y = Y_f \quad \textbf{(18)}$$

These equations form the base of his relationship between image and world coordinates. We combine **Eq's 10-18**, to get:

$$s_x{}^{-1} d'_x X + s_x{}^{-1} d'_x X\kappa_1 r^2 = f\frac{x}{z} \quad \textbf{(19)}$$

$$d'_y Y + d_y Y\kappa_1 r^2 = f\frac{y}{z} \quad \textbf{(20)}$$

Where

$$r = \sqrt{(s_x{}^{-1} d'_x X)^2 + (d_y Y)^2} \quad \textbf{(21)}$$

Then finally, using the extrinsic parameter relationship:

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = R \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} + T \qquad \textbf{(22)}$$

Equations 10 and 11 now become:

$$s_x^{-1} d'_x X + s_x^{-1} d'_x X \kappa_1 r^2 = f \frac{r_1 x_w + r_2 y_w + r_3 z_w + T_x}{r_7 x_w + r_8 y_w + r_9 z_w + T_z} \qquad \textbf{(23)}$$

$$d'_y Y + d_y Y \kappa_1 r^2 = f \frac{r_4 x_w + r_2 y_w + r_6 z_w + T_y}{r_7 x_w + r_8 y_w + r_9 z_w + T_z} \qquad \textbf{(24)}$$

Using this information and ***r*** from **(21),** calibration can continue. We will be using a non-coplanar (more than one plane) calibration target. The method used to calibrate the vision system with a non-coplanar target is now described.

The cameras are then calibrated using the two-stage technique suggested by Tsai: First, find the 3D position, then compute the intrinsic parameters. Intrinsic parameters are calculated in iterations, with initial guesses derived by assuming there is no lens distortion. Since we are using a target consisting of more than one plane, we also set s_x to 1 initially, until it is calibrated. (See [3] for more information regarding the actual calibration method – the important information is found in (10)-(23)).

Appendix H: Programmer Reference: List class

This is the key class that is used during the calibration process, so should be explained. It has the capacity to be altered into a FIFO buffer with the implementation of a "FIFOpop" primitive. Note that it was originally implemented by Texas instruments, and used C++ templates which are not supported under VxWorks. Hence a pseudo templating method was devised. The list structure is explained in Figure 53.

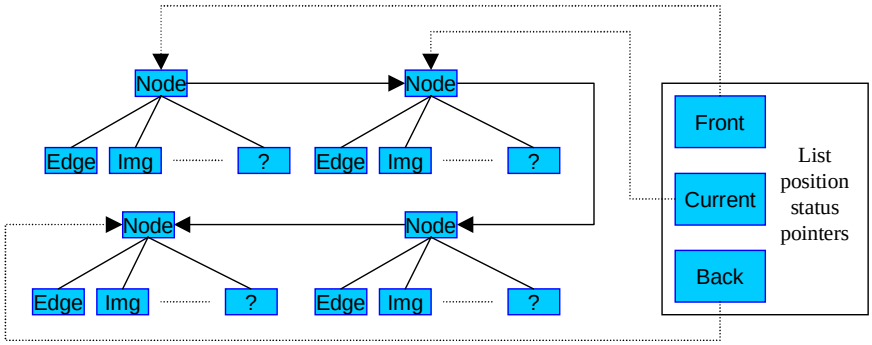


Figure 53: Basic List Structure

There are a number of built in operations that can be performed on these lists. They include pushing, popping, evaluation, list reversal, node removal, and internal operation to move the "Current" pointer to desired locations (searches). The most commonly used primitives are push, pop and value (evaluation). These are explained in Table 13 (Next page).

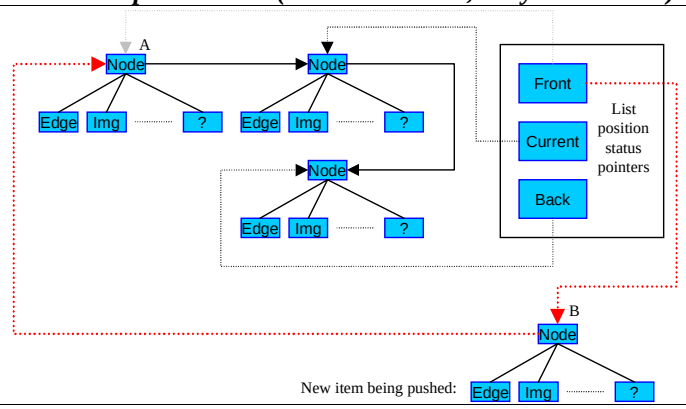
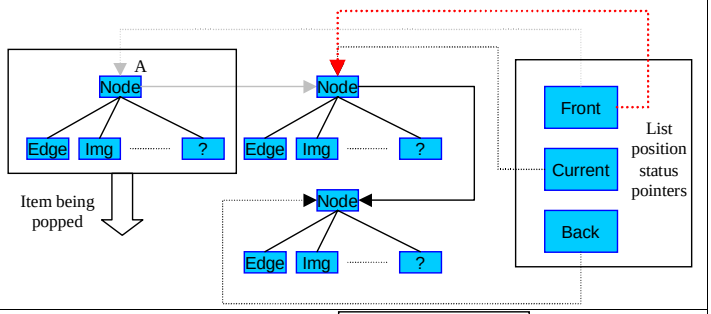
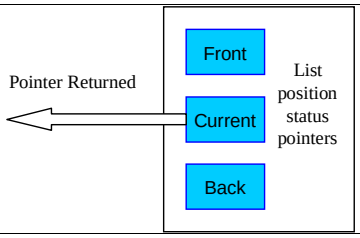
	Operation Explanation	*Pictorial Representation (Red = NEW link, Grey = OLD link)*
Object Push	The list maintains a "Front", "Back" and "Current" pointer. In the simplest *push* case, a node is introduced to the list, and attached at the Front. This requires Node B to be referenced as the NEW "Front", and Node A is the NEXT Node in the list.	
Object Pop	A pop operation is simpler. It finds whatever node is referenced by "Front", sets the NEW Front to be the Node AFTER Node A, and returns Node A. FIFO operation could also be implemented by writing a "FIFOpop" routine, where "Back" will be returned to the application.	
Evaluation	Evaluation involves merely returning a pointer to the Node that is referenced by "Current"	

Table 13: Basic List Primitives

Appendix I: Position Estimation Algorithm

This derivation is based on the rear projection model, however the implemented solution uses the front projection topology. The derivation method is very similar however.

To derive position estimated (X,Y,Z) from pixel coordinates in two cameras….

We start by re-arranging and merging (12,13,14) to give:

$$D_x = X_u - X_d \text{ and } D_y = Y_u - Y_d \quad \textbf{(25)}$$

$$D_x = X_d(\kappa_1(\sqrt{X_d^2 + Y_d^2})^2) \quad \textbf{(26)}$$

$$D_y = Y_d(\kappa_1(\sqrt{X_d^2 + Y_d^2})^2) \quad \textbf{(27)}$$

Hence,

$$X_u = \kappa_1 X_d^3 + \kappa_1 X_d Y_d^2 + X_d = X_d(\kappa_1 X_d^2 + \kappa_1 Y_d^2 + 1) \quad \textbf{(28)}$$

$$Y_u = \kappa_1 Y_d^3 + \kappa_1 Y_d X_d^2 + Y_d = Y_d(\kappa_1 X_d^2 + \kappa_1 Y_d^2 + 1) \quad \textbf{(29)}$$

Thus we have $X_u = f_1(X_d,Y_d)$, $Y_u = f_2(X_d,Y_d)$, where (X_d,Y_d) are the observed ***CCD coordinates*** which have been distorted. Next, we relate image pixel values to these distorted CCD coordinated. From (15,16,17,18):

$$X_f = s_x \frac{N_{fx} X_d}{N_{cx} D_x} X_d + C_x \quad \textbf{(30)}$$

$$Y_f = \frac{Y_d}{d_y} + C_y \quad \textbf{(31)}$$

Hence,

$$X_d = \left(\frac{X_f - C_x}{s_x N_{fx}} \right) N_{cx} d_x \quad \textbf{(32)}$$

$$Y_d = (Y_f - C_y) d_y \quad \textbf{(33)}$$

Now we have the means to convert (X_f,Y_f), which are the observed pixel values, into $(X_d,Y_{,d})$ and hence using (28,29), (X_u,Y_u). Knowing (X_u,Y_u) allows use of the pinhole camera model, as these parameters are no longer affected by radial lens distortion.

This is where the extrinsic parameters come in. We know there are three cameras, and we will assume that they are arbitrarily oriented, but in general, towards the feature. We know the extrinsic parameters such as the rotation and translation matrices required to map a vector in world coordinates into camera frames 0, 1 and 2. The situation is described in Figure 54.

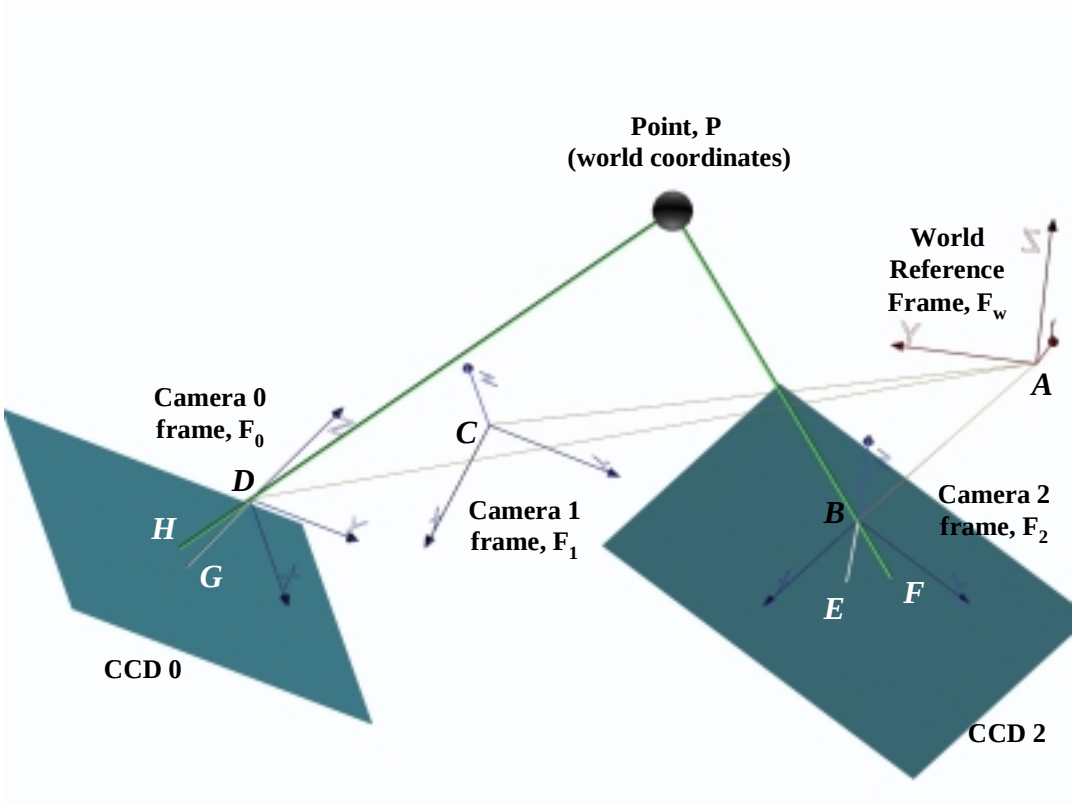


Figure 54: Arbitrary Camera Suite Layout

Figure 54 summary of information:

- World reference frame, centred at A is where the top left-hand corner of the calibration target was located when the system was calibrated.
- Camera's 0 and 2 are Pulnix, 1 is the Sony camera.
- The position we are interested in is the vector from the origin of either F_0 or F_2 (lets use F0).
- We know vectors AC, AC, AD from calibration.
- H, F are where the undistorted points of maximum visual correspondence will be located after the pixel coordinates are mapped via (28,29,32,33).
- E, G are where the optical axes pierce the CCD. ||DG||, ||BE|| = f_0, f_2 respectively.

First let us determine the target location mapping from F0 to F1 (master Pulnix camera to Sony camera). This is given to us indirectly via calibration. We also use the fact that the inverse of a rotation matrix is equal to its transpose.

$$\begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} = R_{W-0} \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} + T_{W-0} \quad \textbf{(34)}$$

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = R_{W-1} \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} + T_{W-1} \quad \textbf{(35)}$$

Hence we can find (x_1,y_1,z_1) in terms of (x_0,y_0,z_0) as follows:

$$\begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} = R_{W-0}{}^{T}\left(\begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} - T_{W-0}\right) = R_{W-1}{}^{T}\left(\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} - T_{W-1}\right) \quad \textbf{(36)}$$

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = R_{W-1}R_{W-0}\left(\begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} - T_{W-0}\right) + T_{W-1} \quad \textbf{(37)}$$

So now once we have estimated the position in terms of F_0, we can directly find the vector in F_1. This will allow direction of the pan-tilt camera directly at the target, no matter how the cameras are arranged.

Next we formulate an expression(s) to find the vector from F_0 to the target. Start by defining a number of vectors. What we know are the following (in terms of the indicated frames)

- Vector $DH(F_w) = R_{0\text{-}w}(X_{u0},Y_{u0},f_0)$
- Vector $BF(F_w) = R_{2\text{-}w}(X_{u2},Y_{u2},f_2)$
- Vector $HF(F_w) = (AD(F_w) + DH(F_w)) - (BF(Fw) + BA(F_w))$

Then,

$$\overrightarrow{DH} \bullet \overrightarrow{HF} = \left\|\overrightarrow{DH}\right\|.\left\|\overrightarrow{HF}\right\|\cos(\theta_0) \quad \textbf{(38)}$$

$$\cos^{-1}\left(\frac{\overrightarrow{DH} \bullet \overrightarrow{HF}}{\left\|\overrightarrow{DH}\right\|.\left\|\overrightarrow{HF}\right\|}\right) = \theta_0 \quad \textbf{(39)}$$

Similarly,

$$\cos^{-1}\left(\frac{\overrightarrow{BF} \bullet \overrightarrow{HF}}{\left\|\overrightarrow{BF}\right\|.\left\|\overrightarrow{HF}\right\|}\right) = \theta_2 \quad \textbf{(40)}$$

There are two options now. We have all the information we need in order to do position estimation, but for more accurate results we could do the following for both cameras, and take the average at the end. This would reduce the effect of pixel quantisation errors.

The next step is to then image the 3D system in 2 dimensions as shown:

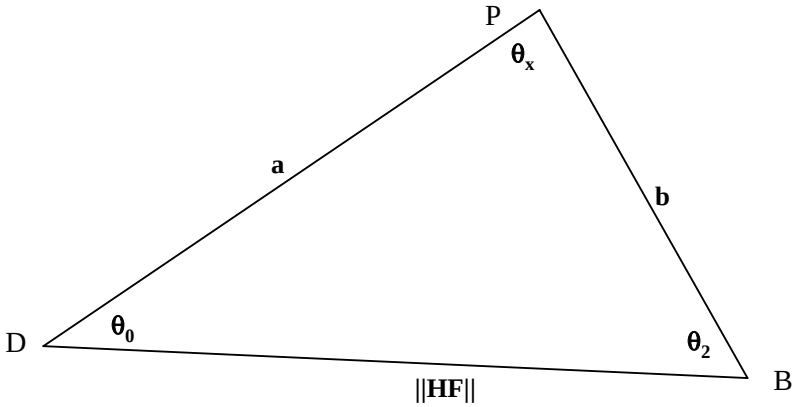


Figure 55: 3D world viewed as 2D

Now, we know $\theta x = 180 - \theta_0 - \theta_2$.
Next, from the sine rule, we can say:

$$\frac{a}{\sin(\theta_2)} = \frac{b}{\sin(\theta_0)} = \frac{\left\|\overrightarrow{HF}\right\|}{\sin(\theta_x)} \tag{41}$$

Or,

$$a = \frac{\sin(\theta_2)}{\sin(\theta_x)}.\left\|\overrightarrow{HF}\right\| \tag{42}$$

$$b = \frac{\sin(\theta_0)}{\sin(\theta_x)}.\left\|\overrightarrow{HF}\right\| \tag{43}$$

Then, from Figure 54, ||DP|| = a - ||DH||. The next step is converting this magnitude into x, y and z components. The situation is described in Figure 56.

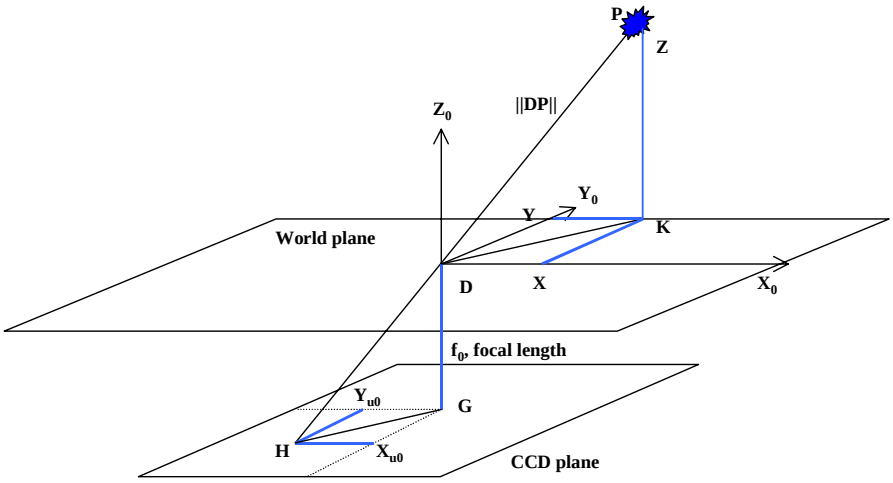


Figure 56: Converting magnitude to X, Y, Z Components

Now, define α to be the angle between DHG, and β to be the angle between $X_{u0}GH$. Note also that inverse tan functions should be implemented using 'atan2'.

Hence,

$$\alpha = \tan^{-1}\left(\frac{f_0}{\sqrt{X_{u0}^2 + Y_{u0}^2}}\right) \quad (44)$$

$$\beta = \tan^{-1}\left(\frac{Y_{u0}}{X_{u0}}\right) \quad (45)$$

$$\left\|\overrightarrow{DK}\right\| = \left\|\overrightarrow{DP}\right\|.\cos(\alpha) \quad (46)$$

And finally by trigonometry again,

$$X = \left\|\overrightarrow{DK}\right\|.\sin(\beta) \quad (47)$$

$$Y = \left\|\overrightarrow{DK}\right\|.\cos(\beta) \quad (48)$$

$$Z = \left\|\overrightarrow{DP}\right\|.\sin(\alpha) \quad (49)$$

And that yields a 3D vector from camera 0 to the point. The range can then be evaluated in X, Y and Z directions.

Appendix J: Tracking Optimisations

When used to implement tracking, the downfall of these methods is processing speed. Currently, only simple optimisations have been used in generating the correlation maps. These methods involve reducing the search area by a number of methods. These include:

- Searching a window of certain size in the region of the last locations (Window)
- Searching this window and only correlating every n^{th} pixel in the template (Window, Sparse)
- Searching the right image by approximating the epipolar geometry of the camera setup. Currently this method assumes the cameras roll angles are both zero (Epipole)
- Searching via the epipolar geometry using a sparse correlation method (Epipole, Sparse)

Window Search

When a template is extracted by selecting the region of interest, the centre of this rectangle can be used as the first correlation peak. To find the peak value in the correlation map on the next iteration (assuming fast enough frame rates), a window can be placed over which to search. The centre of this window is located at the last known correlation peak. For each iteration, a new window search region is used.

Sparse Correlation

Sum of absolute differences uses the absolute pixel intensity difference between the image extract *E*, and template *T*, to provide a measure of *R*. *R* can also be estimated if we ignore every n^{th} pixel. The situation is described in Figure 57. When coupled with a windowing search, overall correlation times can be drastically reduced.

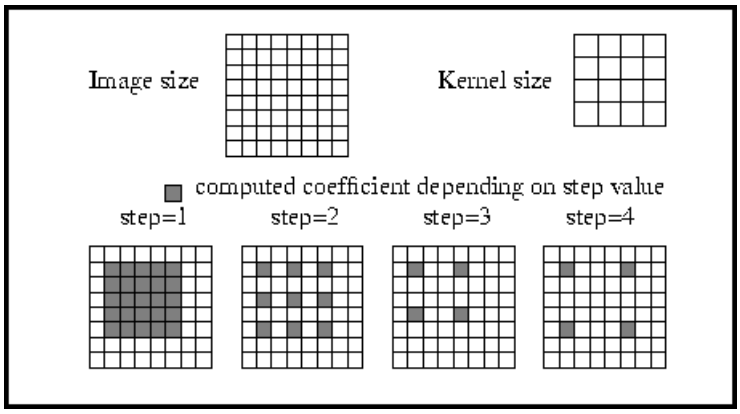


Figure 57: Sparse Correlation [6]

Epipolar Search

Epipolar geometry exists between a calibrated pair of stereo cameras. Once the template is located in one image, the epipolar line for the second image can be determined. It represents the possible point locations of the target, given the relative camera orientations. In the vision system epipolar search implementation, it is assumed that the epipolar lines are horizontal,

and at the same level. This is not a valid assumption given the goal that the vision system will work with arbitrary camera arrangements, but provides a reasonable estimate for testing purposes.

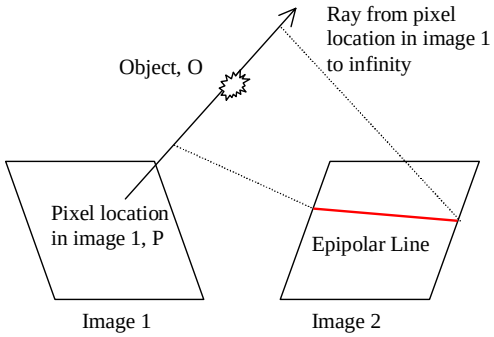


Figure 58: Epipolar Geometry

Appendix K: Raw Results from Calibration Experiments

The following tables contain the raw data used in determining confidence intervals for the calibration results. A *t*-distribution is assumed. Intrinsic and extrinsic parameters are given.

INTRINSIC

LABORATORY RESULTS

Camera 0 (MASTER)		*Confidence Measure: 95%*			
Record	***Focal Length***	***Kappa1 (1/mm^2)***	***Distortion Centre (X)***	***Distortion Centre (Y)***	***Scale Factor***
1	2.875662	0.035942	343.376728	241.621287	1.031723
2	2.838252	0.039604	334.562930	240.347188	1.029102
3	2.836768	0.039413	332.889251	239.427834	1.029518
4	2.838252	0.039604	334.562930	240.347188	1.029102
5	2.836768	0.039413	332.889251	239.427834	1.029518
6	2.855399	0.038040	334.964986	239.954696	1.031039
7	2.830370	0.037520	350.896498	242.377896	1.031451
8	2.859460	0.032135	338.232176	240.523329	1.029285
Mean	**2.846366**	**0.037709**	**337.796844**	**240.503407**	**1.030092**
StdDev	**0.015506**	**0.002600**	**6.324898**	**1.030074**	**0.001113**
	Confidence Measure: 95%		***t Value: 2.7760***		
Upper Limit	2.8616	0.0403	344.0045	241.5144	1.0312
Lower Limit	2.8311	0.0352	331.5892	239.4924	1.0290
Deviation	0.0152	0.0026	6.2077	1.0110	0.0011
Percentage	1.069%	13.536%	3.675%	0.841%	0.212%

Camera 1					
Record	***Focal Length***	***Kappa1 (1/mm^2)***	***Distortion Centre (X)***	***Distortion Centre (Y)***	***Scale Factor***
1	5.474847	0.006809	323.257485	230.230631	1.335349
2	6.731066	0.017913	351.729318	278.283982	1.285752
3	6.736995	0.017756	351.854318	279.146197	1.285692
4	6.731066	0.017913	351.729318	278.283982	1.285752
5	6.736995	0.017756	351.854318	279.146197	1.285692
6	5.412343	0.006371	337.183901	252.798983	1.311395
7	5.778512	0.019529	319.546597	239.936683	1.293253
8	5.994771	0.022019	314.578369	243.219649	1.291356
9	5.779534	0.020504	319.394115	239.496743	1.291671
10	6.714515	0.032237	301.789287	246.001657	1.283908
11	6.756892	0.031630	304.263327	245.127480	1.283747
Mean	**6.258867**	**0.019131**	**329.743668**	**255.606562**	**1.293961**
StdDev	**0.567480**	**0.008123**	**19.783078**	**19.102755**	**0.015831**
	Confidence Measure: 95%		***t Value: 2.7760***		
Upper Limit	6.733845	0.025929	346.302016	271.595482	1.307212
Lower Limit	5.783888	0.012332	313.185321	239.617643	1.280710
Deviation	0.474978	0.006799	16.558347	15.988919	0.013251
Percentage	15.178%	71.075%	10.043%	12.511%	2.048%

Camera 2					
Record	***Focal Length***	***Kappa1 (1/mm^2)***	***Distortion Centre (X)***	***Distortion Centre (Y)***	***Scale Factor***
1	2.870122	0.035564	290.680949	233.333796	1.031068
2	2.869746	0.029559	280.629656	226.787363	1.026060
3	2.864085	0.038686	297.955786	236.533960	1.031751
4	2.872286	0.037404	294.929982	235.204990	1.031094
5	2.909264	0.037054	296.441383	235.819312	1.030585
6	2.885976	0.036221	296.993577	232.952174	1.031327
7	2.883526	0.036041	295.246264	231.767590	1.031304
8	2.892668	0.036448	295.666854	231.192157	1.031331
9	2.910237	0.033569	292.216804	231.399976	1.029318
Mean	**2.884212**	**0.035616**	**293.417917**	**232.776813**	**1.030426**
StdDev	**0.017085**	**0.002665**	**5.311299**	**2.977181**	**0.001779**
	Confidence Measure: 95%		***t Value: 2.7760***		
Upper Limit	2.900021	0.038082	298.332639	235.531698	1.032073
Lower Limit	2.868403	0.033151	288.503196	230.021929	1.028780
Deviation	0.015809	0.002466	4.914722	2.754884	0.001647
Percentage	1.096%	13.847%	3.350%	2.367%	0.320%

Table 14: Laboratory Calibration Results (Raw)

INTRINSIC

UNDERWATER

Camera 0 (MASTER)					
Record	***Focal Length***	***Kappa1 (1/mm^2)***	***Distortion Centre (X)***	***Distortion Centre (Y)***	***Scale Factor***
1	3.747755	0.000379	341.117197	247.825782	1.032722
2	3.862461	0.007921	333.421506	240.652585	1.028741
3	3.791117	0.008627	342.016206	238.729617	1.032436
4	3.801936	0.009277	340.416353	236.661111	1.032155
5	3.836325	0.008687	339.863598	239.665735	1.030779
6	3.875772	0.009354	341.264529	237.444705	1.030657
7	3.728929	0.010189	327.564370	250.050361	1.030498
8	3.691449	0.008860	352.337860	240.648905	1.032099
9	3.836627	0.008732	336.464042	240.604959	1.031119
10	3.833821	0.007773	348.677595	243.899575	1.031872
11	3.846825	0.007505	341.027127	243.613125	1.030192
12	3.849175	0.006915	344.497873	242.446664	1.030341
Mean	**3.808516**	**0.007852**	**340.722355**	**241.853594**	**1.031134**
StdDev	**0.057867**	**0.002517**	**6.467235**	**3.998712**	**0.001157**
	Confidence Measure: 95%		***t Value: 2.7760***		
Upper Limit	3.854889	0.009869	345.904952	245.058010	1.032062
Lower Limit	3.762143	0.005835	335.539757	238.649177	1.030207
Deviation	0.046373	0.002017	5.182598	3.204416	0.000928
Percentage	2.435%	51.378%	3.042%	2.650%	0.180%

Camera 2					
Record	***Focal Length***	***Kappa1 (1/mm^2)***	***Distortion Centre (X)***	***Distortion Centre (Y)***	***Scale Factor***
1	3.827895	0.005510	290.883227	234.330580	1.030563
2	3.769339	0.009446	298.346053	230.717262	1.031560
3	3.816811	0.008109	304.077271	236.332026	1.030427
4	3.755960	0.006534	293.098329	231.819008	1.028556
5	3.835747	0.007636	319.565542	236.271259	1.031502
6	3.815070	0.010769	310.630591	241.726166	1.032418
7	3.830256	0.007015	298.666761	238.049906	1.031273
8	3.841368	0.011075	305.579719	223.409300	1.030071
9	3.840027	0.007125	301.351312	231.976621	1.027193
10	3.793984	0.006125	287.249117	229.330010	1.030289
11	3.836529	0.006440	297.382041	230.550231	1.028861
Mean	**3.814817**	**0.007799**	**300.620906**	**233.137488**	**1.030247**
StdDev	**0.029377**	**0.001869**	**9.197081**	**4.940832**	**0.001528**
	Confidence Measure: 95%		***t Value: 2.7760***		
Upper Limit	3.839405	0.009363	308.318821	237.272942	1.031525
Lower Limit	3.790229	0.006234	292.922990	229.002034	1.028968
Deviation	0.024588	0.001565	7.697915	4.135454	0.001279
Percentage	1.289%	40.127%	5.121%	3.548%	0.248%

Table 15: Underwater Calibration Results (Raw)

EXTRINSIC

LABORATORY

Intermediate Parameters					
	Attempt 1	***Attempt 2***	***Attempt 3***	***Attempt 4***	***Attempt 5***
Tx2	-262.922266	-224.640515	-224.640515	-258.468623	-248.483398
Ty2	-117.183272	-279.771670	-279.771670	-192.838490	-232.584122
Tz2	1029.975286	1076.939565	1076.939565	885.747250	1063.964057
Tx0	148.225058	201.299523	201.299523	163.802863	238.409893
Ty0	-152.242989	-295.803344	-295.803344	-205.574372	-234.012666
Tz0	1036.157636	1054.022032	1054.022032	908.338943	1068.998047
D-Bx	411.147324	425.940038	425.940038	422.271486	486.893291
D-By	-35.059717	-16.031674	-16.031674	-12.735882	-1.428544
D-Bz	6.182350	-22.917533	-22.917533	22.591693	5.033990
Effective Baseline					
magDB	412.68575	426.85729	426.85729	423.06713	486.92141
Should be approximately 450mm			***Mean Baseline: 435.27777***		

UNDERWATER

Intermediate Parameters					
	Attempt 1	***Attempt 2***	***Attempt 3***	***Attempt 4***	***Attempt 5***
Tx2	-57.867133	52.058255	-271.371333	-277.873594	-164.702852
Ty2	60.317092	-402.505417	-96.806755	-307.205273	-221.135022
Tz2	1240.504626	1759.294244	902.056791	877.094248	879.973829
Tx0	418.225331	610.553869	197.205316	165.829671	266.079621
Ty0	62.645158	-434.517095	-120.233919	-320.947525	-238.252531
Tz0	1267.010415	1705.082012	908.317999	877.390538	895.361025
D-Bx	476.092464	558.495614	468.576649	443.703265	430.782473
D-By	2.328066	-32.011678	-23.427164	-13.742252	-17.117509
D-Bz	26.505789	-54.212232	6.261208	0.296290	15.387196
Effective Baseline					
magDB	476.83541	562.03297	469.20370	443.91612	431.39693
Is 450mm when measured above the surface			***Mean Baseline: 476.67703***		

Table 16: Extrinsic Calibration Parameters (Raw)

Appendix L:
Autonomous Submersible Robot: Vision System: Proposal

Abstract

This project involves the further development of a Vision System for an autonomous underwater vehicle and builds on work completed during 1998 [Reynolds 1998]. The major goal of this project is to develop a system that will point a manoeuvrable on-board camera at a selected target. To implement such a system, five major areas of research and development will be identified:

- Develop image digitising software
- Design and install a mount for the Sony camera
- Improve current camera calibration software
- Develop algorithms for underwater image processing and feature tracking
- Test the feature tracking system

Each will be discussed, outlining the sub-tasks that are anticipated to require the most attention. Finally a schedule for the year is determined based on the identified tasks. This schedule can then be evaluated at the end of the year.

Background

The ocean exists as one of our most important natural resources, providing industry with both important minerals, and everlasting tidal energy. While currently under-utilised, it is certain that we will eventually turn to the ocean for its vast supply of natural resources.
However, once industry moves to the ocean, problems arise. We have already designed colossal engineering masterpieces that ride on the ocean surface or in some cases below, but the task of maintenance is time consuming, expensive and sometimes difficult. This is because the areas to be inspected and maintained are predominantly underwater.
The ANU is currently developing an experimental Autonomous Underwater Vehicle (AUV) which could provide a convenient, cost effective solution to this problem. The AUV, named Kambara, is fitted with visual equipment that would make underwater maintenance - and other underwater tasks - significantly easier. Also, since the vehicle is to be autonomous, there will be no need to waste numerous man-hours performing manual vessel inspections - when Kambara could do it automatically.
The possibilities go further than just ocean vessel maintenance. Such an AUV would also provide a useful tool for observing marine wildlife, and monitoring environmental conditions. But first, the prototype has to be developed.

Proposal

This project is concerned with the development of the vision system for Kambara. The vision system hardware includes three cameras - two Pulnix cameras and one Sony pan-tilt-zoom camera. The Pulnix cameras are used to implement stereo vision, which allows depth

perception. Because of its manoeuvrability, the Sony camera will be used by the feature tracking routines. Vision system software support for the above hardware include a device driver for image capturing and a device driver to implement the RS-232 Visca commands that allow the pan-tilt-zoom mechanism to be controlled. A PowerPC which operates at 233MHz, will perform the on board image processing.
The navigation system on-board Kambara could involve one or two layers. A one-layer system would mean that visual data could be used to directly control the thrusters. A two-layer system might have the following configuration: one layer uses the visual data to fix the Sony camera on the target, while the second layer would use the Sony camera position to control the sub movement. The major goal of this project will be to fix the Sony camera on the target, thus implementing the first layer of a two-layer navigation system.
Once the project is complete, I hope to have further developed my understanding of effective computer vision techniques, effects of an underwater environment on image processing, how to minimise the effect of noise on images, and I hope to enhance my understanding of vision based control.
Specific components of the vision system are identified below, accompanied by a description of work that will need to be addressed during the course of the year. The items are listed in the order they will be addressed - each successor generally depends on adequate completion of the previous task.

Image Digitizing Software

Before any image processing is to be done on-board Kambara, the images from the cameras need to be digitised. Each camera outputs S-Video data, which will in turn be sampled by an Imagenation PXC200 on-board frame-grabber. There will only be one frame-grabber on board - a compact PCI card with 4 S-Video inputs. This will allow video-input selection/alternation while sampling at a relatively high speed (20Hz as a minimum). We already have access to a frame-grabber that runs under VxWorks, but the driver needs some modifications. Improvements include:

- Advancing the speed of image capture. Currently the digitiser can acquire images at a rate of 20Hz in High-resolution mode (640 x 480). While this speed should be adequate – tests have only been performed on an Intel Pentium 300, whereas we will be using a PowerPC 233. Obviously there may be some porting difficulties due to platform incompatibilities.
- Thoroughly test digitised image data integrity. Noise in the image means correlation will become difficult, sometimes impossible. If a frame contains errors of any kind, it should be discarded. The most desirable objective is for the digitiser to sample with no errors – this may however prove to be impossible, and is to be investigated.
- Ensure a Client/Server model can be implemented using the existing software. The software architecture for Kambara requires all devices and components to be controlled by the 'Vehicle manager'. Thus, the 'Vehicle manager' can be seen as the server, controlling the operation of the frame-grabber - the client is the process that grabs and processes the images.

Design and Install Mount for the Sony Camera

One of the major design tasks accomplished by Jack Reynolds (final year undergraduate in 1998) was to design the Pulnix camera enclosures. These enclosures needed to be watertight and manoeuvrable. Similarly, the Sony camera needs to be mounted on the AUV. It is to be placed inside the upper enclosure, so the design should be significantly simpler. Also, the pan-tilt-zoom mechanism means that the mount need not move. There may be directions in

which it would be desirable for the mount to slide – this will be determined during the year. Tasks related to designing the Sony camera mount are:

- Assess the camera positioning requirements, and design the mount to allow movement in the required directions.
- Design the mount using AutoCAD – according to HB7 specifications.
- Have mount manufactured by Technical Staff Sponsor (TSS – see below), then install it in the upper enclosure on Kambara.

Camera Calibration

Once the cameras are mounted, and the image digitising software is functional, the cameras need to be calibrated. There are six extrinsic and four intrinsic camera parameters [Nalwa 1994] that need to be determined or measured before the camera suite can be used as a whole. These parameters can be determined using existing algorithms such (eg. Tsai). Tasks to be addressed for calibration are:

- Assess and develop a calibration target. Currently, the calibration target is a laser-printed pattern of unique rectangles on a piece of paper – glued to a cardboard frame. This is not acceptable as cardboard can be bent, thus the target will be distorted. The new calibration target may have to be made of a different material and/or printed in a different fashion
- Improve existing software to calculate the above parameters. The problem is that the target has to occupy the cameras complete field of view and if the target is at an angle, diagonal information tends to be lost. Tsai's algorithm can cater for diagonal orientation, but this functionality has not been implemented in the program yet.
- The calibration software will then need to be ported to VxWorks and tested.

Develop algorithms for image processing underwater

The next and possibly the most important vision system component is the software that will be used to judge depth, track features and perform image processing (correlation, difference of Gaussians (DOG), edge detection etc). Currently we have access to software that operates under Windows NT, which tracks features in mono and stereo. Issues to be addressed during this year in regards to image processing include:

- Development of further feature tracking algorithms, accompanied by attempted performance enhancements.
- The above algorithms need to be tested underwater to evaluate their effectiveness when subjected to noise such as bubbles and floating debris. Perhaps a low pass filtering technique could be used to remove these close-up disturbances.
- Each algorithm should be evaluated against each other to see which is most appropriate in underwater conditions.
- Background research into the area of underwater image processing should also be made before developing any algorithms.
- This software will need to be ported to VxWorks.

Testing of Image Tracking Algorithms

A three stage testing process has been proposed. The first stage involves obtaining images in a laboratory environment, then testing the tracking algorithms using this data. The second stage involves obtaining image data collected underwater, which can then be used to further

develop the tracking algorithms. The final stage will be testing the effectiveness of the system once installed onboard Kambara.

Schedule

A project schedule has been attached at the end of this document.

Assessment Milestones

The major course requirements should also be addressed in the proposal. The dates for each of these major assessment milestones are shown in the schedule. The major assessment items are:

- Written Proposal
- Progress Report
- Seminar
- Draft Thesis
- Demonstration
- Thesis and notebook

Technical Staff Sponsor (TSS) and Resources

The TSS for the Kambara project has not yet been determined. All other resources that will be required for this project are software based – and are provided by the Systems Engineering Department at RSISE.

Conclusion

The vision system project requirements have been assessed, and a number of distinct research areas have been defined. The major project goal will be to develop a system that uses visual data to direct the Sony pan-tilt-zoom camera at the desired target. Finally, a schedule has also been proposed, which takes into account all of the above research areas, in conjunction with project deliverables.

References

Reynolds, J. 1998. *Autonomous Underwater Vehicle: Vision System*, ANU, ACT
Nalwa, V. 1994. *A guided tour of Computer vision*, Addison-Wesley, Reading, Massachusetts

Schedule

The corresponding Gantt chart can be found (shrunk) on the following page.

Task	Duration	Start	Finish
Image Digitizing Software	***25d***	***Mon 3/1/99***	***Fri 4/2/99***
Advance Speed	21d	Mon 3/1/99	Mon 3/29/99
Thorough Test	4d	Tue 3/30/99	Fri 4/2/99
Implement and test Client/Server architecture	15d	Mon 3/1/99	Fri 3/19/99
Digitizing Software finalised	*0d*	*Fri 4/2/99*	*Fri 4/2/99*
Perform background research	**14d**	**Mon 4/5/99**	**Thu 4/22/99**
Design and install mount for the Sony camera	***52d***	***Fri 4/23/99***	***Mon 7/5/99***
Assess Requirements	5d	Fri 4/23/99	Thu 4/29/99
Design Mount	16d	Fri 4/30/99	Fri 5/21/99
Manufacture Mount	28d	Mon 5/24/99	Wed 6/30/99
Install mount	3d	Thu 7/1/99	Mon 7/5/99
Mounts installed	*0d*	*Mon 7/5/99*	*Mon 7/5/99*
Camera Calibration	***35d***	***Mon 5/24/99***	***Fri 7/9/99***
Improve Calibration Target	10d	Mon 5/24/99	Fri 6/4/99
Improve Calibration Software	21d	Mon 5/24/99	Mon 6/21/99
Automate Calibration Software	21d	Mon 6/7/99	Mon 7/5/99
Port to VxWorks	20d	Mon 6/14/99	Fri 7/9/99
Cameras calibrated	*0d*	*Fri 7/9/99*	*Fri 7/9/99*
Develop algorithms for image processing underwater	***104d***	***Mon 5/24/99***	***Thu 10/14/99***
Perform Background Research	15d	Mon 5/24/99	Fri 6/11/99
Develop Feature Tracking Algorithms	36d	Mon 5/24/99	Mon 7/12/99
Test Tracking Algorithms above surface	31d	Mon 5/31/99	Mon 7/12/99
Evaluate/Improve Algorithms	14d	Tue 7/27/99	Fri 8/13/99
Test Tracking Algorithms using underwater data	20d	Mon 8/16/99	Fri 9/10/99
Add in extra functionality if required	7d	Wed 10/6/99	Thu 10/14/99
Port to VxWorks and test on Kambara	21d	Wed 8/18/99	Wed 9/15/99
Algorithms Developed	*0d*	*Wed 9/15/99*	*Wed 9/15/99*
Project Technical Development Done	***0d***	***Wed 9/15/99***	***Wed 9/15/99***
Assessment Requirements	***174d***	***Mon 3/1/99***	***Thu 10/28/99***
Written Proposal	15d	Mon 3/1/99	Fri 3/19/99
Written Proposal Done	*0d*	*Fri 3/19/99*	*Fri 3/19/99*
Background research (for progress report)	14d	Wed 6/30/99	Mon 7/19/99
Progress Report	86d	Mon 3/22/99	Mon 7/19/99
Progress Report Done	*0d*	*Mon 7/19/99*	*Mon 7/19/99*
Seminar	5d	Tue 7/20/99	Mon 7/26/99
Seminar Done	*0d*	*Mon 7/26/99*	*Mon 7/26/99*
More background research (for draft thesis)	14d	Thu 9/16/99	Tue 10/5/99
Draft Thesis	51d	Tue 7/27/99	Tue 10/5/99

Draft Thesis Done	*0d*	*Tue 10/5/99*	*Tue 10/5/99*
Demonstration	10d	Fri 10/15/99	Thu 10/28/99
Demonstration Done	*0d*	*Thu 10/28/99*	*Thu 10/28/99*
More Background Research (for thesis)	21d	Wed 9/29/99	Wed 10/27/99
Thesis and Notebook	173d	Mon 3/1/99	Wed 10/27/99
Thesis and notebook Handed in	*0d*	*Wed 10/27/99*	*Wed 10/27/99*
Assessment Requirements met	*0d*	*Thu 10/28/99*	*Thu 10/28/99*
Project Requirements Complete	***0d***	***Thu 10/28/99***	***Thu 10/28/99***