

An Integrated System for Tracking of
Landmarks on Video Data:
TOMAS
The Torsional
Ocular Movement Analysis System

by

Andrew Edward Alston

Submitted in Partial Fulfillment of
the Requirements of the
Degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

June 1989

© Massachusetts Institute of Technology 1989

Signature of Author _____

Certified by _____
Dr. C.M. Oman, Thesis Supervisor

Accepted by _____
Pr. A. Odoni, CoDirector Operations Research Center



An Integrated System for Tracking of Landmarks on Video Data:

TOMAS, The Torsional Ocular Movement Analysis System

by Andrew Edward Alston

Submitted in Partial Fulfillment of
the Requirements of the
Degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

June 1989

ABSTRACT

An integrated workstation for measurement of video ocular torsion is described. This system determines translation of ocular landmarks automatically by crosscorrelation (based on the "dumbbell" analysis method applied to video (Parker et al. 1985)). Input format is consumer VCR videotape allowing ease of access to the system.

Parker's method was implemented and tested for the first time on actual experiment data. For the data sets used, non linear prefiltering is determined to be the most effective data preparation technique, and test image correction based on reference image mean is added to the correlation process.

Applied to synthetic data consisting of dark marks on a bright background, an average torsion measurement bias of less than 0.05 degrees and a precision of 0.1 degrees (SD) were obtained by the system. Real data was obtained from NASA shuttle vestibulocular experiments. For real data consisting of videotapes of eyes with opaque contact lenses with and two white dot landmarks, precision was estimated at 0.1 degrees. For lower signal to noise ratio transparent lenses with black marks, a precision of 0.45 degrees was obtained. Comparison of the system with photographic measurement techniques support the hypothesis that this method is comparable in accuracy to manual photographic interpretation. System limits are also described.

Experiment setup and lens design improvements are suggested and use of the system for measurement along all three axes is discussed.

Thesis Supervisor: Dr. C.M. Oman

Title: Senior Research Engineer

To Aude,

who made Thomas possible

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. C. Oman for his help and particularly for the time he devoted to the my work over the latter months of this thesis. Sherry Modestino was, as for all students, an irreplaceable help. Her advice, personal and professional, and her efforts to make the lab an enjoyable place are truly appreciated. I would also like to thank Professor A. Odoni, and Marcia Chapman for their help and advice over my two years at MIT.

I am particularly indebted to Mark Shelhamer and Dan Merfeld of the Man Vehicle Laboratory. Mark made the remote control interface for the VCR but beyond this irreplaceable technical help, both he and Dan were always willing to answer questions and think out ideas with me. I am sure that they will both be excellent professors and wish them luck in their endeavors.

Over my stay at the MVL I have been struck by the quality of the students here; without exception they have been interesting, fun and caring and Aude and I will both sorely miss the atmosphere here.

Finally I would like to thank my wife Aude who makes all things possible.

This work was funded by NASA grants NAGW1377, NAS9-16523, and NAS9-15343.

CONTENTS

	Page
INTRODUCTION.....	8
CHAPTER I : The Ocular Torsion Experiments	
at MIT: Data acquisition and Analysis.....	10
A/ The Experiments.....	10
1/ The MIT Sled Experiments.....	10
2/ The Dome Experiments.....	13
3/ The Microgravity Vestibular Investigation (MVI) Experiments.....	16
B/ Analysis Methods.....	16
1/ 35mm Data Analysis.....	17
2/ Search Coil Methods.....	20
3/ Video Based Methods.....	21
C/ Choice of an Integrated Video System	
Performance Characteristics.....	23
1/ Advantages of Video Analysis Methods.....	23
2/ Performance Parameters for Video Systems.....	24
3/ Choice of Video Analysis Method.....	27
CHAPTER II : Automatic Calculation of Image Rotation	
Using the Crosscorrelation Technique.....	28
A/ INTRODUCTION.....	28
B/ The Two Dimensional Fast Fourier Transform.....	30
1/ Convolution and Correlation.....	30
2/ Computational Advantages of the Fourier Transform.....	33

3/ Sampling of Continuous Signals.....	34
4/ Representation and Packaging.....	34
C/ Preliminary Filtering.....	37
1/ Linear Filters.....	37
2/ Thresholding.....	43
D/ Corelation Using the Two Dimensional Fast	
Fourier Transform.....	45
1/ Detection of Integer Translation.....	45
2/ Sub Pixel Translation.....	50
E/ From Translation to Rotation.....	53
F/ Algorithm Implementation.....	54
1/ Data Analysis Pipeline.....	54
2/ The Landmark Tracking Algorithm.....	58
G/ Expected Accuracy.....	62
CHAPTER III : Results.....	64
A/ Method Validation.....	66
1/ Calculation of Translation	
for Synthetic Images.....	68
2/ Synthetic Translation of Stored	
Real Data.....	68
B/ Full System Tests.....	70
1/ Idealized Inputs.....	70
2/ Multiple Image Comparisons,	
Real Data - Transparent Lens.....	72
3/ Multiple Image Comparisons,	
Real Data - Opaque Lens.....	72

C/ Comparison with 35mm Scanning.....	76
D/ System Limits.....	76
1/ Limits of Rectilinear Measurement Technique.....	76
2/ Effect of Blinks.....	78
E/ Operating Characteristics.....	81
CHAPTER IV : Discussion and Recommendations.....	83
A/ Discussion of Results.....	83
1/ System Validation.....	83
2/ System Limits.....	84
3/ Operating Characteristics.....	85
4/ Choice of Hardware/Software.....	86
5/ Conclusion: System Evaluation.....	86
B/ Implications of System Characteristics.....	87
1/ Experiment Development.....	87
2/ Data Analysis.....	89
C/ Suggestions for Further Research.....	89
1/ Measurement of Horizontal and Vertical Eye Movements.....	89
2/ Accuracy Improvement.....	90
3/ Speed Enhancement.....	91
References.....	93
Appendix I : User's Guide.....	95
Appendix II : Programmer's Guide.....	113
Appendix III : Command of Remote Control.....	122
Appendix IV : Source Code.....	126

INTRODUCTION

Space motion sickness has been experienced by as many as fifty percent of the passengers in the Space Shuttle during their first few days in space. This problem is believed to be due to sensory conflict, as the combinations of sensory inputs in weightlessness are very different from those on earth. Vision, and the vestibular system, which includes the otoliths, transducers of linear acceleration, and the semicircular canals, which transduce angular velocity, are considered to be the primary causes of these sensory conflicts. Ocular torsion (OT), the rotation of the eye about the line of sight, can be provoked to evaluate the performance of the otoliths, and to study visual-vestibular interaction.

This thesis describes TOMAS, the first highly accurate automated video system for the measurement of ocular torsion data stored on videotape. This system was developed for analysis of data of the type recorded during the Spacelab Space Life Sciences 1 (SL-1) and Deutsche 1 (D-1) mission MIT-Canadian vestibular experiments and of a type that will be used in the upcoming Spacelab International Microgravity (IML) and Space Life Sciences 1 (SLS-1) experiments. Input data is recorded with a video camera and stored on, or converted to Sony SuperBeta videotape for later analysis. To validate the system, images of human eyes with marked contact lenses and surrounding features were taped, and torsion was calculated and compared with measurements made with other techniques.

Chapter I details the SL-1, D-1, SLS-1, and IML experiments, and the state of the art today in analysis of such data.

Chapter II describes the system that has been developed as a result of this project.

Chapter III presents the results obtained by this system and characterizes its performance.

Chapter IV includes a discussion of these results and suggestions for further research.

I The Ocular Torsion Experiments at MIT:

Data Acquisition and Analysis

Ocular torsion was provoked in two Space Shuttle experiments which were flown once as components of the MIT-Canadian Vestibular Experiments on SL-1 in November 1983 and once on the D-1 mission in October 1985. So far, only the data recorded on 35mm film has been analyzed as no video based analysis system has yet been developed. Upcoming SLS-1, and IML experiments will also require measurement of ocular torsion.

A/ The Experiments

1/ The MIT Sled Experiments

These experiments are based on the following reflex : if a person standing with his head straight up tilts it sideways, his eyes will tend to compensate for the tilt by counterrolling, i.e. rotating in the reverse direction by a few degrees in order to stabilize vision [Miller 1962]. This situation is recreated artificially on a sled by placing the subject in a chair that can be accelerated laterally along rails (figure 1.1). Upon acceleration the subject encounters a force resulting from the sum of gravity and inertia. This force is in the same direction, with respect to his head, as gravity alone in the case of a simple static head tilt (figure 1.2) and provokes ocular torsion. Using the sled allows controlled studies

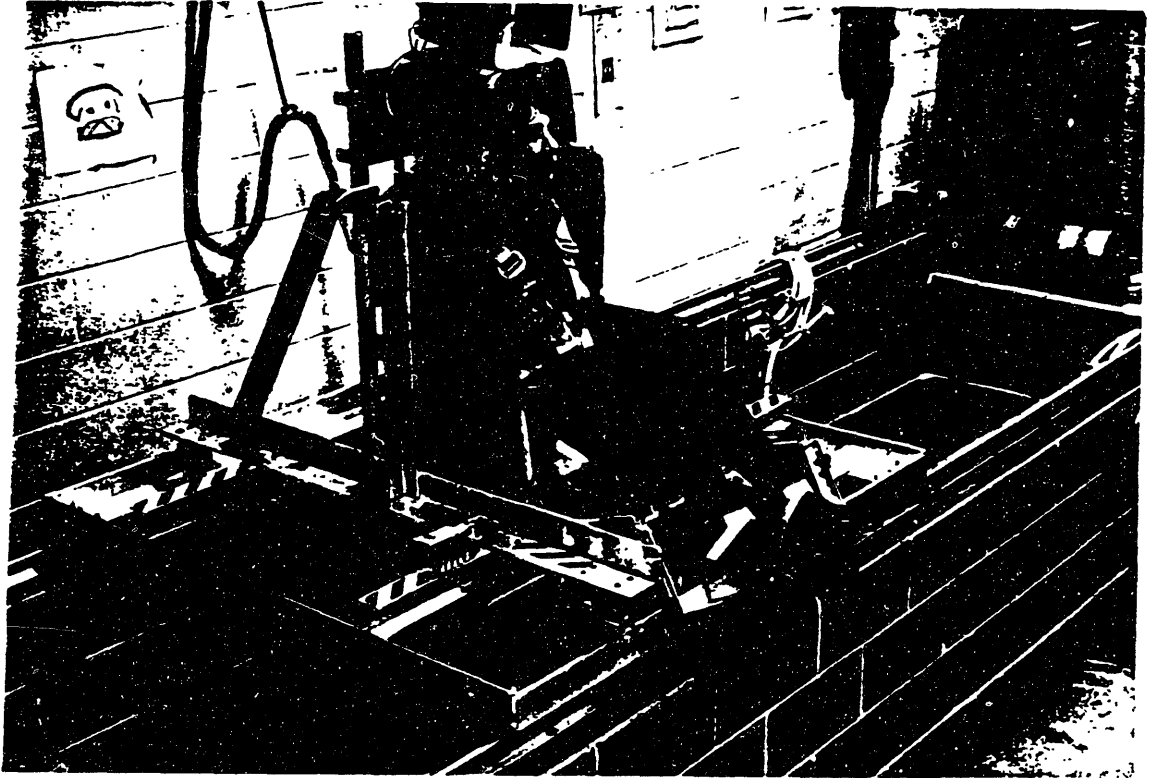


Figure 1.1 : The Sled apparatus at the Man Vehicle Laboratory at MIT. (from Arrott 1985)

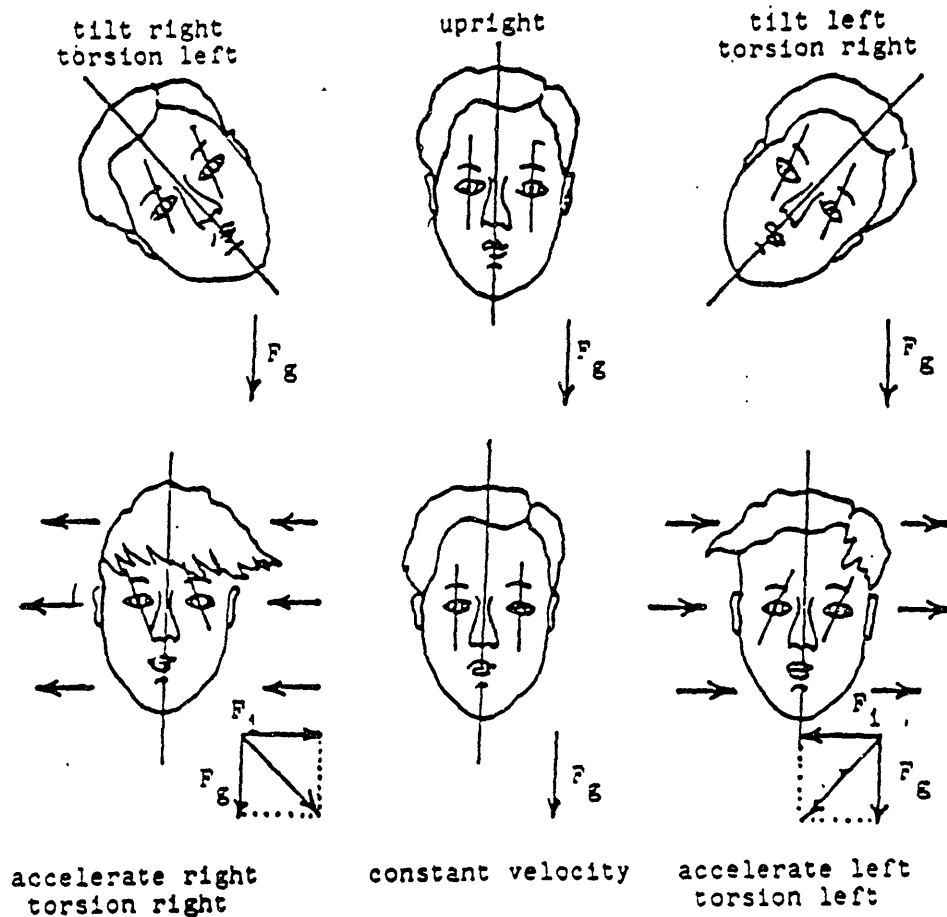


Figure 1.2 : Ocular Torsion induced by lateral acceleration. The sum of the gravitational vector and the inertial force here is in the same direction, with respect to internal axes, as the gravitational force alone when the head is tilted statically. Thus compensatory eye movements are generated. From Arrott (1982).

of the dynamics of the otoliths without interference from the semicircular canals. The MIT sled was built in the late nineteen seventies by Dr. B. Lichtenberg and this and similar experiments have been run on it on at MIT [Lichtenberg 1979, Lichtenberg et al. 1982, Arrott 1985].

On the SL-1 mission subjects were tested pre- and postflight on the MIT sled, and a higher performance NASA ground sled, but no sled was flown in the shuttle. Data was taken by attaching a motor driven Nikon F2 35mm camera and a 55mm lens to the sled, and taking three pictures per second of both of the subjects' eyes. The field of view for these binocular images was approximately 4" x 2 1/2". On the D-1 mission a European Space Agency sled was used in flight, allowing measurements to be taken inflight as well as pre- and postflight. This data was recorded on 35mm film. For pre- and postflight experiment runs on the NASA sled, monocular frames corresponding to 2 1/2" x 2 1/2" fields of view were taken using a Sony XC-37 color video camera with a 43mm lens at a speed of 30 frames per second. Video data was stored on U_matic 1/2" videocassettes and later converted to SuperBeta format.

2/ The Dome Experiments

These experiment center on the feeling of circularvection, or sense of circular motion in the absence of real movement. They are intended to compare the influences of the visual and vestibular systems motion sensation. Vection is induced by putting the subject's head in a rotating "dome" whose base is covered by a pattern of colored dots (figure 1.3). Eye movements are recorded by a camera placed in the center of the base. This



Figure 1.3 : The Rotating Dome Experiment at the Man Vehicle Laboratory at MIT.

does not inhibit the subject asvection is caused by movement of the peripheral field. Vection causes a reaction of the oculomotor system called optokinetic nystagmus (OKN). Torsional eye position is "sawtooth" in shape, with long slow phase movements, followed by rapid fast phase returns to the initial position; it is hoped that this reflex will be correlated with other data such as subjective sensation ofvection. The signal from the dome experiments differs from the sled experiment output in that it is nystagmic: it contains many high frequency saccades which are very rare with the sled experiments.

On the SL-1 and D-1 missions, the experiment was run twice at each pre- and postflight session, once with the subject upright and once supine [Young et al. 1986]. The difference here is that in the first case the subject feels the pull of gravity as an inhibitory effect. Two experiments were also run inflight, one with the subject attached to the floor by elastic cords, and one floating freely. Running the experiment in space allows one to determine whether the visual influence on the sense ofvection increases in weightlessness. During SL-1, data was recorded both on 35mm film with a Nikon F2 camera and a 105mm lens, and on video with a special spacelab video camera; on D-1 only video was used. This video data was recorded in splitscreen format: the left half of the screen contained an image of the subject performing the experiment, and the other half showed the eyeball. This procedure involved editing of the original picture, as only half the picture was stored, rather than a reduced version of the whole picture. Given the small horizontal head and eye movements associated with the dome experiment, the "eyeball" side of the image could be chosen so as to contain the eyeball and very little else. Thus the splitscreen format did

not entail a significant loss of resolution in eyeball size. Spacelab video was converted to NTSC U_matic format, and later transferred to Sony SuperBeta. The upcoming SLS-1 mission will also include similar dome experiments.

3/ The Microgravity Vestibular Investigations (MVI) Experiments

These experiments are scheduled to be performed during the International Microgravity Laboratory (IML) 1 and 2 Spacelab missions beginning in 1991. Measurement of horizontal, vertical, and torsional eye movements is crucial to many of the seven functional objectives of these missions, which will examine the vestibular ocular reflex, and visual vestibular interaction. The subject wears a custom fitted helmet to which a Pulnix camera is attached. Horizontal and vertical eye movements can be measured using equipment from Iscan Inc. The field of view for this set up includes just the subject's eye and eyelid. For some experiments, a custom made corneal-scleral hard contact lens by Medical Instruments Inc. will probably be placed on the eye. This lens was colored black, with two white marks approximately 1/2" apart when pilot data was analyzed for this thesis.

B/ Analysis Methods

Torsion is defined as rotation about the line of sight, but for practical reasons it is measured as the inclination of a meridional line projected into the frontal plane. These two definitions are not exactly equivalent : for example if the subject displaces his gaze laterally and

vertically, a meridian on the eye appears tilted when viewed from the front. This has been called false torsion or "crosstalk" [Collewijn et al. 1985]. In many applications this problem is not significant because the subject's gaze is directed straight ahead. It is possible to geometrically correct for this effect although this was not attempted in this thesis.

Popular methods for the analysis of ocular torsion under this definition include 35mm photography and search coil methods. More recently some groups have attempted to use video based techniques.

1/ 35mm Data Analysis

The first practical method for measurement of ocular torsion between two images was developed by Miller in 1962. He used a prism to superimpose successive test images on a reference image [Miller 1962, Miller and Graybiel 1971]. The test image was rotated until the landmarks on the eye were aligned. Precision for this and all other 35mm film methods was obtained by comparing different measurements of the same image and taking the standard deviation. This assumes that the measurements are samples from an unbiased estimator of the angle of rotation. While this appears reasonable, it is not true in the presence of head rotation relative to the camera unless this rotation is subtracted from eye movement. Miller solved this problem by custom making helmets for his subjects to eliminate head movement. He reported an average resolution of 5.3 minutes of arc, with variations between 5' and 15'. In a more recent version of this procedure, Diamond et al. [1979,1988] have reported a precision of 15-30' or 6', depending on the equipment.

A second method [Lichtenberg 1979; Arrott 1982] was developed at MIT. Given two points on a rotating target, the angle formed between the line connecting these two points at two successive times t_1 and t_2 is equal to the amount that the disk has rotated between t_1 and t_2 (figure 1.4). Thus, if two points are chosen on the iris, rotation can be tracked. In this method, two points are taken on the iris, and two on a biteboard that the subject clamps down on and which is thus fixed with respect to his head. This has been referred to as a "dumbbell" method as the two points joined by a line resemble dumbbell weights. For each test image, two angles are calculated: the angle of the biteboard with respect to the camera horizontal, and the angle of the iral line with respect to the camera horizontal; the difference gives the angle of the iral line with respect to the head, and permits comparison with a reference image taken earlier. As head rotation is measured with this method, the helmet need not fit very tightly and thus does not interfere with the experiment. To acquire the points, the film is projected on to a scanning table using a Hermes Senior Film Analyzer and landmark locations are digitized. This method is accurate but very workload intensive, as it takes the human operator at least 30 seconds to analyze one eye. Also, as many eyes do not present clear natural landmarks, it is usually necessary for the subject to wear contact lenses with artificial landmarks. An average accuracy of 12' was reported for this method with a range of 1-30', with certain "wild'" points outside this range [Lichtenberg 1979].

For all 35mm methods, analysis time is on the order of 30 seconds to one minute per frame. This has led experimenters to use motor driven cameras with a rate of less than five pictures per second. Massoumnia [1983] reports

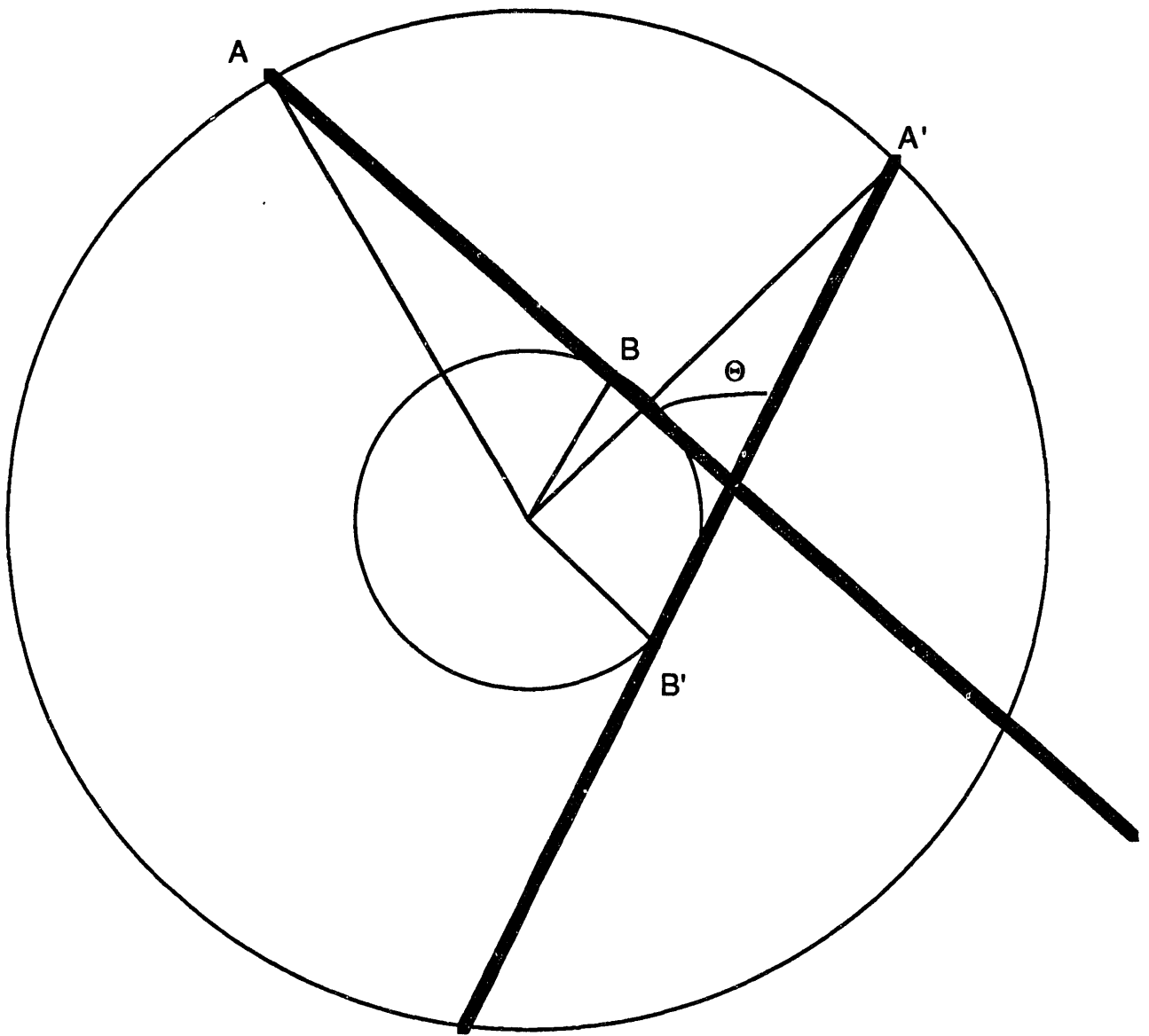


Figure 1.4 : The Dumbbell Method.

At time t_1 points A, and B are on concentric circles of different sizes.. Each circle rotates by an angle θ between time t_1 and time t_2 ; Point A is rotated to A', point B to B'. The angle between lines AB and A'B' is θ .

that horizontal and vertical eye movement fast phases contain significant energy up to approximately 10Hz, with 95% of all energy in the position signal being contained in the 0-10Hz band. This requires a sampling rate of 20Hz or better to avoid aliasing. Thus, while 35mm film may be able to detect low frequency torsional components, it can not be used to analyze nystagmus.

2/ Search Coil Methods

A second set of methods involves wearing contact lenses with wire search coils imbedded in them while sitting within two large field coil systems. Two alternating magnetic fields induce voltages on the search coils that are functions of eye movement. Amplitude or phase information is used to calculate rotation. Robinson [1963] reported a resolution of better than 1' for horizontal and vertical eye movements using amplitude detection. Collewijn applied phase detection to detect head movements in rabbits, and later studied human ocular torsion [Collewijn 1977, Collewijn et al. 1985]. No clear definition of precision was obtained for this method, as no control data comparing this and another method simultaneously yet exists for ocular torsion.

Search coil methods offer continuous information, and can be used when the subject's eyes are closed. Precision is likely very good, but no conclusive measurements have yet been performed. These methods are also somewhat inconvenient. Robinson used a suction device to hold the lens to the eye; this causes discomfort and even some corneal deformation. Although Collewijn used a slightly different method to attach his coils, embedding

them in rings made of silicone rubber, and did not need a suction device, the eye was anaesthetized. Also, blunt forceps were needed to remove the rings. For both methods, cumbersome field coils are placed around the subject, creating a magnetic field that is sometimes hard to control.

Thus while search coil techniques present many advantages and may be extremely precise, they do not lend themselves to all operating environments, and questions exist as to interference between the measurement apparatus and the experiment response.

3/ Video Based Methods

The first video based system for the calculation of ocular torsion was created at the Man-Vehicle lab at MIT [Edelman 1979]. Here, a hair was sandwiched between two soft contact lenses and placed on the eye. A light was shone onto the pupil so that the return signal was a black line on a white background. Precision was on the order of 6' for synthetic data. A rough estimate of 6-18' was proposed for real data [Young et al. 1981].

In 1982 Hatamian proposed a similar non invasive method. Here the center of the pupil was first located, then a circular region around it was sampled. This region was crosscorrelated in one dimension with a similar region acquired from another image, to give an angle of rotation. Accuracy was highly dependent on pupil center detection; with synthetic data, maximum error was evaluated at above .12 degrees; a photographic method similar to the one used by Miller [1962] was applied to real data and a discrepancy of less than .23 degrees was found between the two methods

[Hatamian 1982, Hatamian et al. 1983]. No data is available on the speed of this system as it was never implemented fully.

In Pittsburgh, Wall also developed a similar prototype method using one dimensional circular correlation, and obtained a precision of 20-30'. This system was however never used to process experimental data. (Personal communication, 1988).

EMIRAT, the only operational automatic video system for measurement of ocular torsion was described by Vieville and Masse [1987]. This system uses custom made hardware: a solid element camera is hardwired to a digital frame grabber which digitizes pixels according to 14 levels of grey; the data is then sent to a calculator for analysis. There is no data storage device, as analysis is done at close to real time. The 48ms analysis time enables the system to perform at a 20Hz sampling rate. Analysis is also based on one dimensional circular correlation, after averaging values along a 360 degree ring on the iris. This means that the whole eye must remain open throughout the experiment: a droopy eyelid overlapping on the analysis ring would corrupt the measurement.

For data to be analyzable, eye center computation has to be correct, and eye picture quality has to be sufficient to allow satisfactory recognition of iral marks. It has been observed empirically on real data that performance does not vary regularly with picture quality: data is either analyzable, or not analyzable at all. Some subjects were determined to have eyes that could not be analyzed. An "overall precision" on analyzable data of 0.5 degrees was reported. When a camera was rotated in a range of ± 40 degrees, while filming an eye picture. "sensitivity" was evaluated at 0.4 degrees, and "linearity" at 0.3 degrees. Measurement was

also compared with manual analysis of video data. The standard deviation of the difference between these two types of measurement was .37 degrees. A goniometer with a reported precision of better than 0.2 degrees measured head position. This system has been used to analyze large angles of ocular torsion.

At MIT, Parker used a slightly different correlation method to analyze several digitally rotated video images [Parker 1983, Parker et al. 1985]. This involved using Landsat image analysis techniques to track four areas : two on the eye and two on the biteboard, using two dimensional correlation techniques. The translations of the four points was then used to calculate eye rotation using Lichtenberg's dumbbell method. Parker applied this technique using both circular and rectilinear coordinate systems. Although the latter system is limited in scope to movements of approximately -5 to +5 degrees it is less computationally demanding than the former method. Nagashima implemented the rectilinear coordinate algorithm to create a program capable of comparing two images using a PDP11-34 and a VAX-11/ 780 with a floating point accelerator. This took 75 minutes on the RT-11 and 30 seconds on the VAX, but did not compensate for head rotation. For a real eye with artificial landmarks that was computationally rotated between -5 and 5 degrees, he reported a precision of better than .25 degrees. However neither Parker nor Nagashima's systems were used to analyze actual experimental data.

C/ Choice of an Integrated Video System - Performance Characteristics

1/ Advantages of Video Analysis Methods

A summary of the different methods for calculation of rotation can be found in figure 1.5. Clearly, video based systems are very appealing as they avoid the high workload and low sampling rate problems associated with manual 35mm film scanning, and the interference and invasive nature of the search coil mechanisms. They also allow enhanced quality control as the general experimental setup can be monitored during analysis, permitting detection of such problems as lens slippage or closing of eyelids.

Given the existence of prototype algorithms for the comparison of two images, it was deemed appropriate to create a practical system based on one of the above video methods. This involved creating a system capable of reading data from a videotape, feeding it to a computer and calculating an angle of rotation.

2/ Performance Parameters for Video Systems

The parameters determining system performance can be divided into two types depending on whether or not they depend on the specific video analysis method used.

If there is a loss of information in the transfer from the camera to the computer, precision is degraded no matter which method is used. Thus it is important to characterize the spatial and temporal frequencies of the incoming signal, and compare it to the sampling rate and spatial resolution of the recorded input signal. If, as discussed above, torsional eye movements are assumed to be in the 0-10Hz range, the 30Hz video sampling rate is thus greater than twice the largest input frequency, and no time

OT Measurement Technique	Sampling Rate	Experiment Verification	Status	Time/Frame	Precision	Advantages/Disadvantages
<u>35mm Methods</u> Prism methods (Miller, Diamond) Dumbbell Technique (Lichtenberg, Arrott)	Usually Below 5Hz	Yes	In operation and validated on real data	30s	5.3', 6' ³ 12' ⁴	- Workload intensive - Aliasing
<u>Search Coil Methods</u> Scleral Contact Lens (Robinson) <u>Flexible Ring</u> (Collewinj)	No Limit	No	In operation and validated on real data	Real Time (RT)	1' ⁵	- Eye closed operation - Uncomfortable, requires anaesthesia - Lens slippage causes error in data
<u>Video Methods</u> <u>Bright Pupil</u> (Edelman) <u>Circular Correlation</u> Hatamian Wall Emirat	30Hz 30Hz	No Yes No No	Prototype Prototype Prototype In operation	not RT N/A ¹ not RT 48ms	6' ⁵ >10' 20-30' 30'	- First video system - Extremely sensitive to eye center location
<u>Dumbbell Technique</u> (Parker, Nagashima)	30Hz	Yes	No	2min ²	> 6' ⁷	Limited to + or - 5 deg

1 : System not implemented. 2 : No calculation of head rotation. 3 : average error. 4 : average error; range 1-30' with occasional outliers. 5 : Value for horizontal/vertical movement. No results for torsion. 6 : synthetic data. 7: synthetic rotation of real eye images with no head rotation.

Figure 1.5 Comparison of Ocular Torsion Measurement Methods

related information is lost by sampling. In the same way, if the largest spatial frequency is less than half the pixel size, no spatial information is lost. This can be confirmed by looking at the frequency spectrum of the image: the spectrum must be strictly contained inside the available bandwidth. This is discussed in Chapter II when the Fourier Transform of two dimensional images is considered. It should be pointed out that, as the signal in the spatial domain is of finite dimension, it can not be bandlimited in frequency; it is therefore necessary to choose image sizes that are large enough to avoid spectral leakage and consequent aliasing.

Noise in the input signal and the frame grabber A/D quantization effects constitute other sources of error. Although they represent major sources of precision degradation, none of the above video system descriptions included an analysis of the error they introduced.

Beyond the error due to video image acquisition, additional loss of accuracy is related to violations of the assumptions inherent in each method.

For the circular correlation technique pupil center location was reported to be the most important source of error. Interpolation and cross correlation precision undoubtedly also contributed however.

With the dumbbell technique on video, errors can be seen as coming from three major sources :

1/ noise in the correlation function (insufficient sharpness of image). This can be expected to be lower than in the circular correlation technique, as more points are processed (two dimensional instead of one dimensional analysis);

2/ sub pixel displacements : the correlation function can only calculate integer translations; this problem was partially solved by interpolation; it also appears in the circular correlation method;

3/ difference between rotation and translation : this method detects translations of small areas, when in fact they have rotated; for angles of + or - 5 degrees, this error is considered to be negligible [Parker 1983];

3/ Choice of Video Analysis Method

Among the video methods, Parker's had been shown to be effective on the data set recorded by MIT investigators. Also, with the rectilinear dumbbell approach, sources of error can be pinpointed precisely and independently.

It was therefore decided to use this approach as a basis for the TOMAS system.

II/ Automatic Calculation of Image Rotation Using the Correlation Technique

This chapter examines the image processing techniques used to determine image rotation, and discusses their implementation in the TOMAS system.

A/ Introduction

Although the basic method developed here - i.e. an application of the Dumbbell method to video data using correlation based landmark tracking - is applicable to naked eye images, the system was optimized for eye images with clear artificial landmarks, such as the contact lens markings used on the SL-1, D-1, and IML-1 missions. Figure 2.1 shows a typical test image containing an eye with its contact lens, and the biteboard with its fiducial marks that serve to determine head movement.

Input images were first digitized along a rectilinear grid with values at each node defined by intensity. They were next enhanced using the techniques described in part C of this Chapter. Unambiguous markings from the lenses served as the "weights" at the ends of the dumbbell on the eye. However to track these points, the computer had to be able to recognize them. Thus instead of points, small regions were chosen; these shall be referred to as "Regions of Interest". They were defined by the operator on a reference image at the beginning of each analysis run. The algorithm then located each of these regions on the successive test images using the



Figure 2.1 : Video image from D-1 Spacelab mission. At left is subject performing experiment (body sway is measured). At right the eye with its distinctive contact lens markings. Note also the biteboard fiducial marks under the eye.

rectilinear crosscorrelation and interpolation techniques described in part D. After the Regions of Interest had been located on the test image, they defined the dumbbell for a pair of landmarks, and eye angle could be calculated (Part E). Figure (2.2) illustrates the dumbbell method applied to video data, and figure (2.3) summarizes the steps necessary to calculate eye rotation with this technique. Implementation is discussed in part F, and accuracy and precision are examined in part G.

Before expanding on the topics raised here, it is of interest to examine the underlying mathematical tool used for most of image analysis in this thesis : the two dimensional Fourier Transform.

B/ The Two Dimensional Fast Fourier Transform

1/Convolution and Correlation

A discrete two dimensional image can be mapped into a function $f(i,j)$ where $f(i,j)$ takes on positive values corresponding to the intensity of the image at (i,j) . Let $f(i,j)$ denote such a two dimensional image, and let $h'(i,j)$ denote a linear filter. Then $g(i,j) = f(i,j) * h'(i,j)$ is the convolution of f and h' and represents the output of filter h' given input f . This convolution is defined as follows :

$$g(i,j) = \sum_n \sum_m f(m,n) h'(i-m,j-m)$$

The correlation $g(i,j)$ of two images, $f(i,j)$ and $h(i,j)$ is defined as :

$$g(i,j) = \sum_n \sum_m f(n,m) h(n-i,m-j)$$

$g(i,j)$ can simply be considered to be the convolution of $f(i,j)$ with a filter $h'(i,j)$ where $h(i,j) = h'(-i,-j)$ for all points (i,j) .

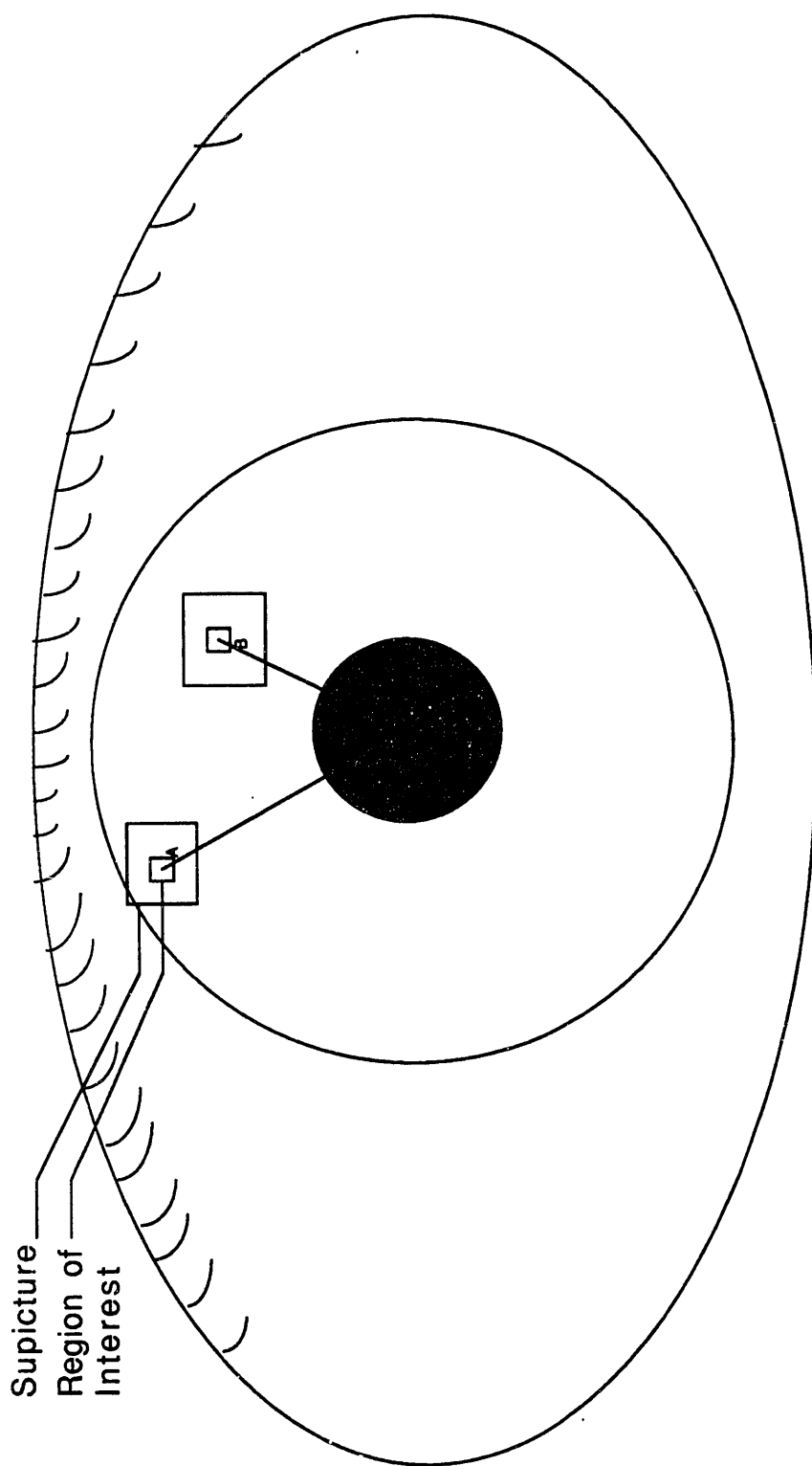


Figure 2.2 : Principle Behind Dumbbell Method. for Video

Instead of acquiring two reference points, two details or Regions of Interest are acquired. The larger squares denote subpictures- i.e. the regions in the test pictures that the original Regions of Interest will be correlated with.

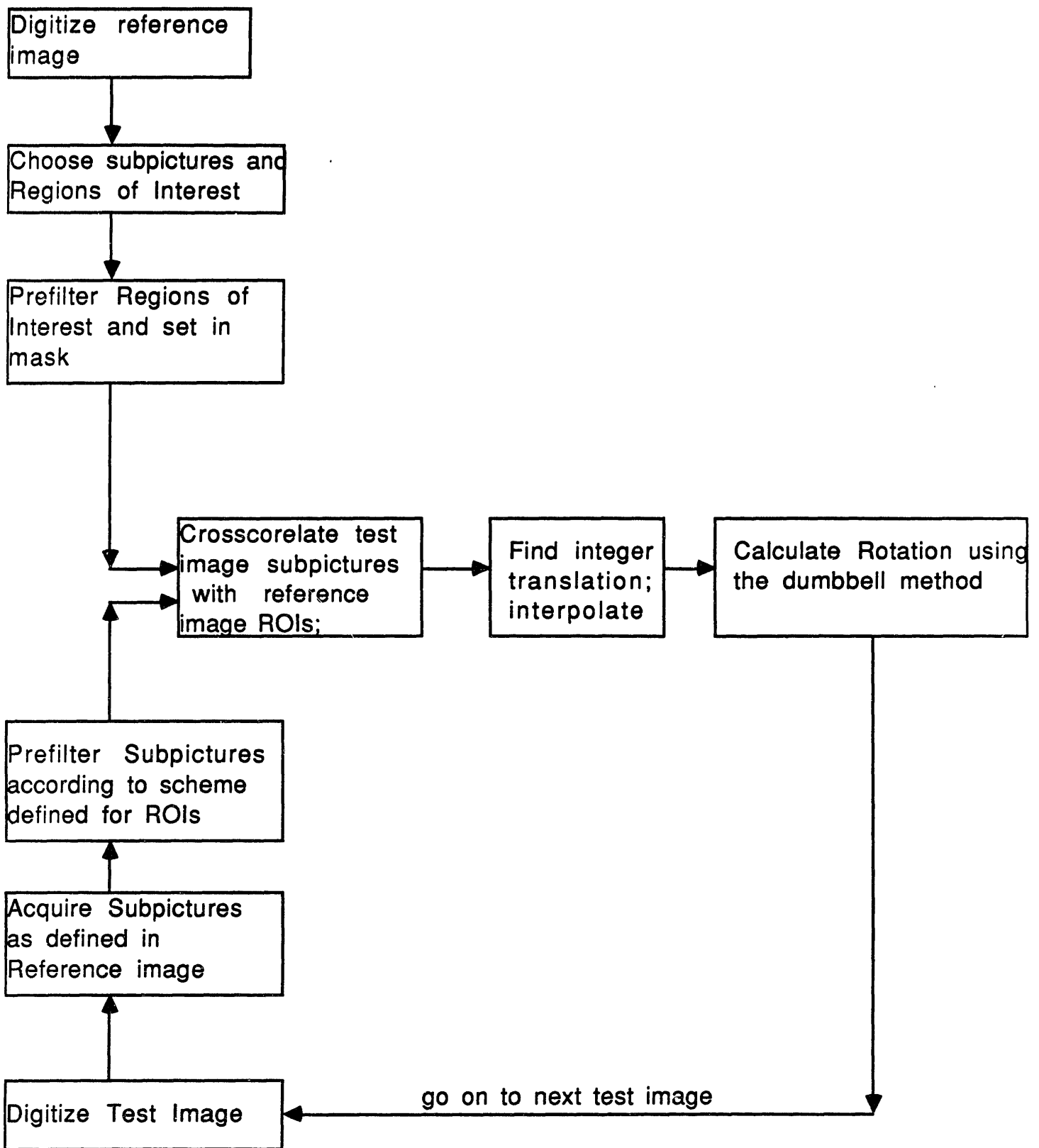


Figure 2.3 : Calculation of eye rotation using the dumbbell method on video data

For an $N \times N$ image, and an $M \times M$ filter the number of operations necessary to perform their convolution is therefore on the order of $N^2 \times M^2$. For large N and M this operation imposes a large computational burden; as a result Fourier Transform methods are often used.

2/ Computational Advantages of the Fourier Transform

The Discrete Fourier Transform of a one dimensional function, $f(x)$ is defined as :

$$F(u) = FT(f(i)) = 1/N \sum_{i=0}^{N-1} f(i) \exp[-j2\pi ui/N]$$

The two dimensional transform can be defined by extension as :

$$F(u,v) = FT(f(i,j)) = 1/N \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f(i,j) \exp[-j2\pi(ui+vj)/N]$$

This is a bilinear transform. Its inverse is given by :

$$f(i,j) = FT^{-1}(F(u,v)) = 1/N \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} F(u,v) \exp(j2\pi(ui+vj)/N)$$

Using the above notation, if $f(i,j)$ and $g(i,j)$ are padded with zeroes to a size $(A \times A)$ where $A > M+N-2$, we have :

$$G(u,v) = F(u,v) * H'(u,v)$$

or

$$g(i,j) = FT^{-1}\{F(u,v) H'(u,v)\} = FT^{-1}\{ FT[f(i,j)] FT[h'(i,j)] \}$$

Thus the Fourier Transform can be used to calculate discrete convolutions.

Also, we have :

$$\begin{aligned} FT(h'(i,j)) &= FT(h(-i,-j)) \\ &= 1/N \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} h(-i,-j) \exp[-j2\pi(ui+vj)/N] \\ &= \{1/N \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} h(i,j) \exp[-j2\pi(ui+vj)/N]\}^* \\ &= FT^*(h(i,j)) \end{aligned}$$

This tells us how to get the Fourier transform of the filter we use to do a crosscorrelation.

Further, if the value of N is chosen such that it is a power of 2 then the two dimensional Fast Fourier Transform (FFT) can be used. This involves invoking N one dimensional transforms along the columns of $f(i,j)$, followed by N one dimensional transforms along the rows of $f(i,j)$. As one FFT demands approximately $N \cdot \log N$ operations, the whole procedure demands on the order of $2 \cdot N^2 \cdot \log N$ operations, rather than the N^4 operations necessitated by the straightforward filtering approach.

3/ Sampling of Continuous Signals

Thus far the discussion has centered on discrete two dimensional signals; however as the input is an analog video signal, sampling must be performed. This is done with a digitizer, and presents no particular problems as long as Nyquist's theorem is respected and amplitude resolution is adequate. Nyquist's theorem states that no information is lost when sampling a signal as long as the sampling rate is greater than twice the largest frequency present in the input signal.

4/ Representation and Packaging

As many real life image spectra decrease rapidly as a function of increasing frequency, it is often useful to plot an altered log magnitude function :

$$D(u,v) = \begin{cases} 0 & \text{if } |F(u,v)| < 1 \\ \log(|F(u,v)|) & \text{otherwise} \end{cases}$$

Finally, as all the images used in this study contained only real elements, it is of interest to note the packaging technique used by the system to store image transforms.

For an image containing only real elements, if we let x^* represent the complex conjugate of x , we have :

$$\begin{aligned} F(u,v) &= 1/N \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} f(x,y) \exp[-j2\pi(ux+vy)/N] \\ &= \{1/N \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} f(x,y) \exp[j2\pi(ux+vy)/N]\}^* \\ &= F^*(-u,-v) \end{aligned}$$

Also, $F(u,v)$ is periodic in u and v , each with period N . ; i.e:

$$F(u,v) = F(u+N,v) = F(u,v+N)$$

This results directly from the definition of the transform.

Thus if the input contained N^2 real points, the output contains N^2 complex points, of which $3*N^2/4 - N$ are redundant. Also :

$$\begin{aligned} F(v,0) &= F^*(-v,0) = F^*(-v+N,0) \quad v \text{ in } [0..N] ; \\ F(v,N/2) &= F^*(-v,-N/2) = F^*(-v+N,N/2) \quad v \text{ in } [0..n] ; \\ F(0,0) &= F^*(0,0) ; \\ F(N/2,N/2) &= F^*(N/2,N/2) ; \end{aligned}$$

And $F(u,v)$ can be retrieved from N^2 real points, as shown in figure (2.4). When two image transforms are multiplied, this leads to a distortion of the zero frequency value. As the Nyquist frequency is generally negligible this usually has little impact on further processing.

$R(0,0)$	$R(0,N/2)$	$R(0,1)$	$I(0,1) \dots \dots \dots$		$R(0,N/2-1)$	$I(0,N/2-1)$
$R(N/2,0)$	$R(N/2,N/2)$	$R(1,1)$	$I(1,1)$		$R(1,N/2-1)$	$I(1,N/2-1)$
etc	etc	etc	etc		etc	etc
		$R(N/2,1)$	$I(N/2,1)$	$\dots \dots \dots$		
		$R(N/2+1,1)$	$I(N/2+1,1)$	$\dots \dots \dots$		
$R(N/2-1,0)$	$R(N/2-1,N/2)$	$R(N-2,1)$	$I(N-2,1)$	$\dots \dots \dots$	$R(N-2,N/2-1)$	$I(N-2,N/2-1)$
$I(N/2-1,0)$	$I(N/2-1,N/2)$	$R(N-1,1)$	$I(N-1,1)$	$\dots \dots \dots$	$R(N-1,N/2-1)$	$I(N-1,N/2-1)$

Figure 2.4 : Packaging of a real FFT so as to put the information from $N \times N$ complex points into $N \times N$ real points. Note that even with this method the points in the Bold square are redundant.

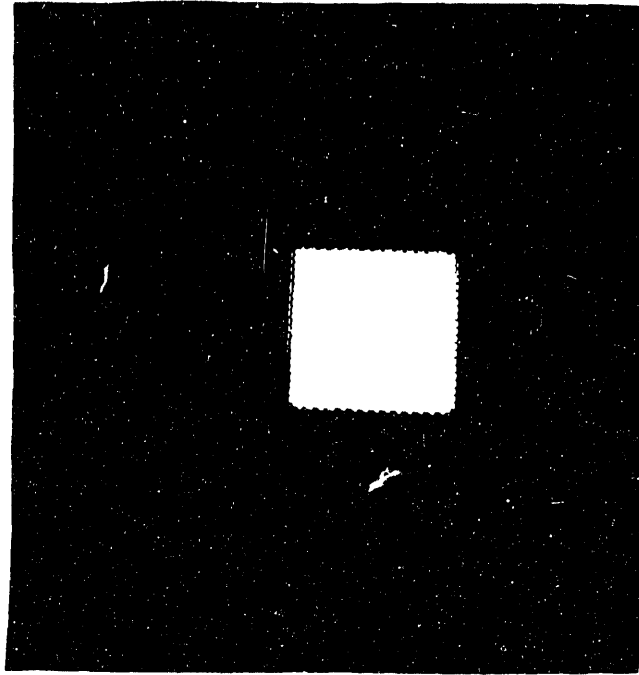
Figure (2.5) shows an image, the positive side of its frequency spectrum, and the positive side of the corrected log of the spectrum.

Given the above knowledge of the Fast Fourier transform in this particular implementation, it is now possible to go on to image analysis per se.

C/ Preliminary Filtering

Although the correlation technique should theoretically work if applied directly to the natural landmarks of an eye image, lack of contrast, changes in lighting and reflections of different objects on the eye create too much noise for this method to be used directly. For example, in the dome experiment, the subject looks inside a rotating drum which is covered with colored marks; the eye is also illuminated laterally. As a result reflections of the colored marks show very clearly on the eye. Also, as iral features are three dimensional, they have shadows when illuminated laterally, and the shape of the features depends on eye position and lighting. Two steps were taken to clarify the image. First the subjects wore contact lenses with test patterns on them, and secondly the image was sharpened through preliminary filtering. Both linear and non linear filtering techniques were tried.

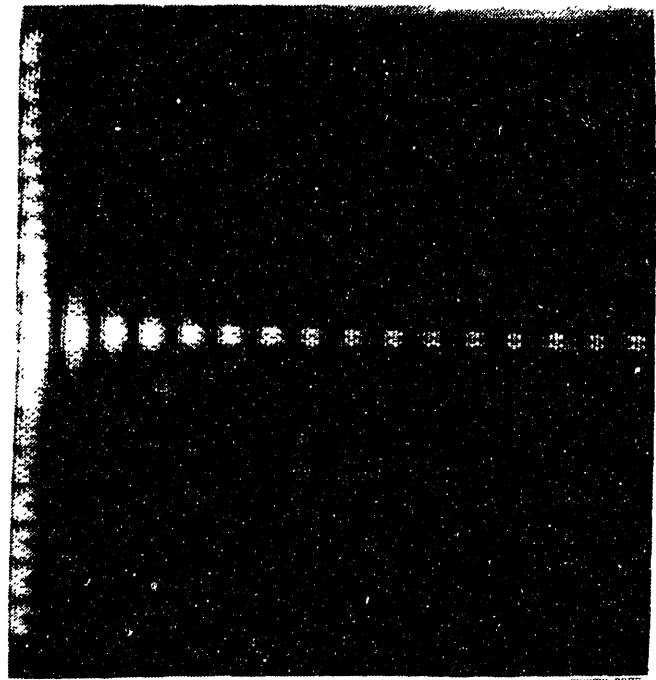
1/ Linear Filters



(a)



(b)



(c)

Figure 2.5 : A synthetic image, the positive side of its frequency spectrum, and the corrected log of the spectrum.

As explained above, the convolution of a linear filter with an image can be implemented using the two dimensional FFT. In previous studies [Parker 1983, Nagashima 1985], a "Mexican hat filter" was suggested. This is defined as the second derivative of a two dimensional gaussian function and is based on using a prefiltering approach similar to that of the human visual system. It is basically a band pass filter which has been shown empirically to pass those frequencies present in the eyeball [Parker 1983, Nagashima 1985]. The two dimensional Gaussian function is defined as :

$$h(i,j) = 1/(2\pi\sigma^2)^{1/2} * \exp(-r^2/2\sigma^2)$$

$$\text{where } r^2 = x^2 + y^2 .$$

This filter is circularly symmetric; its second derivative is :

$$h''(i,j) = 1/(2\pi\sigma^2)^{1/2} * (r/\sigma - 1) * \exp(-r^2/2\sigma^2)$$

In fact a slightly different filter was found to be most effective [Nagashima 1985] :

$$g(i,j) = 1/(2\pi\sigma^8)^{1/2} * (2-r/\sigma^2) * \exp(-r^2/2\sigma^2)$$

Figure 2.6 shows the gaussian and "Mexican Hat" filters with their frequency spectra, and figure 2.7 shows an artificial landmark on an eyeball before and after filtering with these filters. Clearly the image is enhanced.

As the object of the filtering operation was to sharpen the image, gradient filters also seemed appropriate [Gonzalez 1987]. Figure 2.8 shows an eye landmark after gradient filtering. Unfortunately, this approach was very noise sensitive, and was abandoned.



(a)



(b)



(c)



(d)

figure 2.6 : a: Gaussian filter; b: Positive side of spectrum of Gaussian
c: Filter used d: Spectrum.



(a)



(b)



(c)

Figure 2.7: a: original subpicture; b,c : effect of MHF filter chosen, effect of Gaussian filter.

(a)



(b)



Figure 2.8: Gradient Filter : a: original image; b: effect of gradient filter.

Although the Mexican Hat filter improved the quality of raw data, it did not always enhance the image sufficiently. This can be explained by an empirical frequency mapping of a subpicture.

Taking a one dimensional cut through a 128 pixel side of the image, one finds :

- one 10 pixel landmark;
- one 20-30 pixel pupil;
- irial details of two sorts :
 - large marks (5-10 pixels) every 10-128 pixels;
 - fine (1-2 pixel) lines with a periodicity on the order of 10-20 pixels;
- for the dome experiment, colored dot reflections (5 pixel every 10-15 pixel approximately).

Figure 2.9 shows the resulting approximate one-dimensional frequency spectrum for the image. Clearly the contact lens markings can not be enhanced without also enhancing some undesirable part of the image.

As a result of these empirical observations, and given the increase in processing time linked to linear filtering, non linear filtering was examined.

2/ Thresholding

Thresholding is much quicker than linear filtering, and it can be used to sharpen image contrasts. It simply involves setting a threshold value,

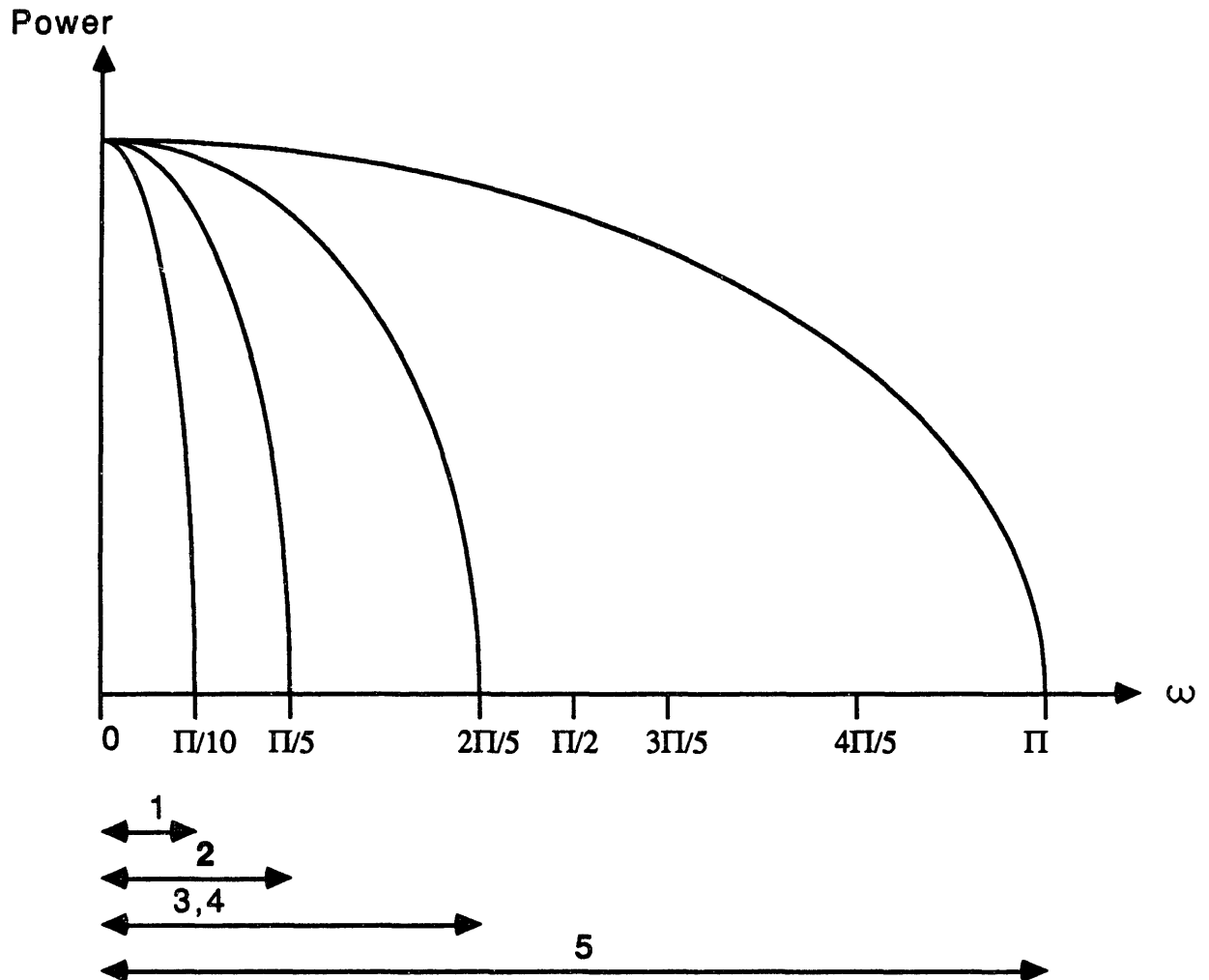


Figure 2.9 : Approximate expected frequency spectra of the different signals present in the eye for the chosen sampling rate (positive half of spectrum) :
 1 : Pupil; 2: Contact lens mark; 3: Large iral marks; 4: Dome reflections;
 5 : High frequency iral lines.

linked in some way to the value of the signal at the points corresponding to the lenses. As the lens markings were black, they corresponded to regions of minimal energy in a given subpicture. These landmarks were isolated by setting a threshold at some value of the range of the subpicture. If I_{min} denotes the minimum value of the subpicture, and I_{max} the maximum value, the following rule was used :

if $I(i,j) < I_{min} + (I_{max} - I_{min})/r$ then set $I(i,j) = 0$

where the user was prompted for a value of r .

An option was also provided allowing the user to invert the image so as to concentrate energy along the contact lens marks. Let M denote the maximum allowable intensity value. The Thresholding rule then became :

if $I(i,j) < M - I_{min} - (I_{max} - I_{min})/r$ then set $I(i,j) = 0$

Finally, the user could further simplify the image by setting to the maximum value all points that were not set to zero by thresholding. This led to a binary image. Figure 2.10 shows a landmark before and after each of the thresholding operations.

The different prefiltering options were used to prepare the raw data for correlation. Without them, analysis of real data would have been impossible.

D / Correlation Using the Two Dimensional Fast Fourier Transform

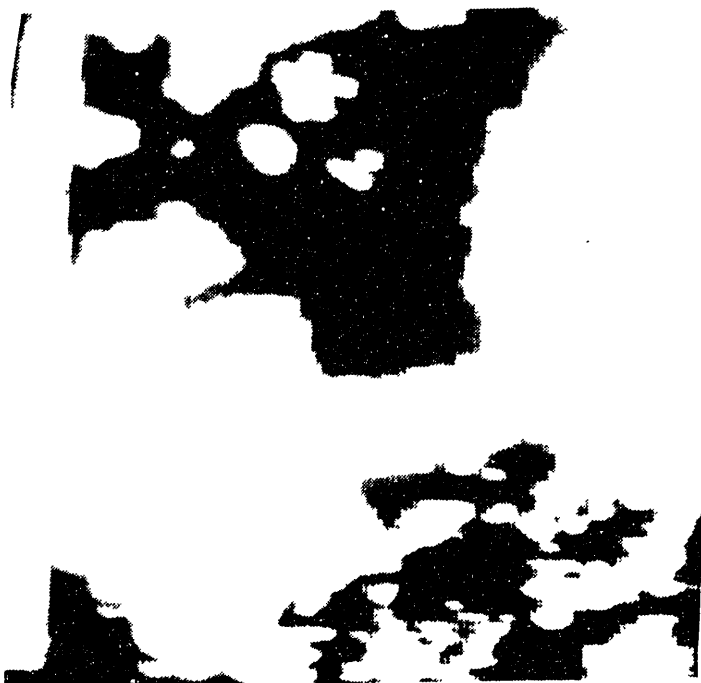
1/ Detection of Integer Translation



(a)



(b)



(c)



(d)

Figure 2.10 : Non Linear Filtering. a: original image; b: after thresholding (22% of range set to zero); c: original image after inverting and thresholding (78% of range set to zero). d: after inverting thresholding, and tranforming to binary image.

Correlation was chosen as the basis for automatic tracking of landmarks on the eye; this section examines this method in detail.

The correlation function, as defined in part B, is a measure of the similarity between two images; in a rectilinear coordinate system, it also measures the translation between them. if $f(m,n)$ and $h'(m,n)$ denote two images, we have :

$$g(i,j) = \sum_m \sum_n f(m,n) h'(m-i,n-j)$$

If (i',j') is found such that $g(i',j')$ maximizes $g(i,j)$ then $f(m,n)$ most resembles $h'(m,n)$ shifted by (i',j') . The greater the value of $g(i',j')$ the closer the second image matches a shifted version of the first. The type of mapping of the plane defines the type of shift that is detected: polar coordinate systems yield rotational shifts, while rectilinear systems track translation. Although eye rotation was being tracked, a rectilinear coordinate system was used for simplicity.

When dealing with finite images, the values on the edges of the correlation are skewed, as they either correspond to sums involving only a few terms, as in the case of linear correlation, or they include terms corresponding to the opposite side of the image, as in circular correlation. This problem was solved by choosing a small region of interest to track, and correlating it with a larger picture of which it was a part. For reasons of processing speed, the regions of interest from the original image were not correlated with whole test images but with 128×128 pixel sub images which could be safely expected to include the chosen landmarks. The 32×32 pixel region of interest was set in the center of a 128×128 square, $f(m,n)$, with all the surrounding points set to zero. The subpicture, $h(m,n)$, of the test

image was also 128x128 square. A circular crosscorrelation was then applied to these two images. This is a linear crosscorrelation of the infinite images obtained by considering each 128x128 picture to be one period of an infinite image. Given the structure of the landmark image, the resulting circular crosscorrelation function $g'(i,j)$ takes on identical values to the linear crosscorrelation function $g(i,j)$ for points not on the edge of the correlation function (i,j must be in $[-48,48]$).

To correctly measure similarity between the images, some allowance had to be made for the energy in the image; otherwise the crosscorrelation function would simply have taken on large values at those points where the underlying images had greatest intensity.

Energy is defined, for an image $f(m,n)$:

$$e(f) = (\sum_{m,n} f(m,n)^2)^{1/2}$$

For each point of the crosscorrelation function $g(i,j)$, the underlying energy from the reference landmark is simply the energy of the landmark, $e(f)$. For the test subpicture the corresponding energy is determined by convolving $e(f)$ with a mask filter $m(m,n)$ defined as :

$$m(m,n) = \begin{cases} 1 & \text{for } (m,n) \text{ in } [-16,16] \times [16,16] \\ 0 & \text{elsewhere} \end{cases}$$

Thus $e(i,j) = (m(m,n) * f^2(m,n))^{1/2}$, should be used to normalize the crosscorrelation between subpicture and landmark. Consider the function :

$$C(i,j) = g(i,j)/e(i,j)$$

or in one dimension :

$$C(i) = g(i)/e(i)$$

Assuming that the input region of interest is $h'(i)$ defined for i in $[0..n]$, and that the test function $f(i)$ is such that for some offset r :

$$\text{for } i = 1 \text{ to } n \quad f(r+i) = h'(i)$$

then

$$\begin{aligned} C(r) &= g(r)/e(r) \\ &= \sum_i h'(i) f(r+i) / ((\sum_i f^2(r+i))^{1/2}) \\ &= \sum_i h'^2(i) / ((\sum_i h'^2(i))^{1/2}) \\ &= ((\sum_i h'^2(i))^{1/2}) \end{aligned}$$

For others values of $C(j)$ we have :

$$\begin{aligned} C(j) &= g(j)/e(j) \\ &= (\sum_i h'(i) f(j+i)) / ((\sum_i f^2(j+i))^{1/2}) \end{aligned}$$

but, as for all j :

$$\sum_i h'(i) f(j+i) \leq [(\sum_i f^2(j+i)) * (\sum_i h'^2(i))]^{1/2}$$

$C(i)$ is maximized at offset r . Similarly in two dimensions, maximizing $C(i,j)$ yields the translation between two images.

A further enhancement of sharpness of the normalized crosscorrelation function is obtained if the region of interest is zero mean, both in the reference picture and in the test picture. For the reference picture, its mean value is simply subtracted from the region of interest. As it is impossible to determine the mean value of the region of interest in the test picture (this would imply knowledge of the location of the region of interest in the test picture), the same mean value that was calculated for the reference picture is subtracted from the test picture.

Figure 2.11 summarizes the operations that were needed to obtain landmark translation.

2/ Sub Pixel Translation

As described above, the correlation technique can detect integer values of image translation. However, with real data, images can be expected to move by some non integer amount. Assuming that the images are smooth enough, the algorithm can be expected to find the nearest integer neighbor to the actual value of translation. Thus using only integer translation introduces two types of error : first, the nearest neighbor to the maximum, and not the maximum itself is detected, and secondly, the risk of error due to noise increases as the crosscorrelation maximum will be less sharp at the nearest neighbor than at the true maximum. Both these problems can be lessened by using a denser digitizing grid, but this entails a large computational overhead, as the number of operations involved in each Fourier transform increases in a faster than linear manner with the number of pixels. A second solution is to increase the density of the grid, but maintain the number of points being processed by choosing smaller subpictures and areas of interest. This however limits the area over which the eye can move during the experiment. Another scheme, which does nothing to solve the signal to noise problem, but does solve the non integer aspect consists in interpolating the crosscorrelation function to find the true maximum. Assuming that there has been no aliasing in any of the images used, then the discrete crosscorrelation function can be assumed to be the sampled version of the continuous crosscorrelation between the original region of interest

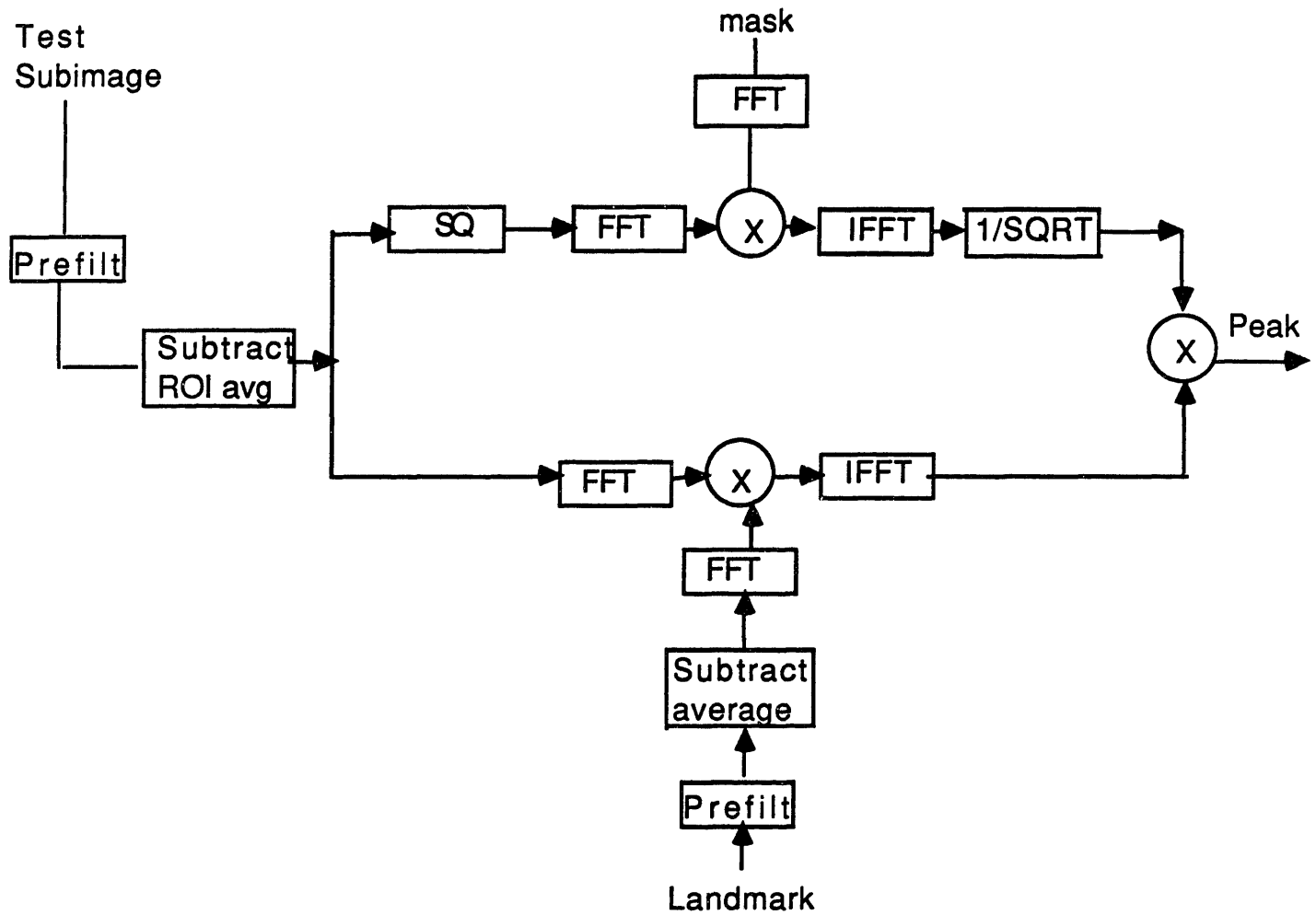


Figure 2.11 : Landmark Tracking Algorithm

FFT : Fast Fourier Transform. IFFT : Inverse Fast Fourier Transform. SQRT : square root. SQ : square.

and the test subpicture. As long as Nyquist's theorem is respected, the original function can be determined from the sampled version by applying a low pass filter. However this is not possible because the low pass filter would have infinite extent; as a result a number of different interpolating functions are used. In 1983 Parker studied different interpolating functions and determined the high resolution cubic spline functions to be most effective [Parker et al. 1983]. However in the peak search problem, it is necessary not only to interpolate the function but to determine which points should be calculated. As the cubic spline is only simple to use in a one dimensional context it was not well adapted to this problem. As a result, Parker used an iterative process using successive interpolations with parabolas defined by three aligned points. This involved finding the peak of the integer crosscorrelation and storing the values of the function at each of its eight neighbors. Then the parabolas passing through each set of three horizontal and vertically aligned points was defined. If $f(i,1)$, $f(i,2)$ and $f(i,3)$ are the values of the crosscorrelation at three horizontally aligned points, then the parabola passing through these points is defined as :

$$f_H^i(x) = a x^2 + b x + c$$

where :

$$a = 1/2 (f(i,1) - 2 * f(i,2) + f3)$$

$$b = 1/2 (f(i,3) - f(i,1))$$

$$c = f(i,2)$$

The parabolas passing through the vertically aligned points are defined in a similar fashion. The value of each "horizontal" parabola was calculated at points whose offset with respect to the center was a function of the values

of the "vertical" parabolas, and vice-versa. This process was repeated three times to find the approximate offset of the maximum.

E/ From Translation to Rotation

Given the translation of each of the landmarks, calculation of eye rotation is simply determined by using the dumbbell method. If correction is to be made for head rotation, two angles are first calculated : the angle, with respect to the horizontal defined by the camera, of the line containing the fiducial marks; and the angle with respect to the horizontal of the line containing the iral landmarks. Each angle is calculated as follows : let M and N denote the original landmarks, and (x_1, y_1) and (x_2, y_2) their respective coordinates; let M' and N' denote the rotated landmarks with coordinates (x_1', y_1') and (x_2', y_2') . Then

$$\theta = \arccos((x_2 - x_1) / ((x_2 - x_1)^2 + (y_2 - y_1)^2))$$

$$\text{and } \theta' = \arccos((x_2' - x_1') / ((x_2' - x_1')^2 + (y_2' - y_1')^2))$$

represent the angles of the two lines MN and M'N' with respect to the horizontal. Assuming that the points M and N have rotated around the center of the eye to reach M' and N' then the angle that the eye has rotated is

$$\psi = \theta' - \theta$$

For experiments where the head cannot be completely secured, and some small head rotation can occur, it is essential to measure the angle of the head with respect to the horizontal, and to track two landmarks on the biteboard. Let PQ define the line between these points on the original, and P'Q' the line on the test image, then ϕ and ϕ' are defined as above, and the angle of rotation of the eye is

$$\Psi = \theta' - \phi' - (\theta - \phi) = \theta' - \theta + \phi - \phi'$$

F/ Algorithm Implementation

1/ Data Analysis Pipeline

Figure 2.12 shows the TOMAS system configuration and figure 2.13 illustrates it in operation. It was decided to build a system using off the shelf hardware whenever possible; as a result, except for the circuits allowing the PC to control the remote controls which were designed by Mark Shelhamer of the MVL, all the hardware was bought commercially. The cost of the whole system in 1987 dollars was approximately \$25 000. The design for the remote control can be found in appendix 3.

Data was fed to the system from a SONY SL-HF-1000 VCR and digitized in a frame grabber. The VCR used standard consumer SuperBeta videotape. The system was managed by a PC'S LIMITED 286 series PC/AT personal computer with 1024K memory, and a 20286 math coprocessor. Most workload intensive operations were performed by a MERCURY COMPUTER SYSTEMS ZIP3216 array processor. The control loop was closed by a command line from the PC to the remote commander of the VCR. A television monitor was used for the human interface, and an EVERTZ MICROSYSTEMS LTD ECM 4000 time encoder served to determine exact position on the tape. The program was written in Microsoft C version 4.0 on the PC, and routines from the MERCURY COMPUTER SYSTEMS

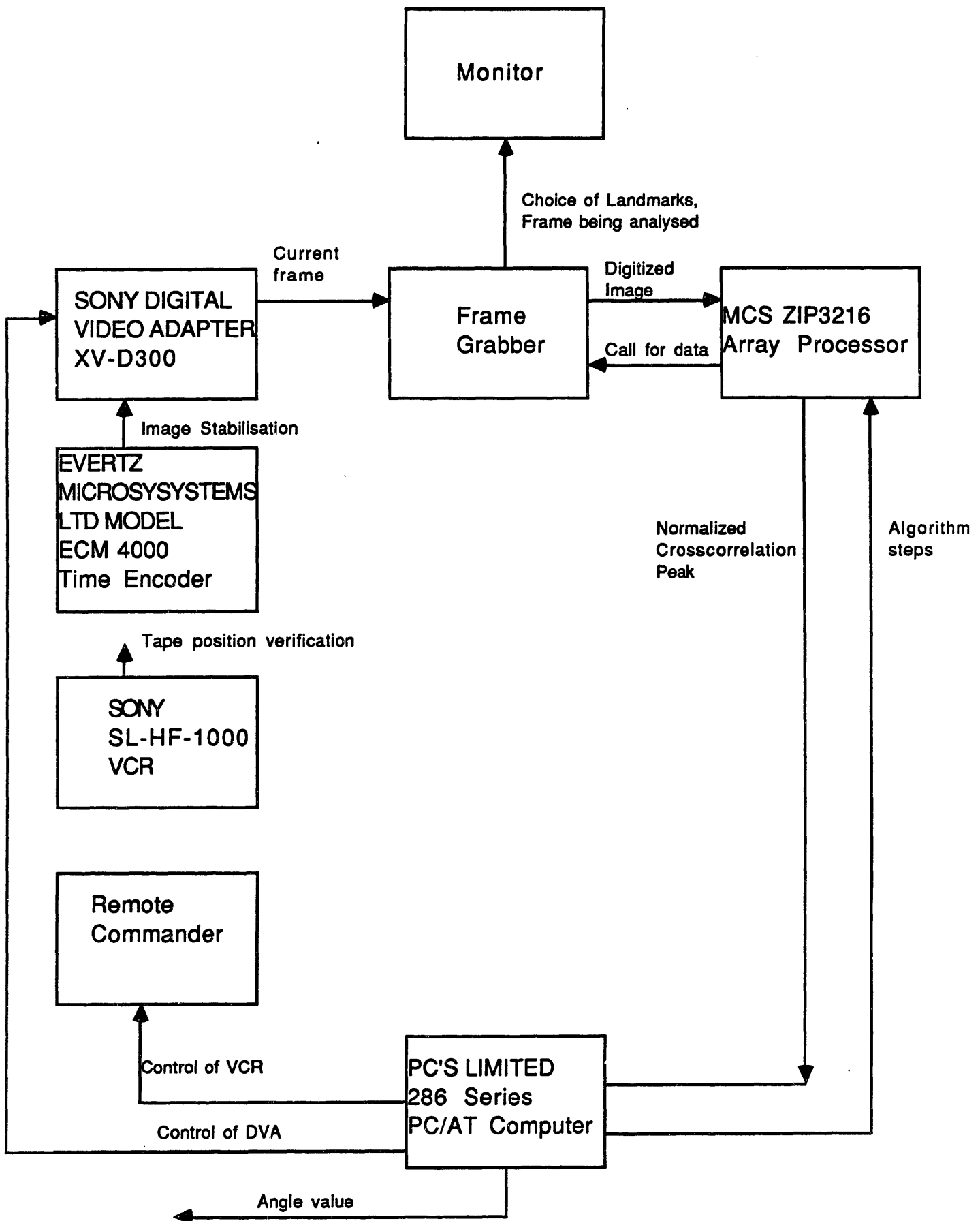


Figure 2.12 : Data Analysis System

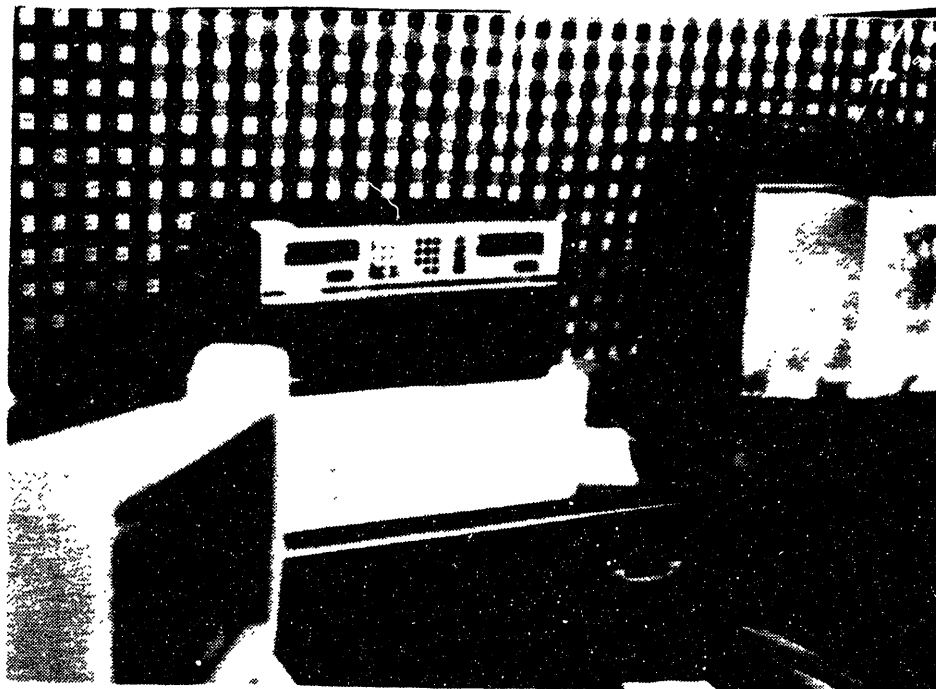


Figure 2.13: Workstation for Analysis of Torsion

Zip3216 Standard Algorithm Library and ZIP executive were called to command the array processor. The PC operating system was DOS version 3.0.

While the array processor performed rapid calculations, and was particularly useful for Fast Fourier Transform computation, much time was lost in image scaling between successive operations, and in data transfers between the PC and array processor. The frame grabber was incorporated in the array processor system, enabling rapid data transfers along the internal bus at a rate of 40 Mbytes per second. The video image was digitized in 480 horizontal and 640 vertical lines, with an 8 bit value for each pixel. Each 32 bit internal bus transfer contained two such values, corresponding to a transfer rate of 20 million points per second. The Sony VCR provided a still image containing both fields from the corresponding frame, and had a frame by frame advance capability. The image was a standard 525 line NTSC color signal. In pause mode the VCR tape angle of attack changes and the synch pulse cannot be read reliably; hence the VCR disregards the sync signal on the tape and adds a pulse signal from an internal oscillator. The resulting output synch pulse was not always in phase with the data, and the frame grabber had difficulty locking on to the image properly. This problem could be solved with a time base corrector, or by using a VCR with a dynamic tracking head that could read the original signal in pause mode. For this application, a SONY DIGITAL VIDEO ADAPTER XV-D300 was inserted between the VCR and frame grabber; as this component had a greater tolerance on input than the frame grabber, and a better internal oscillator than the VCR, it corrected the signal in much the same way that a Time Base Corrector would, and was much less expensive.

2/ The Landmark Tracking Algorithm

The input to this program is a video cassette containing images of eyes corresponding typically to 50 second experiment runs. At 30 frames per second this makes 1500 images per run.

The program begins by prompting the operator for the size of the subpictures. This can be 64x64, 96x96 or 128x128. All three cases lead to 128x128 image transforms, as, given the size of the landmark (32x32 pixels), a 64x64 pixel circular crosscorrelation would lead to aliasing of the correlation function at most offsets. The 128x128 input image size does lead to aliasing, but only on the edges of the picture. While there is no aliasing of the 96x96 or 64x64 images, they provide less range for the crosscorrelation function as this function is only a good measure of similarity for areas where there is complete overlap between the landmark and test image. Thus, given the identical cost involved for these image sizes, it is usually preferable to choose a 128x128 subpicture, unless factors such as location of the landmark with respect to the edges of the screen become important.

The options for preliminary filtering appear next, as the operator is asked whether to filter the images using the Mexican Hat filter, whether to invert them, and whether or not to threshold them. If thresholding is chosen he must also decide whether or not to use binary images.

The program also asks whether head rotation, as well as eye rotation should be calculated, and prompts for reference image position on the time encoder. If the operator has no knowledge of the pictures he next chooses to define the regions of interest with the video interface; otherwise he can

enter their positions analytically. Finally the program prompts for the number of comparisons to make in this run.

At this point the array processor is initialized, as is the video system, and the operator sets the tape at the reference image. If he chose to use the video interface, the operator selects the subpictures sequentially by moving a cursor made of horizontal lines delimiting the subpicture and region of interest containing the landmark. After each subpicture is selected, it is displayed, and a level for thresholding is chosen if that option was called initially. The image is then inverted if this option is in effect, and the filtered landmark is displayed. Mexican Hat filtering is performed if called for earlier. As preliminary filtering is now finished, the region of interest is isolated and its mean value is calculated and subtracted. Finally, the filtered picture is displayed. This procedure is repeated until each subpicture has been selected, and the corresponding filtered landmark approved by the operator. If any filtered sub picture is not considered satisfactory, it can be replaced, and the preliminary filtering repeated.

The program now calculates the Fast Fourier Transform of the input data, and is ready to start analyzing the run.

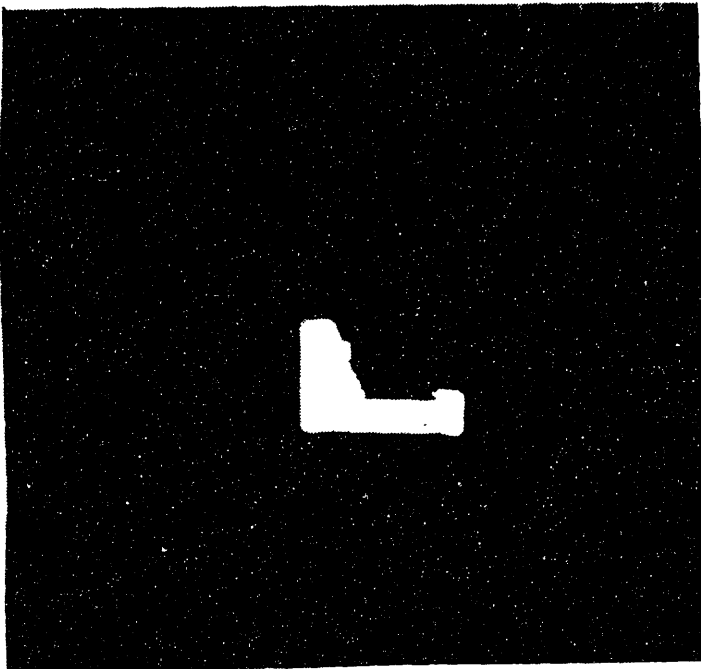
The system advances the video tape by one frame and captures the two (for eye rotation only) or four (if head rotation will be subtracted from eye rotation) subpictures on this test image. The exact same filtering operations that were performed on the initial images are now done on these; each filtered subpicture is then crosscorrelated with the corresponding reference landmark, according to the chart shown in figure 2.11. Figure 2.14 shows the different levels of processing for an eyeball landmark, and figure



(a)



(b)

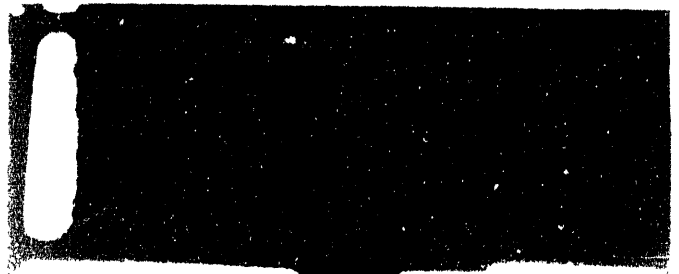
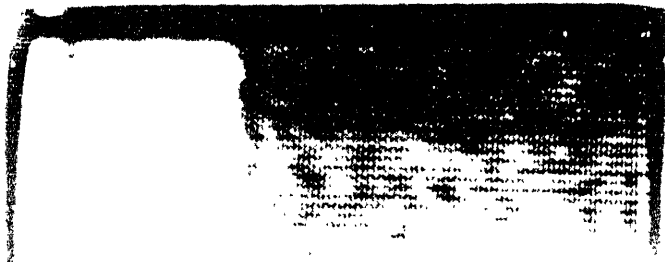


(c)



(d)

Figure 2.14 : Detection of Translation. a: original eye subpicture. b: after thresholding; c: Region of Interest d: Crosscorrelation function when image is compared to itself (Maximum is at center of image);



11

11

11



2.15 illustrates processing of a fiducial mark. Finding the peak of each crosscorrelation function yields the translation of each of the landmarks on the test image with respect to the original image. Eye rotation is then calculated and stored in a file, and the system goes on to the next image.

G/ Expected Accuracy

For the purposes of this thesis, bias will be defined as the average error in calculating an angle value. Precision will be defined as the distance of measurements from the average, and characterized by the standard deviation of the measurements. Accuracy is the sum of precision and bias. There are four major factors limiting the accuracy of this method.

Although the above correlation method provides a good test of similarity between images that have been translated with respect to each other, the landmark here has in fact rotated. Thus the method can only be used accurately for small angles where rotation can be approximated by translation, or with targets that are insensitive to rotation: circularly symmetric targets. Parker (1985) showed that this method worked well for angles between -5 and 5 degrees. Performance for angles greater than five degrees was studied experimentally in this thesis. A second source of error is due to the interpolation, as there is always some limited aliasing in the images, and the interpolation function is of finite extent. A third source of error comes from the video interface : this includes errors due to tape quality, poor synchronization between video instruments, and passing from an analog to a digital signal. Finally changes in the image during the run such

as variations in lighting have effects on the crosscorrelation function that can vary from annoying to catastrophic.

Thus, while a number of factors could be expected a priori to affect the precision of this algorithm, it appears to be sufficiently robust to function in a controlled environment.

III/ Results

A number of tests were run to document the performance of TOMAS, using different types of data, and different components of the system. The following conventions were used to classify data :

- Real data refers to video images of human eyes taken under experimental conditions; it is of two kinds:

- eyes with transparent soft contact lenses with black marks on the iris; this data was acquired in the MIT Canadian Spacelab experiments before the system was created, and the presented a particularly low signal to noise ratio: for the dome experiment for example the reflections of the dots on the eye were of approximately the same intensity as the contact lens marks themselves and effectively masked them. For the sled experiment data, poor lighting, and focusing problems degraded the images significantly.

- eyes covered with opaque hard contact lenses from the MVI experiment. Two white dots were drawn on a dark background.

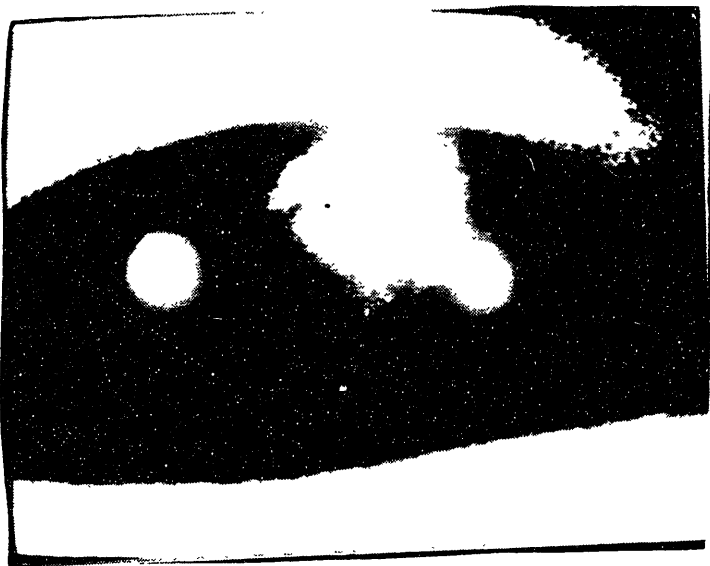
- Idealized data refers to video recordings of a flat paper target containing two black crosses on white backgrounds; clearly this is an idealized image as the signal to noise ratio is extremely high; however on processing, this data goes through the entire data analysis pipeline.

Figure 3.1 shows each of the data types that were videotaped.

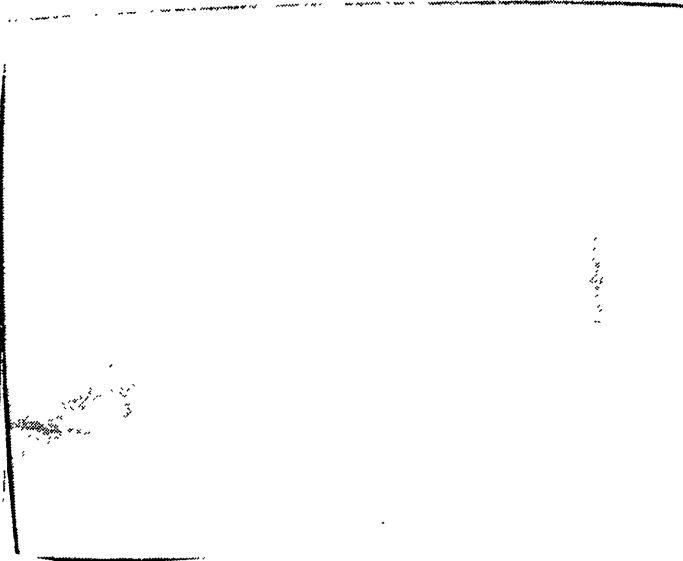
- Synthetic data refers to computer generated images; this data is not pushed through the entire analysis pipeline as it is not acquired through



(a)



(c)



(b)

Figure 3.1 : Real data used for validation. a: eye with transparent soft contact. b: eye covered with an opaque hard contact lens. c: "Idealized" data

the video interface.

- Stored real data refers to images that were previously acquired from videotape and stored in memory. When this data is compared to itself or to translated versions of itself, the video interface is again avoided.

A/ Method Validation

Before testing the full system, the method and its implementation were studied.

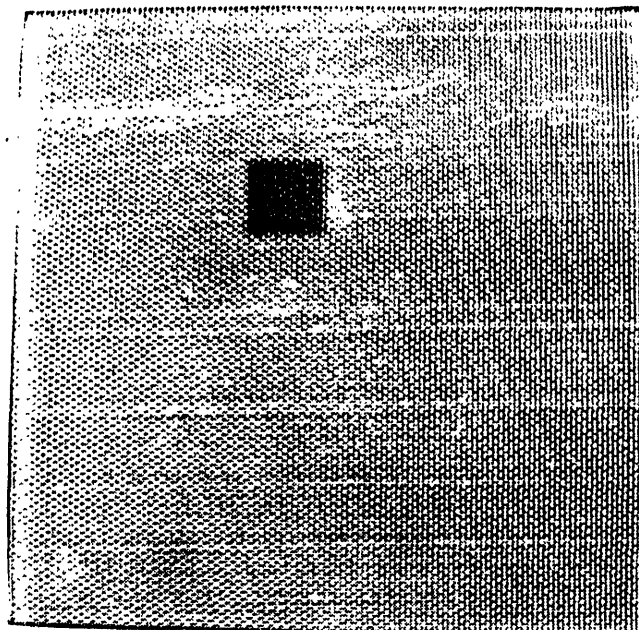
1/ Calculation of Translation for Synthetic Images

Synthetic images were translated synthetically by a whole number of pixels with respect to each other; the crosscorrelation program was used to determine translation. The image chosen is shown in figure 3.2; it consists of a plain square with a higher intensity subimage. The program determined translation accurately as long as the ratio of intensity between the two areas was above 12/11.

2/ Synthetic Translation of Stored Real Data

Real data was acquired and stored in memory; a landmark was then extracted from this and translated with respect to the original by a whole number of pixels. When the program was run on raw data, it did not find the

(a)



(b)

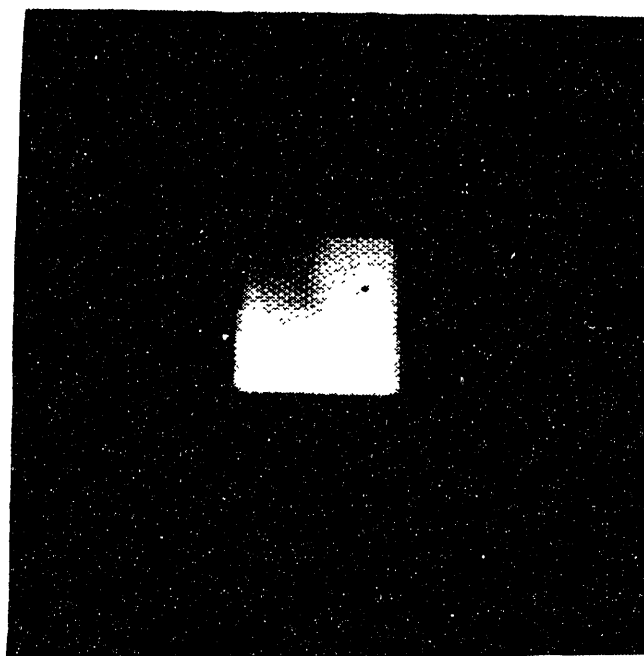


Figure 3.2 : Synthetic test image for calculation of translation. a: test image; b: test Region of Interest. Translation was calculated correctly for all translations as long as the ratio of intensities exceeded 12/11.

cross correlation peak. Three different prefiltering techniques were then employed :

- Threshold filtering;
- Mexican hat filtering;
- Gradient filtering;

These methods are described in chapter II (fig 2.6 -2.8).

Each of these methods proved sufficient to calculate translation exactly in the absence of a video interface (figure 3.3).

B/ Full System Tests

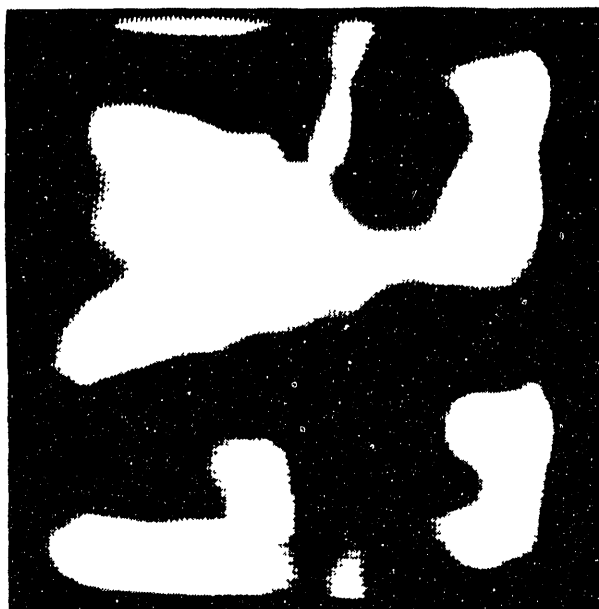
After the above partial system tests, full system validation was attempted.

1/ Idealized Inputs

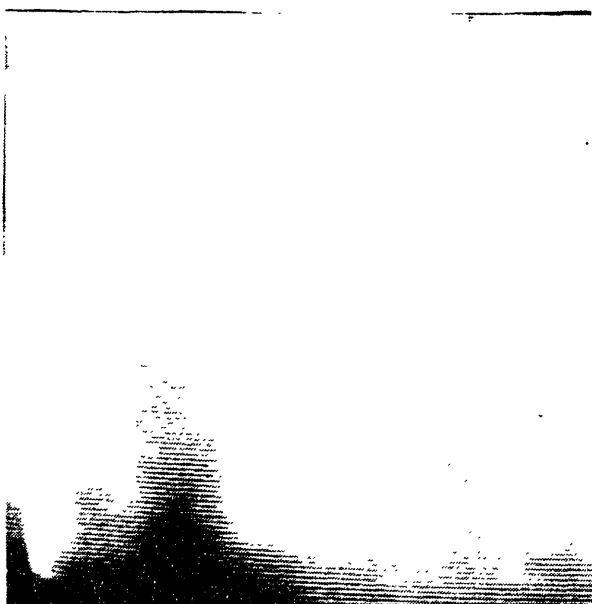
The idealized image was videotaped in six positions : zero to five degrees in increments of one degree. The error in construction of the video targets was on the order of .05 degrees. Figure 3.4 shows the output from the test, and table 3.1 examines its precision. The overall standard deviation of .078 degrees suggests high precision of the measurement. In terms of accuracy, all six average results are on the order of, or less than, the accuracy of construction of the experiment. Finally, when the same tape was analyzed a second time the results confirmed those documented



(a)



(b)



(c)



(d)

Figure 3.3 : Calculation of integer translation. Translation is "up" by 10 lines. a: original subpicture; b: crosscorrelation after Thresholding; c: after Mexican Hat filtering; d : after Gradient filtering. The correct maximum is detected in all three cases, but the crosscorrelation function is sharper in case a.

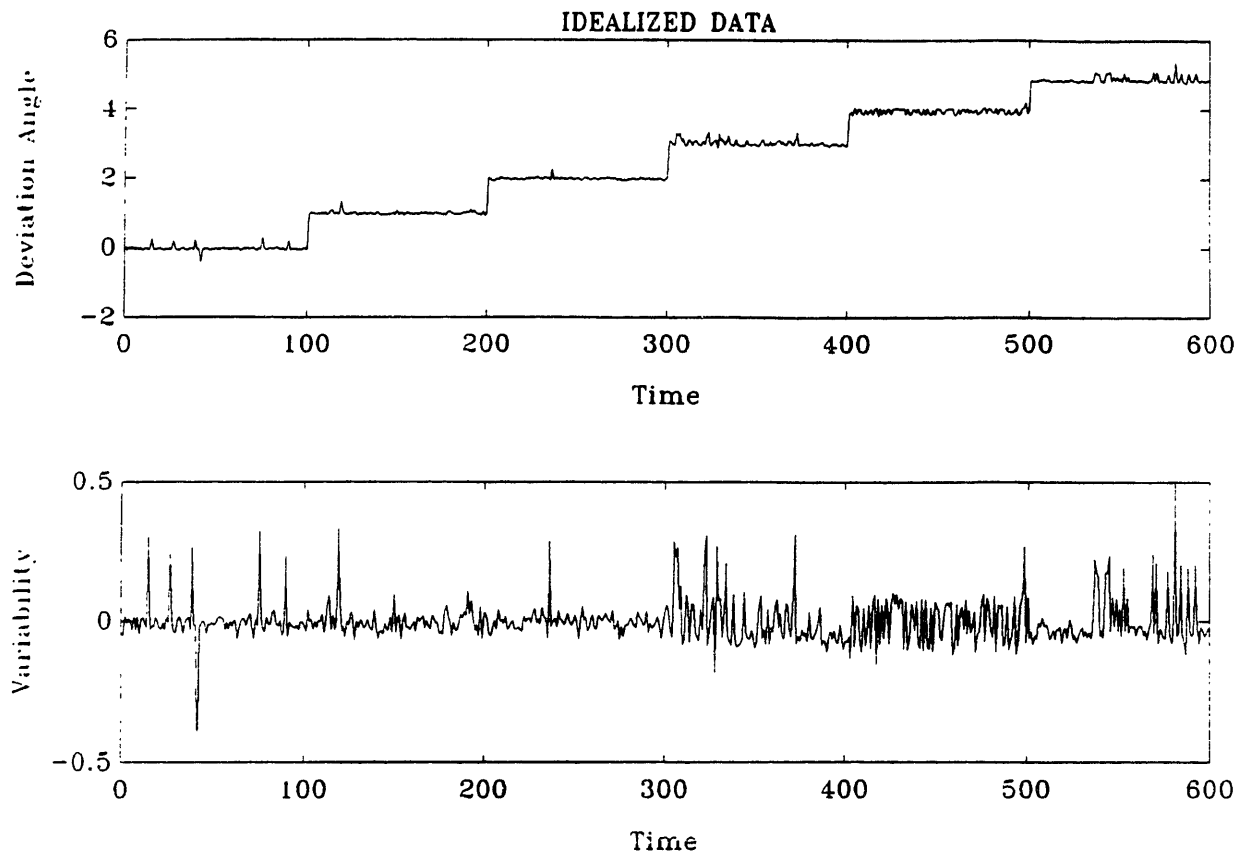


Figure 3.4 : Output from test of "ideal" data. Top: test output; bottom : difference between output value and mean value for each segment.

Angle	average value	Sample Variance	Sample Standard Deviation
0	-0.027	0.006	0.077
1	0.983	0.002	0.049
2	1.984	0.0015	0.039
3	3.024	0.010	0.100
4	3.949	0.0065	0.081
5	4.877	0.0097	0.0986
Overall		0.006	0.078

Table 3.1: Precision for "idealized" data. 100 points were taken for angles of 0,1,2 3,4, and 5 degrees.

above.

2/ Multiple image comparisons, Real Data - Transparent lens

Data from the same ten second run was analyzed twice. The raw data is shown in figure 3.5; to remove outliers the two following rules were used :

- if $x(i) > x(i-1) + 5$ then $x(i) = x(i-1)$
- if $x(i) > x(i-1) + 2$ and $x(i) > x(i+1) + 2$ then $x(i)=x(i-1)$

The resulting data is shown in figure 3.6; figure 3.6c shows the difference between the two signals; this had an average value of .52 degrees, and a standard deviation of .64 degrees. As the variance of the difference between the plots is twice the variance of each plot alone we can estimate the standard deviation of the measurements at approximately .45 degrees.

Furthermore, the saccades on the filtered signals confirm qualitative analysis of the videotape. Thus, in spite of poor input video data quality, quantitative results can be obtained.

3/ Multiple Image Comparisons, Real Data - Opaque Lens

This data was also run twice, with no subsequent filtering. The resulting plots are shown in figure 3.7. The sample mean of the difference between the two plots was 0.029, and its sample standard deviation was calculated as 0.160. Using the same reasoning as above, we can estimate the standard deviation of each measurement at approximately 0.11 degrees. Also almost all points on the difference plot (221 / 230) were contained in the

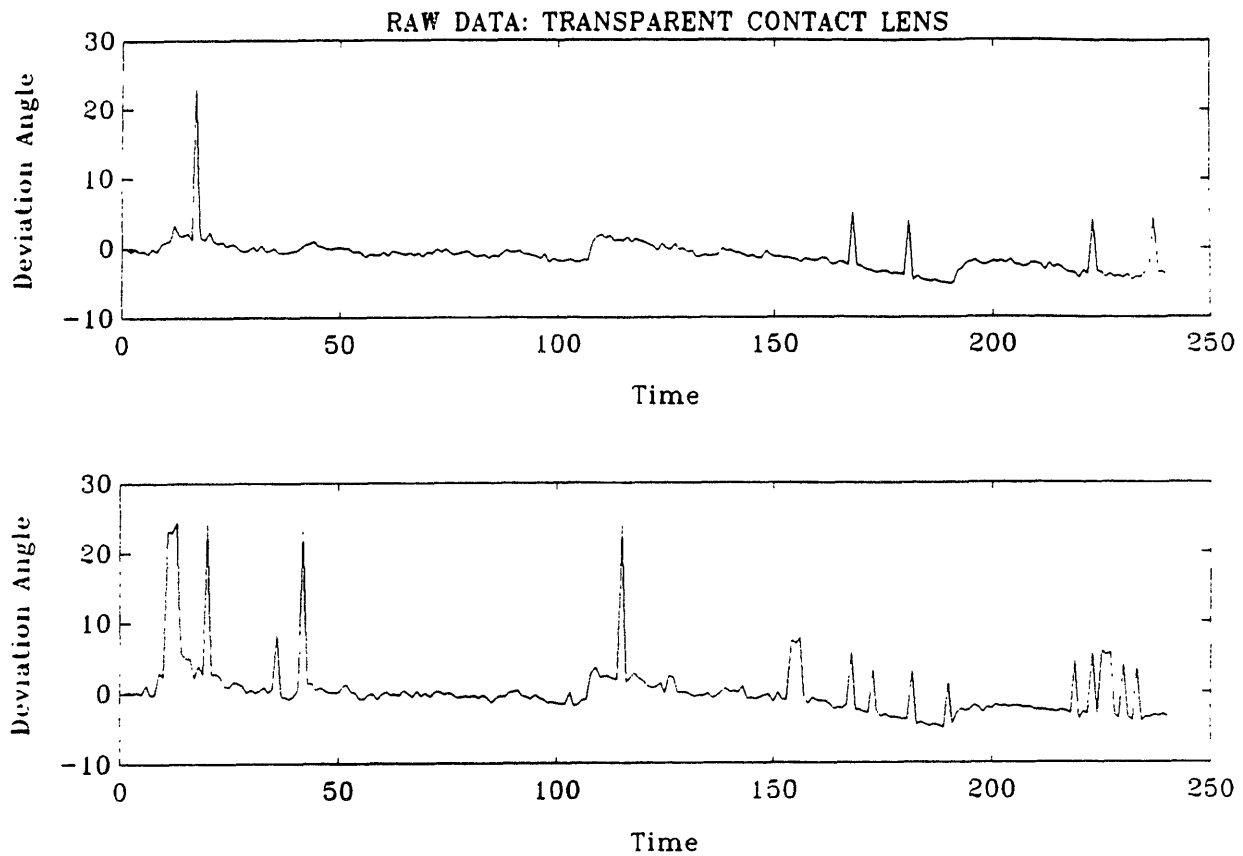


Figure 3.5: Tracking eye rotation with a transparent lens with black marks. Top and bottom : two different measurements of the same videotape input; With raw data, trends are difficult to quantify although they are clearly visible.

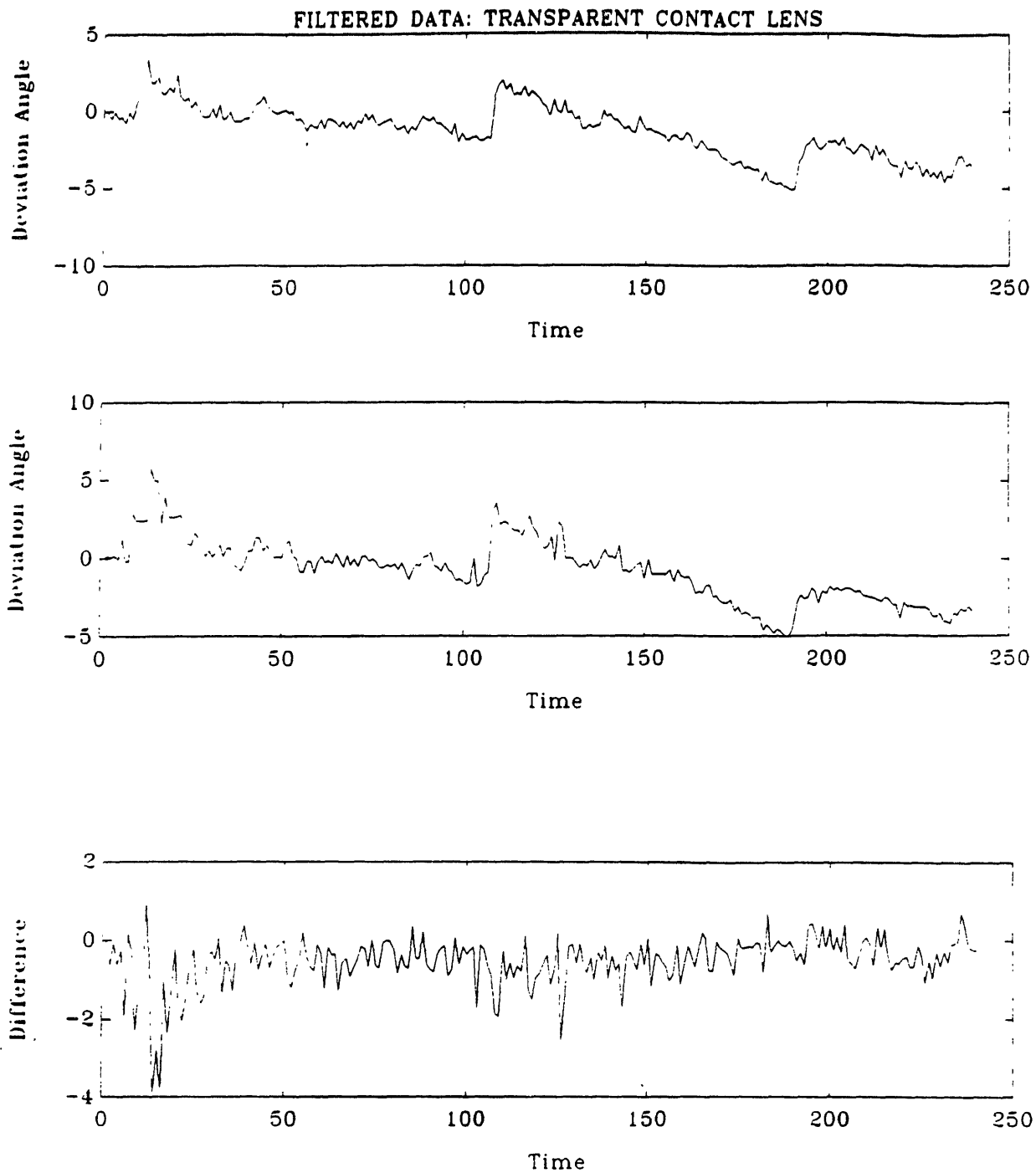


Figure 3.6: Tracking eye rotation with a transparent lens with black marks. After filtering. Top and middle : two different measurements of the same videotape input; bottom : difference between the two measurements. The sample standard deviation of the difference is .64 degrees, suggesting a measurement standard deviation of .45 degrees.

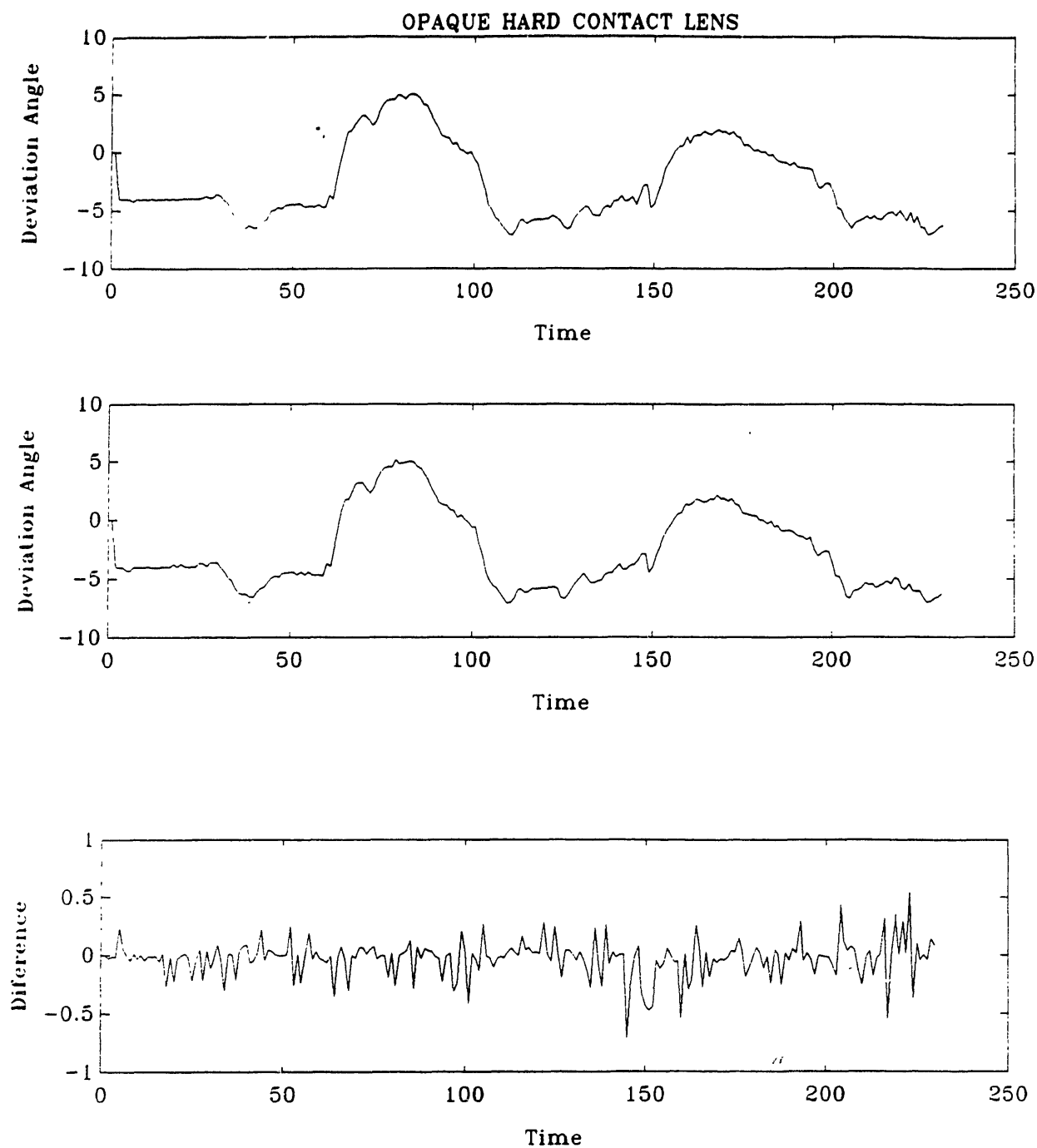


Figure 3.7 : Tracking eye rotation for an opaque hard lens with two bright dots. Raw data. Top and middle : two different measurements; bottom : difference between the two plots. The sample standard deviation for the difference is .16 degrees and 221 of the 230 points are in the interval $[-.4;+.4]$; the standard deviation of each of the measurements is thus .11 degrees approximately.

interval $[-.4,+.4]$, and only one (.72) was above .6 degrees. Finally, the output runs were also confirmed by qualitative analysis of the videotape.

C/ Comparison with 35mm Scanning

As a final test this method was compared with the other one used in the Man Vehicle Laboratory for measurement of ocular torsion. Two different sets of data consisting of 4 and 3 slides respectively were selected from a dome run. They were scanned manually once. They were then projected onto a screen and videotaped. Rotation was calculated from the resulting videotape using TOMAS. For each set of data, a reference slide was chosen and rotation was calculated five times for each remaining slide. The average value, dc , and sample standard deviation were then stored (table 3.2). To compare results, the difference between the angle value of the test image and that of the reference image had to be calculated. Thus for the scanner, precision for this difference, ds , was .6 degrees. For all angles calculated, $dc - ds$ was less than .6 degrees. This tends to confirm that for good input video pictures, the automatic method is at least as accurate as the 35mm scanning technique on transparent lenses with black markings. However, it is not surprisingly less precise on this data than on the opaque lens data.

D/System Limits

1/ Limits of Rectilinear Measurement Technique

	Scanner Data		Video method		
	angle	difference ds	average dv	standard deviation	dv-ds
Reference	2.263	/	.047	.012	.047 ¹
1	2.775	.512	.716	.200	.204
2	3.013	.750	1.09	.155	.344
3	1.008	-1.25	-1.653	.012	.403
Reference	5.714	/	.027	.025	.027 ¹
1	5.272	.442	.297	.166	.145
2	5.839	.125	.627	.194	.502

Table 3.2: Comparison Between Video and Manual Scanning Methods.

1: For the reference image dv-ds is the accuracy of the video method when comparing an image to itself. This is clearly better than the .3 degree average precision of the manual scanning method.

Parker recommends use of the rectilinear technique for deviations of less than 5 degrees, but suggests that it is still quite accurate for angles of up to 10 degrees (Table 3.3) [Parker 1983]. The gradual loss of accuracy of the system with increasing angle was investigated by rotating the idealized inputs by 8,10,15,20,25, and 30 degrees. These results, along with the data from the original rotation of idealized data, tend to confirm Parker's results (Table 3.4).

2/ Effect of Blinks

The system was used on hard opaque lens data containing blinks to determine their effect on measurement (fig 3.8). Data containing three blinks was analyzed twice and the difference plot was calculated (figure 3.8). The blinks were contained in the first three seconds, or 90 data points. Clearly the data presents some "wild" points in the area containing the blinks; also there is a loss of precision in those areas, as the standard deviation of the difference between the two plots calculated over the first 90 points is .19 degrees while over the following 90 points (without blinks) it is .14 degrees. Manually, three areas containing blinks were found: these were of widths 8, 5 and 11 points respectively; although each contained a "wild" point, the other points in the region did not show up on the plot as being obviously wrong. Thus while the program seems to detect blinks, it is up to the operator to define the region around each "wild" point that should be discarded.

angle	mean error	std
1	-.021	.045
2	-.029	.052
3	-.029	.060
5	-.043	.088
10	-.094	.179

Table 3.3: Effect of Rotation on Accuracy of Translational Measurements. Data is subpicture of eye, with no contact lens. All angles in degrees; std: Standard Deviation. From Parker 1983.

angle	mean	std	angle	mean	std
0	-0.027	.077	8	7.992	.046
1	0.984	.049	10	9.848	.106
2	1.985	.039	15	14.093	.369
3	3.024	.100	20	19.141	.249
4	3.949	.090	25	23.102	.220
5	4.877	.099	30	28.978	.218

Table 3.4: Data from original idealized data test (angles 0..5 degrees), and limit test on idealized data (angles 8-30 degrees). Results tend to corroborate those of Parker (Table 3.3). Std and mean seem to deteriorate gradually, with a bias in angle estimation toward lower values.

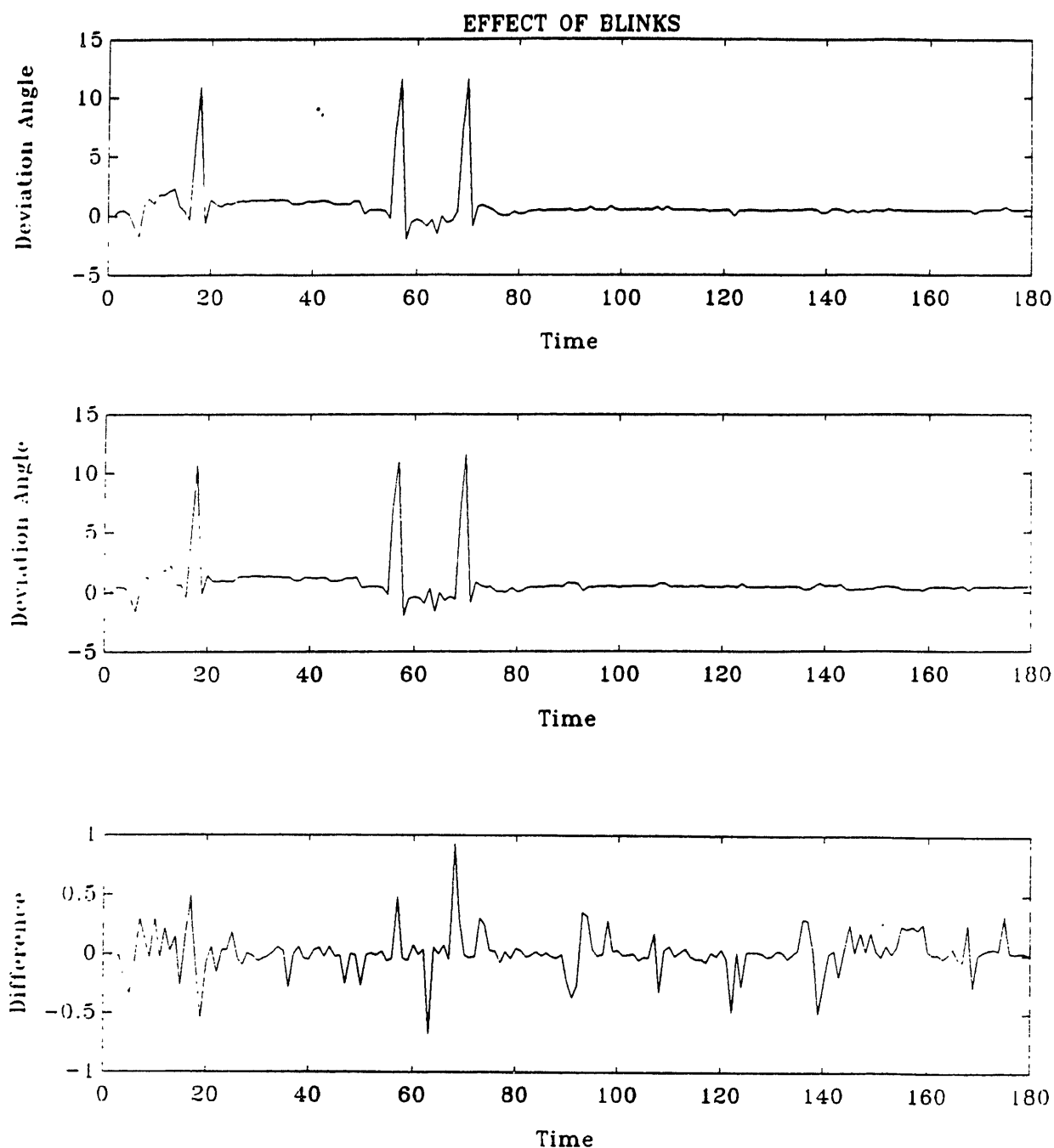


Figure 3.8 : Effect of Blinks. Top and middle: two different measurements of opaque lens data containing three blinks. Bottom: difference between the two plots. The blinks are visible, and the first half of the difference (containing the blinks) has a higher standard deviation (.19 degrees) than the second half (no blinks, .14 degrees). However, the operator must choose which points surrounding the "wild" points to discard.

E/ Operating Characteristics

Execution time was estimated from the different test runs, as well as the global "overhead" time necessary to start an analysis run.

For an image with just two landmarks, (no fiducial marks) the program execution required 13 seconds per frame; for an image with all four marks, analysis took 25 seconds per frame. Thus for a fifty second run containing 1500 frames, analysis time can be expected to take 5 1/2 or 11 hours depending whether fiducial marks are used or not.

Image acquisition takes approximately two minutes if all the filtering options are chosen correctly, and landmark locations are known. However, for an image of a type that is unfamiliar to the operator, it is necessary to try different prefiltering schemes, and to run the program once or twice comparing the reference image to itself before choosing final landmark positions and options. The time involved in choosing landmark locations and options is a function of original image "quality" i.e. suitability for analysis. For the opaque lens data for example, very few options were necessary, and excellent results were obtained in all cases. For the transparent lens data on the other hand, approximately 15 minutes of trial and error work was necessary to optimize these parameters for a given run.

Also, for a run where the eye moves considerably, it is important to choose a reference image such that the regions of interest remain inside the test subpicture for as many frames as possible. When the ROI leaves the subpicture, results similar to those encountered in the case of blinks can be expected.

Finally, due to a problem with the frame grabber software, the program occasionally crashes after an image acquisition. This may mean that an experiment run of 1500 images would have to be segmented into two 750 image analysis runs. The additional overhead here is slight as the same reference image is chosen for both runs, and therefore options and reference landmark locations need only be optimized once.

IV Discussion and Recommendations

In part A of this chapter the results obtained by the system are examined and its configuration is evaluated. Part B deals with the implications in terms of experiment set up and data analysis, and part C contains suggestions for further research.

A/ Discussion of Results

1/ System Validation

Clearly, the analytic method validation tests (III A/) were successful. This confirms that the hardware chosen for the system is sufficiently sensitive to calculate translation of contact lens markings on the eye in the absence of a video interface.

The outcome of the test on idealized data (table 3.1, figure 3.4) suggests that for high signal to noise ratios in input data, angles can be determined with a bias on the order of .05 degrees and a precision of better than .1 degree over the [-5 , +5] degree range. When opaque contact lenses with high contrast landmarks are worn, little precision appears to be lost, as the standard deviation of the measurements is again approximately .1 degree (figure 3.7).

The difficulties encountered by the system when analyzing data

involving transparent contact lenses with black markings (figure 3.5,3.6) may result from a number of causes :

- The data was taken on 3/4" tape (U_MATIC) format, and then transferred to SuperBeta format, leading to some signal degradation;
- As discussed in chapter II, the contact lens markings were in the same spatial frequency range as many natural iral landmarks, and especially in the same range as the high intensity dome reflections.
- The data was taken with no a priori knowledge of the analysis system characteristics, and minimum lighting and focus levels were not always respected.

However, even with the above input data quality constraints, segments of the transparent lens data were analyzable; further, the low variance on the measurement of 35mm slides that were videotaped (table 3.2) suggests that the transparent contact lenses with black markings can be used quite successfully for measurement of ocular torsion if the experimental setup is adequate.

2/ System Limits

Relatively little degradation occurred either in terms of measurement mean or variance with increasing angle. For angles of less than 10 degrees, landmark rotation effects appeared negligible, and even for larger angles (up to 30 degrees) it appeared to be small enough (less than two degrees) to allow qualitative data analysis. One should however be cautious, as the amount of distortion of the region of interest for a given angle depends on

landmark shape. For example circularly symmetric landmarks should not lead to any distortion.

For the landmarks used, the degradation of precision and accuracy seem to be gradual, suggesting that the system can still be used even for relatively large angles for qualitative analysis. This is particularly true for calculation of torsional eye velocity, as small position errors may have negligible effect on the velocity calculation.

The system also seemed capable of detecting blinks in the data; however special caution should be exercised when analyzing data from time periods surrounding clearly recognizable blinks.

3/ Operating Characteristics

The processing time of 13 seconds per frame for images with two landmarks, and 25 seconds per frame for images containing fiducial marks, compares favorably with the 75 minutes per frame obtained with the RT-11 system by Nagashima. Given that the processing time includes data acquisition and VCR control, a substantial savings was also obtained compared to the 30 seconds his software needed to calculate rotation without fiducial marks on a VAX-11/780 with a floating point accelerator.

Overall analysis time was closely correlated to input signal quality: images with lower signal to noise ratios such as the data from the D-1 mission demanded considerable time in choosing correct landmarks and options, and repeated analysis of the same run was often necessary. On the other hand, for the opaque lens data with no blinks, measurement variance

was very low on the first analysis run, and program setup was straightforward and quick.

4/ Choice of Hardware/Software

The hardware for TOMAS was divided into two parts : the array processor and PC which were responsible for algebraic operations, and the video interface, which contained and digitized the input video data. As explained in chapter two, two main problems surfaced :

- the input VCR was not designed to display a still frame image accurately as without a DT head the angle of attack of the tape was different on still frame mode from play mode;
- much of the high speed processing capability of the array processor was wasted by the need for frequent interaction with the PC.

5/ Conclusion: System Evaluation

The tests on opaque lens data suggest that this is a more accurate and robust method for video measurement of ocular torsion. The calculated precision of approximately .1 degrees is certainly competitive with the other measurement methods.

The comparison with 35mm scanning suggests that even for transparent lens data, TOMAS can be as accurate as other systems for measurement of ocular torsion.

Finally, a comparison with EMIRAT (the other operational video based torsional analysis system) shows that the two systems have different

characteristics, and are therefore suited to different experiment environments: the EMIRAT system does not use a marked contact lens, and it performs at a higher speed than TOMAS; however the use of contact lenses with TOMAS appears to permit higher accuracy. The data storage facility on TOMAS allows for a higher sampling rate (30Hz instead of 20Hz), and enables the operator to verify analysis by running data through the system more than once. Also, given an adequate experiment setup, TOMAS can be expected to work on almost all data sets, while EMIRAT remains sensitive to variations in iral features by subject and over time, and to pupil center detection. In particular certain subjects' eyes were not analyzable under any experimental conditions. Finally, except for remote control command, TOMAS only uses readily available commercial hardware, and could be copied for approximately \$25000. EMIRAT was custom made by the Laboratoire d'Electronique et de Technologie de l'Information in Grenoble, France for the Laboratoire de Physiologie Neurosensorielle in Paris [Vieville and Masse 1987]. Construction cost was not reported.

B/ Implications of System Characteristics

1/ Experiment Development

The TOMAS system seems to be extremely well suited to opaque contact lens data of the type described above; however, given certain procedural precautions, transparent lenses with small markings can be expected to give very good results:

- Videotaping the data directly on Sony SuperBeta format tape would lead to some noise reduction (see above).
- As reflections from the dome cause serious problems it might be worth changing the lighting (so that reflections do not occur on critical parts of the lens, i.e. subimages).
- For the sled, lighting should be improved; on the existing data, lighting is clearly insufficient. Another alternative here would be to use a more light sensitive camera. Also, focusing is an issue for the sled data, as the fiducial marks and eyes were generally not both in focus at the same time. The lighting and focusing problems are clearly linked, as improvements in lighting or use of a more light sensitive camera would allow the operator to set a smaller aperture and depth of field would increase.
- A minimum distance should be respected between landmarks; a rule of thumb would be to choose them at least 250-300 pixels apart if the image is 480*640 pixels.
- White marks might possibly be more visible than dark ones on some iral backgrounds;
- The shape of the marks is also of some importance. For high quality images, portions of simple crosses as currently exist on the lenses give excellent results; however broadening somewhat the spatial frequency spectrum of the regions of interest would be a further safeguard. The crosses might be replaced by multipointed stars for example. A "chirp" or "dartboard" type target could also be chosen so to broaden the spatial frequency of the region of interest. In this case, as the target would be circularly symmetric there would be no limit on image rotation (see above).

- A possible solution to most of the signal to noise problems would be to design a slightly larger lens, and draw the marks over the sclera; these marks should be as close as possible to the iris without risking overlap so as to ensure that the lens moves with the globe. Once implemented, with marks are chosen as described above, this solution could be expected to give excellent results.

2/ Data Analysis

As a safeguard against poorly chosen regions of interest, and to lower measurement variance, data should be run through the system more than once. A data editor might be added to allow comparison and removal of obvious outliers from the data sets.

Also, after analysis, the operator should review the tape at low speed, stopping at frames corresponding to "wild" measurement points, and defining regions to discard because they contain blinks, or because the region of interest is not contained in the test subpicture. This procedure is quite straightforward for data which has been time coded, but much more difficult otherwise.

C/ Suggestions for Further Research

This research leaves a number of avenues open to create a faster, more accurate system.

1/ Measurement of Horizontal and Vertical Eye Movements

As it exists now, the system could be used to measure horizontal and vertical eye movements. The output file from TOMAS contains the x,y coordinates of each landmark for each frame, and the resulting torsion angle with respect to the reference frame. The simplest approach would be to average the horizontal and vertical movements of the landmarks as calculated by the system; this would be proportional to the sine of the horizontal and vertical eye movements. However for each frame there is an uncertainty in acquisition of one pixel vertically. This is due to the imperfect video interface and would be solved by using a VCR with a dynamic tracking head (see below). This uncertainty is of no importance for calculation of torsion, as the angle between the "dumbbell" lines is independent of translation of the whole frame. As the whole image is 480 x 640 pixels wide, the resulting additional uncertainty from this error would be approximately 1/120 of the eye "height" if the eye took up half the screen vertically. Averaging results as described above would further reduce the measurement error.

Thus, by adding a few lines of code, the system could be used for measurement of horizontal and vertical eye movements as well as torsional ones.

2/ Accuracy Improvement

Circular correlation has been suggested as an alternative to rectilinear correlation, as with circular correlation the program could be

used for any angle with any type of landmark. The only limit on angle of rotation would then be subpicture size. Other sources of imprecision would appear however, as this method demands a fixed sampling origin which must remain stable over time; also, the video signal itself is best tuned to a rectilinear frame grabber. It is therefore necessary to use interpolation to change to circular coordinates [Parker 1983]. Given these drawbacks, and the small loss of accuracy associated with increasing angle for the landmarks used in this work, circular correlation may not be more accurate than rectilinear correlation for the eyeball tracking problem.

Another possible way of improving accuracy would be to correct for false torsion using horizontal and vertical eye movement information.

Accuracy would certainly be enhanced if the video interface were changed : A 3/4" Video Tape Recorder with a dynamic tracking (DT) head would output a much improved still frame image. Also a frame grabber that could lock on to more widely varying sync input signal (perhaps with a better phase lock loop circuit) would eliminate the need for a Digital Video Adapter.

3/ Speed Enhancement

Finally, speed might also be increased. One possible solution would be to first track the center of the eye using a real time algorithm; precision of this algorithm would not be very important, as its only object would be to reduce the size of the subpicture in which the region of interest would be searched for. If the area then became small enough, direct crosscorrelation calculation might be faster than Fourier Transform methods; accuracy would then also be enhanced.

Another solution might be to track the regions of interest using two one dimensional correlations rather than one two dimensional one.

Investing in the Image Processing Library for the ZIP3216 system might also lead to improvements in speed, as more processing could be done in the array processor, leading to less time consuming interaction with the PC. Other hardware systems might also be investigated for later versions of TOMAS; beyond processing speed, criteria for choice of such systems should include capacity for stand alone processing, and ease of direct programming of the system itself.

REFERENCES

- Arrott, A.P., (1982) : Torsional eye movements in man during linear accelerations, S.M. Thesis, MIT, Cambridge.
- Arrott, A.P., (1985) : Ocular torsion and gravito-inertial force, PhD Thesis, MIT, Cambridge.
- Collewijn, H., van der Steen, J., Ferman, L., and Jansen, T.C. (1985) : Human ocular counterroll: assessment of static and dynamic properties from electromagnetic scleral coil recordings. *Exp Brain Res* 59 : 185-196.
- Collewijn, H. (1977) : Eye and head movements in freely moving rabbits. *J Physiol*, 266 : 471-498.
- Collewijn, H., van der Mark, F., and Jansen T.C. (1975) : Research Note : Precise recording of human eye movements. *Vision Research*, 15 447-450.
- Diamond, S.G., Markham, C.H., Simpson, N.E., and Curthoys, I.S. (1979) : Binocular counterrolling in humans during dynamic rotation. *Acta Otol.* 87:490-498, 1979.
- Diamond S.G., and Markham C.H. (1988) : Ocular torsion in upright and tilted positions during hypo- and hypergravity of parabolic flight. *Aviat. Space Environ. Med.* 1988: 59:1158-62.
- Edelman, E.R. (1979) : Video based monitoring of torsional eye movements. S.M. Thesis, MIT, Cambridge.
- Hatamian, M. (1982) : An image analysis algorithm for real time measurement of X,Y and torsional eye movements- Theory and implementation. Ph.D. Thesis, University of Michigan.
- Hatamian, M. (1983) : Design considerations for a real time ocular counterroll instrument. *IEEE Trans on Biomed. Engin.* vol BME-30 NO. 5
- Lichtenberg, B.K. (1979) : Ocular counterrolling induced in humans by horizontal accelerations. Sc.D. Thesis, MIT, Cambridge.
- Lichtenberg, B.K., Young, L.R., and Arrott, A.P. (1982) : Human ocular counterrolling induced by varying accelerations. *Exp. Brain Res.* 48:127-136
- Massoumnia, M-A, (1983) : Detection of fast phase of nystagmus using digital filtering. S.M. Thesis, MIT, Cambridge, Ma.
- Miller, E.F., II. (1962) : Counterrolling of the human eyes produced by tilt with response to gravity. *Acta Otolaryngologica* 54:479-501.
- Miller E.F., and Graybiel A. (1971) : Effect of gravito-inertial force on ocular counterrolling. *J. Applied Phys.* 31:697-700.

- Parker, J.A. (1983) : Comparison of interpolation methods for image resampling. IEEE Trans Med Imag. Vol MI-2, No. 1.
- Parker, J.A., Kenyon, R.V., and Troxel D.E. (1983) : Measurement of torsion from multi-temporal images of the eye using digital signal processing techniques. Ph.D. Thesis, MIT, Cambridge Ma.
- Parker, J.A., Kenyon R.V., and Young, L.R. (1985) : Measurement of torsion from multi-temporal images of the eye using digital signal processing techniques. IEEE Trans. Biom. Engin., Vol BME-32, No 1.
- Robinson, D.A. (1963) : A method of measuring eye movement using a scleral search coil in a magnetic field. IEEE Trans Biomed. Electron. BME10:137-145.
- Vieville, T, and Masse, D. (1987) : Ocular Counter-rolling during active head tilting in humans. Acta Otol. Vol 103 No. 3-4.
- Young, L.R., Lichtenberg, B.K., Arrott, A.P., Crites, T.A., Oman, C.M., and Edelman, E.R. (1981) : Ocular torsion on earth and in weightlessness. Annals NY Acad. Sci. vol 374, 80-92.
- Young, L.R., Shelhammer, M., and Modestino S. (1986) : MIT/Canadian vestibular experiments on the Spacelab-1 mission : 2. Visual vestibular tilt interaction in weightlessness. Exp Brain Res 64: 299-307

APPENDIX I: USER'S GUIDE

TORSIONAL OCULAR MOVEMENT ANALYSIS SYSTEM

TOMAS

USERS GUIDE

Introduction

This guide contains information on TOMAS hardware and connections, and explains how to use the system, and how to copy tape to the Sony VCR with time encoding. The user must be familiar with the DOS operating system to use TOMAS; additional information on the different parts of the system is contained in the documents referenced in this text.

I/ Physical Set Up

The system diagram is shown in figure 1. The necessary connections are as follows :

- PC to remote controls: plug in 36 pin connector in back of PC;
- PC to array processor: 2 color coded 32 line buses; (see ZIP User's Manual chapter 1 for details)
- VCR to time encoder: Line 1 out to bottom plug of "VITC RD'R INP" on Time encoder. Be sure that line 1 (L1) has been chosen on "input select" on the VCR;
- Time encoder to Video Input on Sony Box: Go from the same plug on the Time encoder to the Video Input plug on the Sony box;

- Video out on Sony Box to Video in on Array processor: The input on the array processor is the small (8 pin) connector, J4 on board VD of the array processor.

- Video out on array processor to TV Video in: this is the larger (16 pin) plug (J5 on board VD); (see Zip Video handbook for details on array processor connections).

II/ Analyzing a Run

1/ Overview

I suggest the following strategy to analyze a run of data:

- choose a reference frame somewhere in the run, load it into the Digital Video Adapter (Sony box), and stop the VCR. Then choose landmark locations and options. Run TOMAS; as the VCR is on STOP, you will be comparing the image in the Sony box to itself. Run TOMAS repeatedly with different options/landmarks until the correlation angle is calculated to be essentially zero (i.e. less than 0.1 degree): each landmark is calculated to have moved less than half a pixel in any direction. This means that the program has found the maximum integer translation to be zero for each landmark when comparing it to itself.

- With the same image in the Sony box, run TOMAS using the options

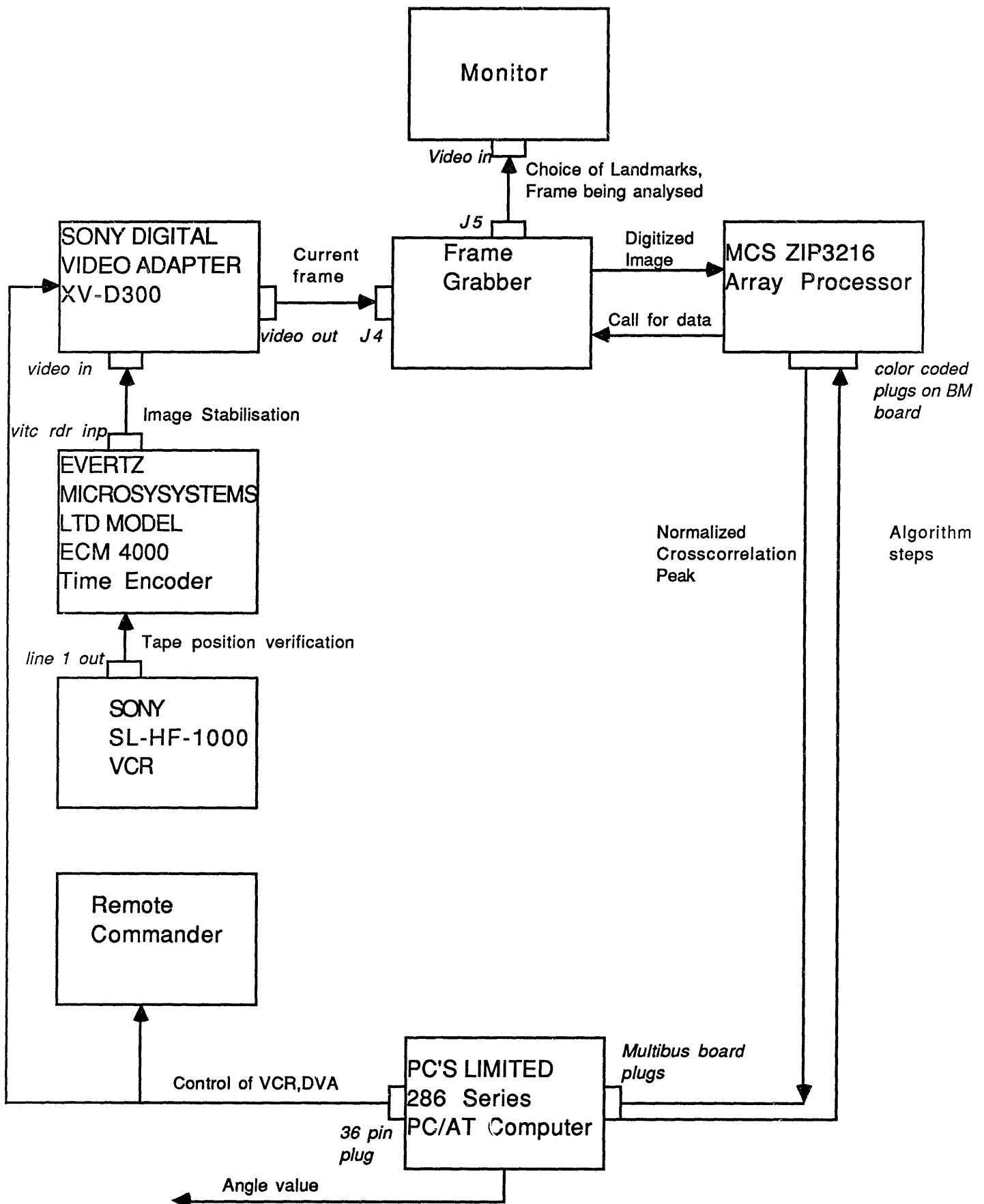


Figure 1 : Data Analysis Pipeline. Connectors in italics.

chosen previously, a number (say 10) of times. If the result is consistently zero (i.e. less than .1 degrees), then you can be confident that your choice of subpicture locations and landmarks is pretty good, and you are ready to analyze your data.

- Now switch the VCR back to PLAY, press Pause when you get to the reference image, store in the Sony box, and run the program, comparing the different frames in the run to the original.

2/ Running TOMAS

- POWER :

Power up each part of the system not forgetting to press the Video button on the TV after switching it on. Be sure that the remote controls can command the VCR and Sony Box from where they are. The system components are: The PC, The PC monitor, the array processor, the VCR, the time encoder, the TV, and the Sony box.

- Starting the program : type TOMAS, this will put you in the directory c:\TOMAS and will call the program.

At this point you will be asked to set the options; for most questions the default values are fine.

a/ Choosing the options

- output file name: respond with an answer of the type
xxxxx.yyy the first part of the name may contain up to eight letters,
and the second part up to three. DO NOT put in a path such as c:\foo\
in front of the filename; all files are put in the c:\results\
subdirectory.

- 128x128 subpictures ? (Y/N) [Y] : the default value of "y", "Y"
or simply <cr> (carriage return) is appropriate in almost all cases. The
only time to answer "n" or "N" is if you know that one of your
landmarks is very close to the edge of the image, and with the 128 x
128 size you cannot get it between the bright line of the video input
cursor because the black lines defining the subpicture are "stuck"
against the side of the picture. If you do answer "n" then you are
given the choice between 64x64 96x96 and 128x128. Remember that
by choosing something other than 128x128 you are reducing the
scope of your crosscorrelation function. If you answer:

-128*128, landmark translation of up to 48 pixels in any direction
will be searched for;

- 96x96, this number reduces to 32;

-64x64, you are down to 16 pixels.

- Mexican Hat Filter ? (Y/N) [N] : The default answer of no
seems indicated unless you can not get satisfactory results otherwise,
and you have reasons to believe that linear filtering will improve

your images; remember that this option almost doubles processing time.

- Threshold images ? (Y/N) [Y] : Generally it is a good idea to threshold images; if you are using opaque lens data with bright dots or some other suitably high signal to noise data (meaning with regions of interest that are extremely clear and different from their background) then you might get away without thresholding; (try both with and without...)

- Are you using the time encoder? (Y/N) [Y] : If you know the time code for the first frame of your run type "Y" or <cr> here; the system next prompts for the reading on the time encoder "reader" window; .A typical response would be: 11:23:35:10 <cr>.

Note: For the first preliminary run on new data, you will not yet know the time code;

- Invert Subpictures ? (Y/N) [N] : Generally this option which makes everything black white and vice versa has no effect on processing so don't use it without a reason;

- Binary Image ? (Y/N) [N] : This works in conjunction with the thresholding option and sets all values above the threshold to the maximum positive value, enhancing contrast but throwing away information; again, don't use it without a reason.

- with fiducials ? (Y/N) [N] : answer "Y" if you are tracking four subpictures (including two fiducial marks), <cr> if only two.

- select with video interface ? (Y/N) [Y] : if you don't know the coordinates of your landmarks (for instance the first time you run the program on a reference picture) press <cr>, otherwise press "n".

- number of images : this is the number of comparisons to do; if you are still trying to choose your landmarks I suggest answering "1".

- Run Program ? (Y/N) [Y]: If you are happy with what you have done so far press <cr>.

b/ Acquiring the reference sub images

The program now initializes the array processor, and video board; it prompts you with:

Acquire Landmarks ? Before answering be sure that you have loaded your reference picture into the Sony box; to do this:

- press PLAY on the VCR;
- press PAUSE when you are near the reference image;
- Use the JOG SHUTTLE DIAL on the VCR to get to the reference picture location;

- load this image into the Sony box by pressing STOP MOTION on that box and waiting for the image to stabilize.

This might also be a good time (if you are still on your preliminary runs) to note on the time encoder the time coordinates of the first frame of your run.

Once you have chosen your reference image, and have stored it in the Sony box by pressing stop motion then :

- if this is a preliminary run, and you are going to want to compare this image to itself, press STOP on the VCR. The program always automatically sends the commands to step through the videotape frame by frame, and store each successive frame in the Sony box; pressing STOP makes the VCR ignore these commands, and allows you to keep the same image in the sony box, always comparing it to itself.

You may now answer "y <cr> " to the "Acquire Images" prompt on the screen.

At this point you are ready to start acquiring the reference image subpictures;

if you answered "yes" or <cr> to the question " select with video interface" then the following prompt will appear :

use arrows to move cursor, zero to accept : Be sure that the NUMLOCK

indicator is on on the PC keyboard (otherwise press the numlock button); you will see on the screen two long black horizontal lines, and two smaller white ones in between; these define the (larger) subpicture and (smaller) Region of Interest. Move the cursor until the landmark that you want to use as a basis for measurement of translation is between the white lines; if you find that a black line gets stuck against an edge and you would like to be able to push it a little further up or to the side to be able to get your landmark between the white lines, then you are in a (rare) case where you should have chosen the option 96x96 or 64x64 in the "number of lines" in the options section. Choose different landmarks for now and then exit the program as soon as you are given the chance, and start again...

To move the cursors :

4,6 (left and right arrows) move it by 8 pixels;

3,9 move you up and down by 8 pixels;

2,8 (up and down arrows) move by 2 pixels, and should be used for fine tuning.

When you are satisfied with your positioning, hit "0". The coordinates of your subpicture will appear on the screen: "ACQUIRED AT LINE XXX COL XXX " you might want to note these, as then the next time you run the program over this image, you will know the coordinates of your subpictures, and will not have to use the video interface.

if you answered "no" to the question "select with video interface" (you knew from experience the coordinates of your reference image landmarks) then the following prompt will appear :

-acquire at line : give the line number of the top left hand corner of the subpicture (you got this either from the results file (see below) or when you acquired this landmark the first time you ran the program on the picture (see above))

- acquire at column : see "acquire at line"

c/ preliminary filtering

The subpicture has now been acquired; you are prompted:

press <cr> to display image You will see the sub image you chose in the upper left hand corner of your TV screen.

press <cr> to stop display: carriage return stops display.

If you chose the "Threshold Subpictures" option, you are now prompted :

percent of range to set to zero : (in dec) : give a decimal integer value between 0 and 100; x sets all values under x% of the range to zero.(i.e makes them black).

The program displays the subpicture after each prefiltering operation chosen in the options. At each display the display/stop display prompts appear.

Finally, after all the preliminary filtering procedures have been done, the small Region of Interest is extracted and displayed. As the program has made this region zero mean, the lower values of the region have become negative and are therefore displayed by the Video board at the maximum positive value. In practice, this has the effect of inverting the image; therefore, do not be surprised if regions you expect to see black are displayed as very white.

The program next prompts :

satisfactory image: if you are happy with your choice of landmark locations and threshold values, press "y", otherwise press "n" to acquire and filter the landmark again.

Parts b and c are now repeated for each subpicture.

- Note: If you find you have made a mistake while processing the reference image, wait for the prompt asking you if you want to leave the program (using ^C will cause the ZIP to crash).

d/ program execution

Once all the subpictures have been acquired and filtered the following prompt appears :

- Press Y to start run, N to exit :
- If you are not satisfied with your options and subpictures press "N" or "n";
- ,if you are satisfied, and this is a if this is a preliminary run, press "Y" or "y";
- if you are satisfied, and this is not a preliminary run, you are now going to analyze the run:
 - go to the beginning of the experiment run on the tape;
 - press play on the VCR;
 - press PAUSE and use the JOG SHUTTLE DIAL to get to the beginning of the run;
 - press STOP MOTION on the Sony box;
 - now answer "Y" or "y" to the prompt.

The program will now execute; if this is a preliminary run, and you chose your landmarks with the video interface, you can find their numerical values on the "# 0" line of the results file; each

landmark's x and y coordinates are listed sequentially (line first then column). Null values for the fiducial marks are put in if their displacement is not being measured.

The second line of that file will contain the thresholding percentages that you chose; make a note of them if you want to use them for future runs.

IV/ Notes, Comments and Bugs

- Occasionally, the program will crash during acquisition of subpictures; you can tell this has happened if the cursor suddenly disappears from the TV screen, but the "Use arrows to move, 0 to accept " prompt is still displayed on the PC. (This is linked to a bug in the ZIP software I think). If this happens, the best thing to do is SWITCH OFF the PC and the ARRAY PROCESSOR. Then start again.

- Occasionally, the program will crash after acquisition of a test image; You can tell this has happened if the program stops with "acquire downloaded" displayed on the screen, and there is no movement for 30 seconds. Most of the results calculated so far (up to the last comparison that was a multiple of 20) are saved in the results file. Again do a hard boot, and run the program again with a DIFFERENT results file, from the point of the crash.

- If you wait too long with the video option on the TV before initializing the ZIP video, the TV will revert to normal viewing; press the video button again to go back to video.

- If you stay on pause on the VCR for more than 5-10 minutes, without sending any commands (such as frame advance) the VCR reverts to PLAY mode.

- The batteries may run out on the Sony box remote commander. You can tell this has happened if your data suddenly becomes flat (the Sony box kept comparing the original with the same test image). Replace the batteries, or avoid using the Sony box as follows : when the prompt "Press Y to start Run, N to Exit " appears, press the FLASH MOTION button on the Sony box, and press the button marked "----" repeatedly, to lower the sampling rate of the box to once every four seconds. In this way, the Sony box will acquire the videotape image every four seconds, and display it. There is a very small chance of error if the Sony box grab occurs at the exact same instant as the Zip video board grab.

- To see your results, the program **mform** in the c:\results\ directory will format your angle values for entry into Matlab, and will put them in "**mnfilename**" if "filename" is your results file.

- Don't forget to clean up the c:\results\ directory from time to time.

III/ Copying Tape From U_matic to Sony with Time Encoding

For connections see figure 2.

- Be sure VCR input select is on the correct input line (I always use Line 1);

- to set the time encoder reading to zero, press "SET" in the "PRESET" zone on the time encoder. Press again for time to start running.

- press the VCG button under the "generator" window to see time code generation;

- record the videotape;

- when you are done, press "VCG" under the reader window to be able to see the recorded tape.

For further information, see the MODEL ECM 4000 EDIT CODE MASTER INSTRUCTION MANUAL (most information is on pages 2.4-2.7)

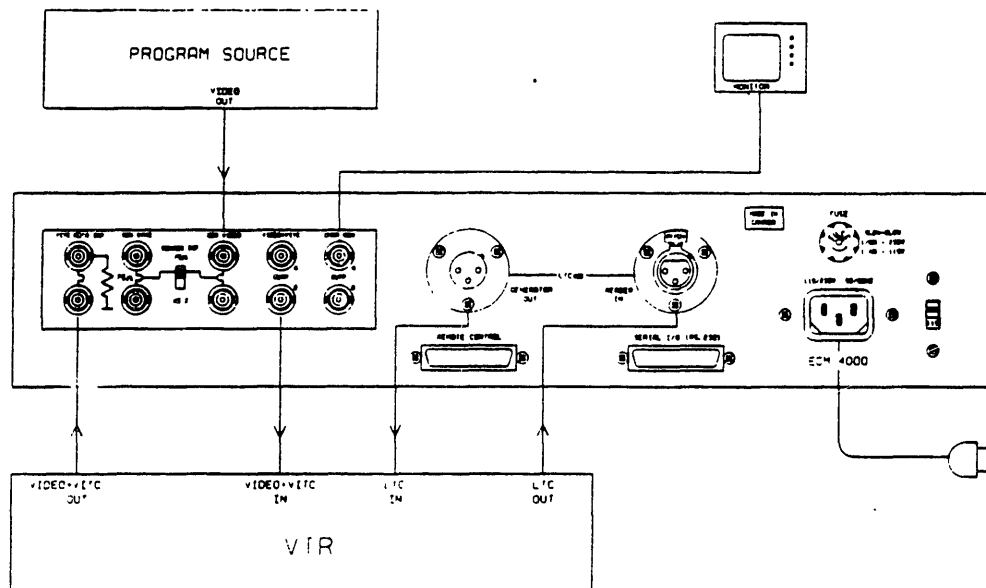


Figure 2 : Recording from U_matic to Sony with Time Encoding.
 From Evertz Model ECM 4000 Edit Code Master Instruction Manual
 p2-5.

APPENDIX II: PROGRAMMER'S GUIDE

TOMAS: Programmer's Guide

This file contains 3 parts:

- part I describes the organization of the files;
- part II lays out the algorithm, and states where each of the main subroutines can be found;
- part III goes into compiling and linking of C and ZIP programs.

I/Description of Files used in Landmark Tracking Program

1/ High Level Programs

- Tomas.c contains the main program, and the initialization subroutines.
- prep.c contains the routines for image (reference, and test) acquisition and prefiltering.
- u_comp.c contains the Correlation algorithm : q_track. This finds the translation of one landmark with respect to an original; u_comp.c also contains the peak finding subroutines.
- u_face.c contains the subroutines for the user interface.

2/ Middle Level Programs

- `u_angle.c` contains the subroutines that translate the one dimensional integer peak value of the crosscorrelation function to two real abscissa and ordinate values, and then find the angle of rotation with respect to the reference image.

- `u_mex.c` calculates the mexican hat filter.

- `r2dfft.c` contains FFT subroutines.

3/ Low level Programs

- `util.c` contains programs for management of ZIP memory; they use the zip executive functions extensively.

- `vidlib.c` contains the zip video management programs.

- `inutil.c` contains low level "useful" functions that do not involve the ZIP, like `summary` (open a results file and read the angle values), or `get16` (acquire an unsigned short integer).

- `zipcalls.c` contains calls to zip library routines.

- `command.c` contains the functions for command of the VCR from the PC.

II Algorithm/file correspondence

A/ Main program

Description	functions	File	
////////////////////////////////////			
/ main program	main	Tomas.c	/
/			/
/ *****initialization*****			/
/ parameter initialization	main,tables,n_load	Tomas.c	/
/ Make mexican Hat filter, mask	filters	Tomas.c	/
/ Acquire initial images, prefilter	acq_roi*	Prep.c	/
/ Define Original Angle	r_find_angle	u_angle.c	/
/ advance 1 frame	command	command.c	/
/ Fourier transform reference subpics	lfft2d	r2dfft.c	/
/ stop motion	command	command.c	/
/			/
/ *****for each picture until end of run : (run loop)*****			/
/ *			* /
/ * acquire test images and prefilter	acq_landmarks	Prep.c	* /
/ * advance 1 frame	command	command.c	* /
/ * send interrupt	q_interr	autil.c	* /
/ *			* /

B/ Acquiring Initial Images

Description	functions	File
////////////////////////////////////	////////////////////////////////////	////////////////////////////////////
/		/
/ Acquire Initial Images	Acq_roi	Prep.c
/		/
/ ***** initialization *****		/
/ initialize interface	init_face	u_face.c
/ grab reference screen	grab	vidlib.c
/		/
/ ***** loop on subpictures *****		/
/ *		*/
/ * ***** loop on each picture until accepted *****		/
/ * *		*/
/ * * acquire image with interface	face	u_face.c
/ * * OR give image coordinates	acq_roi	prep.c
/ * *		*/
/ * * grab and display subpicture	re_grab,display	vidlib.c
/ * * Non linear prefilter	acq_roi	prep.c
/ * * mexican hat filter	mexfilt	u_mex.c
/ * * capture ROI in Subpict	capture	autil.c
/ * * find average, subtract	f_avg, acq_roi	prep.c
/ * * extend to analysis size	extend	autil.c

/ * * is image acceptable ? * * /

/ * * ***** * /

/ ***** /

////////////////////////////////////./////////////////////////////////////

C/ Tracking Translation of a Test Subpicture with respect to a Reference

Description	functions	File
////////////////////////////////////		
/		/
/ Track translation of 1 subimage	q_track	u_comp.c
/		/
/ *****		/
/		/
/ fft of test image	q_lfft2d	r2dfft.c
/ square test image	vmul32	zipcalls.c
/ fft of squared image	q_fft2d	r2dfft.c
/ convolution of test image, reference	q_cmullc	zipcalls.c
/ convolution of square, mask	q_cmullc	zipcalls.c
/ inverse fft of power	q_lfft2d	r2dfft.c
/ inverse fft of correlation	q_lfft2d	r2dfft.c
/ correl * sqrt(power)	recip,vmul32	zipcalls.c
////////////////////////////////////		

III Compiling and Linking C and Zip Programs

A/ Compiling and Linking C Programs

TOMAS was written using Microsoft C version 4.00; the DOS version is 3.00. Given the size ZIP - C communication infrastructure, the large memory model must be used: when you compile the C program foo.c the appropriate command is

```
msc /AL foo
```

The finished program should be linked with the zip programs zip.obj and zipio.obj which can be found in the c:\driver\ directory. Also the stack size should be increased to 20000; given the different programs involved in TOMAS, the correct link statement is :

```
link /ST:20000 tomas u_comp u_face command+
u_angle util inutil prep+
vidlib zipcalls r2dfft+
u_mex o_util filters+
c:\driver\zip.obj c:\driver\zipio.obj , , ,c:\c\;
```

I have written a number of batch files for compilation; these are:

```
cltomas.bat: compiles Tomas.c, and performs link;
ltomas.bat : performs link;
cutil.bat  : compiles all files except Tomas.c.
```

B/ Compiling and Linking ZIP Files

The MCS system is the Mercury SPS-3216-IBM ATC-D Array Processing System, with the VID-N-170S-CV-DI Zip Video I/O.

If you are going to start programming the ZIP you should really look at the manuals (ZIP3216 user's guide and software reference manual especially).

To compile a ZIP program foo.zsr, type:

```
cpasm foo ;
```

to link your file with the executive and the library files you are using, I suggest doing :

```
zlink @zoo.cmd ;
```

and then, just put all you object file names into zoo.cmd, not forgetting to end this file with a <cr>.

All my zip files are listed in TOMAS.CMD.

Finally, I suggest leaving exec as the first file in zoo.cmd; then when you do zlink @zoo.cmd the output file will be EXEC.ZIP; if you want to change the name, do so using the DOS command RENAME (my zip file is TOMAS.ZIP).

APPENDIX III: COMMAND OF REMOTE CONTROL

Appendix III: Command of Remote Control

The PC commands the remote controls by sending signals through the Parallel port. This is a Metrabyte parallel digital I/O interface, model PIO12. The base address of this port is 0x300, and the control word to use for writing out from this port is 0x80 (see program command.c in appendix IV for further details).

The signals go directly from the parallel port to the interface circuit inputs shown in figure 2. Figure 1 shows the structure of the remote control switching system, and table 1 shows the Connection between these switches and the interface circuit. Figure 3 describes the power source for the remote control. For further details on the remote control, see the Sony SL-HF1000 RMT-148 Service Manual.

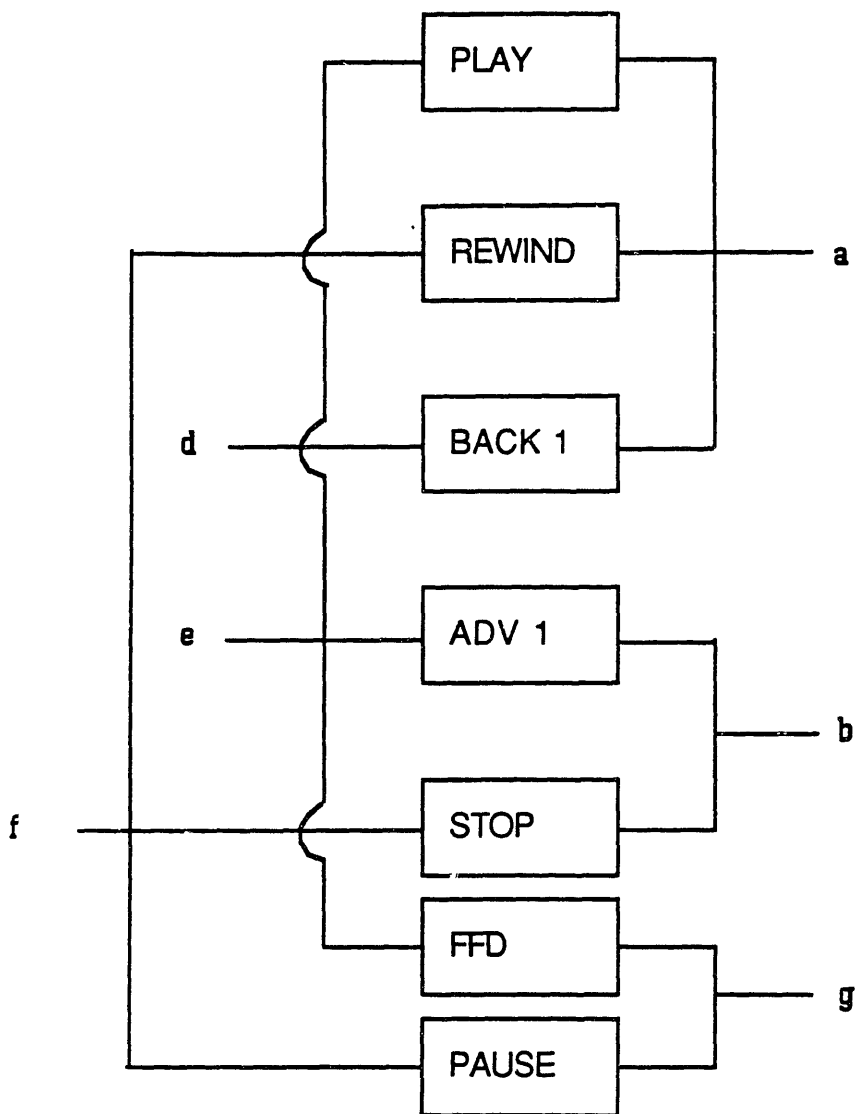


Figure 1: Remote Control Switch Interconnections

Command	Switch Connections	Interface Circuit Pins
play	a & e	A out
rewind	a & f	B out
back 1	a & d	C out
advance 1	b & e	D out
stop	b & f	E out
ffd	e & g	F out
pause	f & g	G out
stop motion (Sony box)		H out

Table 1: Connections from Interface Circuit to Remote Control Switches

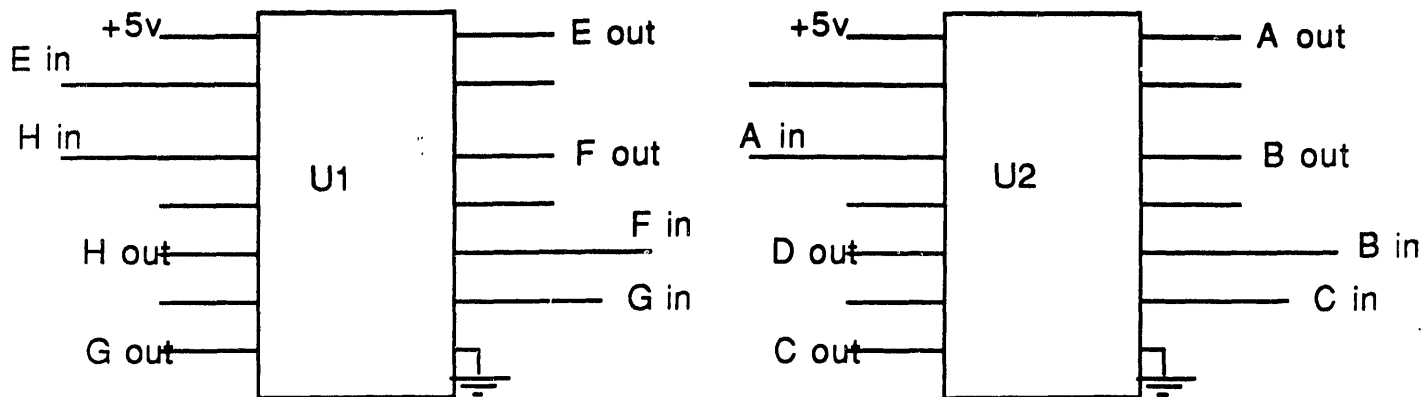


Figure 2: Remote Control Interface circuit.

Note: 2kohm resistor in series with each pair of output lines. 1kohm resistor between each output and ground.

U1, U2: 4066 CMOS Quad analog switch

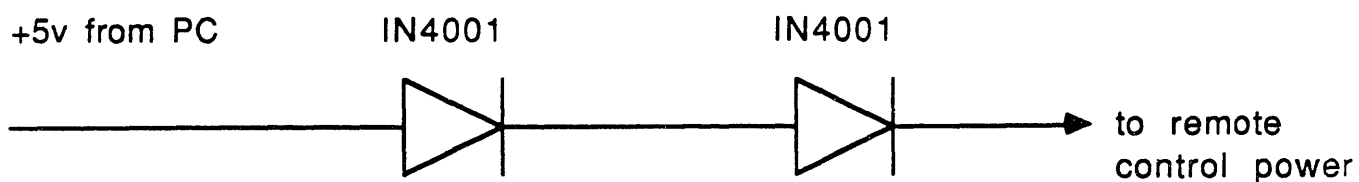


Figure 3: Power for Remote Control

APPENDIX IV: SOURCE CODE

TOMAS .c

```

/*****
*
*      NAME : TOMAS (Torsional Ocular Movement Analysis System)
*
*
*      DESCRIPTION : Calucates x,y coordinate translation of
*
*                    subpictures over successive frames, and
*
*                    calculates resulting torsion angle
*
*
*      SEE READ.ME FOR DETAILS ON PROGRAM SETUP AND EXECUTION
*
*
*
*      UPDATES      :
*
*                    TOMAS Version 1.0 02/20/1989 Andrew Alston
*
*
*****/

```

```

#include "c:\c\stdio.h"
#include "c:\driver\mcshdr.h"
#include "c:\driver\zip.h"
#include "c:\driver\import.h"

```

```

N16 data[0x8000];

```

```

main ()

```

```

{

N32 wkbuf,lcoef,lut,buf1,showbuf;

N32 screen,scradr,uplin, dnlne,uproi,dnroi;

N32 argcount,nwords,dtype,dummy,bufadr;

N32 mask,tmask,sq,tsq,tland,mh,tmh;

N32 land[4],roi[4],troi[4];

N32 im_top,cmultgt,vmultgt;

N32 acqovly,acqentpt,fftovly,fftentpt;

N16 datlin,datcol,imlin,imcol,mask_val,control;

N16 nelets,mexh;

N16 strtcol,strtrow;

N16 thresh[4],invert,do_mask,binary,r_avg[4];

N16 run,nb,hand;

int acq_row[4],acq_col[4],abs[4],ord[4],voisins[3][3][4];

int intval,result[4];

int rc,i,init,incr,hres,blank,loop,regions,def;

char s[80],filnam[80];

double theta,theta0,r_abs[4],r_ord[4];

FILE *fp,*fopen();

aa_init();

getstr("output file name ?",s);

sprintf(filnam,"c:\\results\\%s",s);

printf("output file is %s\n",filnam);

fp=fopen(filnam,"w");

```

```

/*****parameter initialisation*****/

/***** set option independent variables *****/

nelets=0x4000;
nwords=0x4000;
dtype=0;
imlin=0x80;
imcol=0x80;
for (i=0;i<8000;i++) data[i]=0;
mask_val=0;
cmultgt=0x50000000;
vmultgt=0x70000000;
    im_top=0x7fffffff;

def=1;
if (dy_or_n("128x128 subpictures ? (Y/N) [Y]",def)) datlin=128;
else getint("# lines in subpic (normally 128)",&datlin);
while ((datlin !=64) && (datlin !=96) && (datlin!=128))
    getint("choose 64, 96 or 128",&datlin);
datcol=datlin;

/***** fill array "data" with MHF if applicable ****/
mexh=0;
if (datlin==imlin)

```

```

        if (dy_or_n("Mexican hat filter ? (Y/N) [N]",mexh))
        {
            mexh=1;

            mex(data,imlin/0x80);
        }

/***** set options *****/

n_load(&invert,&do_mask,&regions,&binary,&nb,&hand,fp);
printf("dlin %d i %d dm %d rg %d bn %d nb
%d\n",datlin,invert,do_mask,regions,binary,nb);    if (dy_or_n("Run Program ?
(Y/N) [Y]",def)) ; else exit(1);

    printf("\n\n\n");

    strtrow=(imlin-datlin)/2;
    strtcol=(imcol-datcol)/2;
    tables(result,abs,ord,r_abs,r_ord,acq_row,acq_col,&theta,regions);

/***** initialize zip and set up video *****/

zipinit("tomas.zip");
ctrans("lut",&lut);
hres=0;
init_vid(hres,lut);
ftrans("acquire",&acqovly,&acqentpt);

```

```

/*****translate image names*****/

ntrans(&wkbuf,&lcoef,&buf1,&showbuf,land,roi,troi,&tmh,&mh,&tland,&mask,&tm-
ask,&sq,&tsq,&screen,&upline,&dnlne,&uproi,&dnroi);

    getadr(showbuf,&bufadr);

    ftrans("ltdfft",&fftovly,&fftentpt);

    getadr(screen,&scradr);

/***** create mask, Mexican hat filter *****/

filters(data,mexh,mh,tmh,mask,tmask,datlin,datcol,imlin,imcol,strtrow,strtcol,mask_val,buf1,cmultgt,nelets,wkbuf,lcoef,fftovly,fftentpt,nwords,dtype);

/*****acquire initial images*****/

acq_roi(fp,datlin,roi,acq_col,acq_row,data,buf1,acqovly,acqentpt,nwords,dtype,nelets,showbuf,bufadr,screen,scradr,invert,do_mask,thresh,binary,mexh,r_avg,imlin,strtrow,strtcol,mask_val,regions,hand,upline,dnlne,uproi,dnroi,troi,tmh,wkbuf,lcoef,cmultgt,im_top,fftovly,fftentpt);

printf("acquisition done\n");

r_find_angle(acq_row,acq_col,regions,r_abs,r_ord,theta,&theta0);

```

```

        fprintf(fp,"#    0 : %3.3f %3.3f% 3.3f% 3.3f% 3.3f% 3.3f% 3.3f% 3.3f
%3.31f\n",r_abs[0],r_ord[0],r_abs[1],r_ord[1],r_abs[2],r_ord[2],r_abs[3],r_
ord[3],theta);

        command(8); /* forward 1 frame */

        control=1;

        for (loop=0;loop<regions;loop++)

q_lfft2d(roi[loop],troi[loop],imlin,imcol,control,buf1,cmultgt,nelets,wkbuf-
,lcoef,fftovly,fftentpt);

/***** run loop *****/

        command(128); /* stop motion */

        if (y_or_n("type Y to start run, N to exit")) ;

        else zipout();

        for (run=0;run<nb;run++)

        {

                printf("\n\n\n comparaison # %d\n\n\n",run+1);

                /***** acquire landmarks *****/

acq_landmarks(datlin,land,acq_col,acq_row,data,buf1,acqovly,acqentpt,nwords-
,dtype,nelets,showbuf,bufadr,invert,do_mask,thresh,binary,mexh,r_avg,imlin,-

```

```

strthrow,strtcol,mask_val,regions,tland,tmh,wkbuf,lcoef,cmultgt,im_top,fftov-
ly,fftentpt);      command(8); /* forward 1 frame */

/***** loop on sub-images *****/

intval=1;
for (loop=0;loop<regions;loop++)
{
    q_interr(intval);

    /***** find integer translation *****/

    q_track(land[loop],tland,tsq,toi[loop],tmask,wkbuf,lcoef,cmultgt,vmultgt,i-
m_top,bufl,showbuf,imlin,imcol,nelets,fftovly,fftentpt);
    r_maxim(land[loop],imlin,datlin,nelets,data,loop,nwords,dtype,voisins,&resu-
lt[loop]);
}

/***** interpolate and find angle *****/
command(128); /* stop motion */
for (i=0;i<regions;i++) printf("result[%d]=%d\n",i,result[i]);
find_coords(0x20,imlin,datlin,result,abs,ord,regions);
ftune(abs,ord,voisins,r_abs,r_ord,regions);
r_find_angle(acq_row,acq_col,regions,r_abs,r_ord,theta0,&theta);
fprintf(fp,"##4d : %3.3f %3.3f% 3.3f% 3.3f% 3.3f% 3.3f% 3.3f% 3.3f

```

```

%3.31f\n",run+1,r_abs[0],r_ord[0],r_abs[1],r_ord[1],r_abs[2],r_ord[2],r_abs-
[3],r_ord[3],theta);          if (run % 20 == 0)
    {
        printf("data backup\n");
        fclose(fp);
        fp=fopen(filnam,"a");
        printf("data backup done\n");
    }
}

```

```

/***** close data file; print results on screen *****/

```

```

    zipcl();
    printf("in to data\n");
    fclose(fp);
    printf("data file written\n");
    summary(filnam,nb);
}

```

```

/***** initialisation *****/

```

```

n_load(pinvert,pdo_mask,pregions,pbinary,pnb,phand,fp)
N16 *pinvert,*pdo_mask,*pnb,*pbinary,*phand;
int *pregions ;

```

```

FILE *fp;

{
    int def,i,j ;
    int testr[80];
    def=1;
    if (dy_or_n("threshold subpictures? (Y/N) [Y]",def)) *pdo_mask=1;
    else *pdo_mask=0;
    if (dy_or_n("Are you using the Time Encoder ? (Y/N) [Y]",def))
    {
        for (i=0;i<80;i++) testr[i]=0;
        i=0;
        printf("time ? ");
        testr[i]=getch();
        printf("%c",testr[i]);
        while (testr[i]!=13)
        {
            i++;testr[i]=getch();printf("%c",testr[i]);
        }
        for (j=0;j<i;j++) fprintf(fp,"%c",testr[j]);
        fprintf(fp,"\n");
        printf("\n");
    }
    def=0;
    if (dy_or_n("invert subpictures? (Y/N) [N]",def)) *pinvert=1;
    else *pinvert=0;
    if (dy_or_n("binary image? (Y/N) [N]",def)) *pbinary=1;

```

```

    else *pbinary=0;

    if (dy_or_n("with fiducials? (Y/N) [N]",def)) *pregions=4;
    else *pregions=2;

    def=1;

    if (dy_or_n("select with video interface? (Y/N) [Y]",def)) *phand=1;
    else *phand=0;

    getint("number of images?",pnb);
}

tables(result,abs,ord,r_abs,r_ord,acq_row,acq_col,ptheta,regions)

int acq_row[],acq_col[],abs[],ord[],result[],regions;
double *ptheta,r_abs[],r_ord[];
{
    int i;
    for (i=0;i<4;i++)
    {
        result[i]=0x99;
        acq_row[i]=0;
        acq_col[i]=0;
        abs[i]=0;
        ord[i]=0;
        r_abs[i]=0.0;
        r_ord[i]=0.0;
    }

    if (regions==2)
    {

```

```

        r_abs[2]=0.0;r_ord[2]=0.0;

        r_abs[3]=100.0; r_ord[3]=0.0;

        result[3]=0;

        result[2]=0;

    }

    *ptheta=0.0;
}

```

```

ntrans(pwkbuff,plcoef,pbuf1,pshowbuf,land,roi,troi,ptmh,pmh,ptland,pmask,ptm-
ask,psq,ptsq,pscreen,pupline,pdnlne,puproi,pdnroi) N32 land[],roi[],troi[];
N32 *pwkbuff,*plcoef,*pbuf1,*pshowbuf,*ptmh,*pmh,*ptland,*pmask;
N32 *ptmask,*psq,*ptsq,*pscreen,*pupline,*pdnlne,*puproi,*pdnroi;
{
    /*****internal buffers, + display*****/

    ctrans("wkbuf",pwkbuff);

    ctrans("lcoef",plcoef);

    ctrans("buf1",pbuf1);

    ctrans("showbuf",pshowbuf);

    /***** landmarks *****/

    ctrans("el1",&land[0]);

    ctrans("el2",&land[1]);

    ctrans("fl1",&land[2]);

    ctrans("fl2",&land[3]);

    /***** rois *****/

    ctrans("er1",&roi[0]);

```

```

    ctrans("er2",&roi[1]);

    ctrans("fr1",&roi[2]);

    ctrans("fr2",&roi[3]);

/***** trois *****/

    ctrans("ter1",&troi[0]);

    ctrans("ter2",&troi[1]);

    ctrans("tfr1",&troi[2]);

    ctrans("tfr2",&troi[3]);

/***** analysis buffers *****/

    ctrans("tmh",ptmh);

    ctrans("mh",pmh);

    ctrans("tland",ptland);

    ctrans("mask",pmask);

    ctrans("tmask",ptmask);

    ctrans("sq",psq);

    ctrans("tsq",ptsq);

/***** interface buffers, display *****/

    ctrans("screen",pscreen);

    ctrans("upline",pupline);

    ctrans("dnlne",pdnlne);

    ctrans("uproi",puproi);

    ctrans("dnroi",pdnroi);

}

filters(data,mexh,mh,tmh,mask,tmask,datlin,datcol,imlin,imcol,strtrow,strtcol,mask_val,bufl,cmultgt,nelets,wkbuf,lcoef,fftovly,fftentpt,nwords,dtype)

```

```

N16 data[],datlin,datcol,imlin,imcol,mask_val,strtrow,strtcol,nelets,mexh;
N32

mh,tmh,mask,tmask,bufl,cmultgt,wkbuf,lcoef,fftovly,fftentpt,nwords,dtype; {

    N16 control ;

    control=1;

    if (mexh==1)
    {

        zput(data,&mh,&nwords,&dtype);

q_lfft2d(mh,tmh,imlin,imcol,control,bufl,cmultgt,nelets,wkbuf,lcoef,fftov-
ly,fftentpt);  }

    acq_mask(data,mask,nwords,dtype);

q_lfft2d(mask,tmask,imlin,imcol,control,bufl,cmultgt,nelets,wkbuf,lcoef,fft-
ovly,fftentpt); }

/*****acquire mask *****/
acq_mask(data,mask,nwords,dtype)
N16 data[];
N32 mask,nwords,dtype;
{

    N16 i,j,ilim;

    printf("nl6nwords %d \n",(N16)nwords);

    for (i=0;i<(N16)nwords;i++) data[i]=0;

```

```

        ilim=0x4000;

        for (i=0;i<0x20;i++)
            for (j=0;j<0x20;j++) data[0x80*(i+0x30)+j+0x30]=ilim;

        zput(data,&mask,&nwords,&dtype);
    }

aa_init()
{

printf("*****-
*****\n");    printf("*
                *\n");  printf("*
                        *\n");  printf("*      TOMAS (Torsional Ocular
Movement Analysis System)      *\n");  printf("*
                                    *\n");  printf("*
                                            *\n");

printf("*****-
*****\n");    printf("\n\n\n");
}

```

U_COMP.C

```
#include "c:\c\stdio.h"
```

```
#include "c:\driver\mcshdr.h"
```

```
#include "c:\driver\import.h"
```

```
/******
```

```
*   Name : q_track(land,tland,tsq,троi,tmask,wkbuf,lcoef,cmultgt,      *
```

```
*   vmultgt,im_top,bufl,showbuf,rows,columns,nelets,fftovly,fftentpt) *
```

```
*                                                                 *
```

```
*   Args :                                                                 *
```

```
*   N32  land : pre-filtered landmark, on test image (0x80*0x80)      *
```

```
*   N32  tland: fft of land (0x80*0x100)                                *
```

```
*   N32  tsq   : address of what will be fft of square of land         *
```

```
*   N32  троi  :fft of region of interest on original (0x80*0x100)    *
```

```
*   N32  tmask: fft of 0x20*0x20 mask (0x80*0x100)                     *
```

```
*   N32  wkbuf: work buffer for fft                                     *
```

```
*   N32  lcoef: buffer containing exponential values for fft           *
```

```
*   N32  cmultgt: target for autoshift before cmullc is called          *
```

```
*   N32  vmultgt: target for autoshift before vmull32 is called         *
```

```
*   N32  im_top: maximum for image (0x7fffffff)                         *
```

```
*   N32  bufl  : buffer for calculation of maximum                     *
```

```
*   N32  showbuf: display buffer                                         *
```

```
*   N32  fftovly, fftentpt : entry point & overlay of lfft2d          *
```

```
*                                                                 *
```

```
*   N16  rows,columns,nelets: rows columns,elements in picture        *
```

```
*                                                                 *
```

```

*      Description : finds normalised cross correlation function of      *
*
*      tland and troi; puts in land                                     *
*
*****/
/*****
*
*
*      Name : r_maxim(land,imlin,datlin,nelets,data,loop,nwords,dtype,  *
*      voisins,pmax_trans)                                             *
*
*
*      Args :   N32 land : contains correlation function                *
*
*      N32 imlin : lines (columns) in correlation function             *
*
*      N16 datlin : lines in original subpicture                       *
*
*      N16 nelets : elements in crosscorr (imlin *imcol)               *
*
*      N16 data : array                                                 *
*
*      int loop : subpicture numbber [0..3]                             *
*
*      N32 nwords : same as nelets                                     *
*
*      N32 dtype : data type for zip communication (dtype=0)          *
*
*      int voisins : will be filled with neighbors of max              *
*
*      for subsequent interpolation (see u_angle.c                       *
*
*      N16 pmax_trans : pointer to crosscorr maximum                    *
*
*
*      Descr : finds maximum of crosscorr function in area             *
*
*      of linear crosscorrelation (defined by datlin)                  *
*
*      puts that points coordinate in pmax_trans, and                  *
*
*      fills voisins[][][loop] with the values at                      *
*
*      the neighbors of the maximum                                     *
*

```

```

*****/
/*****
*
*
*   Name : frust(i_init,j_init,i_end,j_end,pmax,pindex,data)
*
*   Args : N16 i_init j_init, i_end, j_end, pmax,pindex,data
*
*   Descr : searches array "data" from (i_init,j_init) to (i_end,j_end)*
*           for maximum value; maximum is placed in pointer pmax
*           coordinate of maximum in pointer pindex
*
*****/
/*****
*   Name :   neighbours(data,ordinate,imlin,nelets,voisins,loop)
*
*   Args :   N16 data[],ordinate,imlin,nelets,loop
*
*           int voisins[][][];
*
*   Descr : fills voisins[][][loop] with neighbors of ordinate
*           from "data", given that array is (imlin*imlin)
*
*****/

```

```

q_track(land,tland,tsq,troi,tmask,wkbuf,lcoef,cmultgt,vmultgt,im_top,bufl,s-
howbuf,rows,columns,nelets,fftovly,fftentpt) N32

```

```

land,tland,tsq,troi,tmask,wkbuf,lcoef,cmultgt,vmultgt,im_top,bufl,showbuf;

```

```

N32 fftovly,fftentpt;

```

```

N16 nelets,rows,columns;

```

```

{
    N32 max32,min32;
    N16 control,type;

```

```

        /*****fft's*****/

        control=1;

        q_lfft2d(land,tland,rows,columns,control,buf1,cmultgt,nelets,wkbuf,lcoef,ff-
        tovely,fftentpt);    unpack(land,tsq,rows,columns);

        q_auto_shift(tsq,buf1,rows,columns,vmultgt);

        vmul32(nelets,tsq,tsq,tsq);

        q_fft2d(tsq,rows,columns,control,fftovly,fftentpt);

        /**** convolution,inversion****/

        printf("convolution,inversion\n");

        type=0;

        q_cmullc(tland,toi,tland,rows,columns,im_top,buf1,type);

        q_cmullc(tsq,tmask,tsq,rows,columns,im_top,buf1,type);

        control=3;

        q_lfft2d(land,tsq,rows,columns,control,buf1,im_top,nelets,wkbuf,lcoef,fftov-
        ly,fftentpt);

        q_lfft2d(land,tland,rows,columns,control,buf1,im_top,nelets,wkbuf,lcoef,fft-
        ovly,fftentpt);    recip(nelets,tsq,tsq);

        vmul32(nelets,tsq,tland,tsq);

        npack(tsq,land,rows,columns);

}

```

```

r_maxim(land,imlin,datlin,nelets,data,loop,nwords,dtype,voisins,pmax_trans)

int voisins[3][3][4],loop;

N16 datlin,data[],imlin,nelets,*pmax_trans;
N32 nwords,dtype,land;

{

    N16 ilim,index[4],max_index;

    int i,max[4],max_val;

    /*****find maximum*****/

    ilim=((datlin-0x20)/2);

    zget(data,&land,&nwords,&dtype);

    frust(0,0,ilim,ilim,&max[0],&index[0],data);

    frust(0,0x80-ilim,ilim,0x80,&max[1],&index[1],data);

    frust(0x80-ilim,0,0x80,ilim,&max[2],&index[2],data);

    frust(0x80-ilim,0x80-ilim,0x80,0x80,&max[3],&index[3],data);

    max_index=index[0];

    max_val=max[0];

    for (i=0;i<4;i++)

        if (max[i]>max_val) {max_val=max[i];max_index=index[i];}

    *pmax_trans=max_index;

    printf("maximun translation at %d\n",*pmax_trans);

    neighbours(data,max_index,imlin,nelets,voisins,loop);

}

frust(i_init,j_init,i_end,j_end,pmax,pindex,data)

```

```

N16 i_init,j_init,i_end,j_end,*pindex,data[];
int *pmax;
{
    N16 i,j;
    int dummy;
    *pmax=data[0x80*i_init+j_init];
    *pindex=0x80*i_init+j_init;
    for (i=i_init;i<i_end;i++)
        for (j=j_init;j<j_end;j++)
        {
            dummy=data[0x80*i+j];
            if (dummy>*pmax)
                (*pindex=0x80*i+j;*pmax=dummy;)
        }
    printf("max=%x at %x\n",*pmax,*pindex);
}

neighbours(data,ordinate,imlin,nelets,voisins,loop)
N16 data[],ordinate,loop,imlin,nelets;
int voisins[3][3][4];
{
    int i,j,k;
    for (i=0;i<3;i++)
    {
        for (j=0;j<3;j++)

```

```
{  
    voisins[i][j][loop]=data[(ordinate + (i-1) * imlin +j-1) % nelets];  
    printf(" %d %d %d",i,j,voisins[i][j][loop]);  
}  
printf("\n");  
}  
}
```

U_FACE.C

```

#include "c:\c\stdio.h"

#include "c:\driver\mcshdr.h"

#include "c:\driver\import.h"

/*****

*   name :      init_face(phglin,phgcol,mclin,mcol,pscrlin,pscrcol,      *
*               pstrtcol,pstrtrow,datcol,loop)                          *
*   Args :      N32 *phglin,*phgcol : pointers to screen size          *
*               N32 mclin[],mcol[] : coordinates of subpictures        *
*               N16 *pscrlin,*pscrcol : same as phglin,phgcol          *
*               N16 *pstrtrow,*pstrtcol : startrow, column for subpict  *
*               acquisition                                              *
*   Desc :      initializes above variables                             *
*****/

/*****

*
*   Name :      face(plin,pcol,datlin,datcol,hglin,hgcol,upline,uproi,  *
*               dnlne,dnroi,showbuf,bufadr,dtype)                      *
*   Args :      N32 *plin,*pcol : pointer to start row, column of subpict *
*               N32 showbuf : screen                                    *
*               N32 bufadr : buffer address                            *
*               N32 dtype : data type (0)                             *
*               N32 hglin,hgcol : lines, columns in showbuf           *
*               N32 upline,downline,uproi,dnroi : bright and dark lines *
*               defining subpicture, and region of interest            *
*   desc :      finds x,y coordinates of subpicture                    *
*****/

```

```

*****/

```

```

init_face(phglin,phgcol,mclin,mcol,pscrlin,pscrcol,pstrtc col,pstrtrow,dat-
col,loop) N32 *phglin,*phgcol,mclin[],mcol[];

```

```

N16 *pscrlin,*pscrcol,*pstrtrow,*pstrtc col,datcol;

```

```

int loop;

```

```

{

```

```

    int nb;

```

```

    *phglin=0x1e0;      /* 1d0 260 pick up full tv screen */

```

```

    *phgcol=0x278;

```

```

    for (nb=0;nb<loop;nb++) mclin[nb]=0xf0; /* e8,130 center of screen */

```

```

    for (nb=0;nb<loop;nb++) mcol[nb]=0x138-datcol/2;

```

```

    *pscrlin=(N16)*phglin;

```

```

    *pscrcol=(N16)*phgcol;

```

```

    *pstrtc col=0;

```

```

    *pstrtrow=0;

```

```

}

```

```

/*****

```

```

*                                                                 *

```

```

*   Name : face(plin,pcol,datlin,datcol,hglin,hgcol,upline,uproi,   *

```

```

*           dinline,dnroi,showbuf,bufadr,dtype)                      *

```

```

*   Args : N32 *plin,*pcol : pointer to start row, column of subpict *

```

```

*           N32 showbuf : screen                                     *

```

```

*           N32 bufadr : buffer address                             *

```

```

*           N32 dtype : data type (0)                             *

```

```

*          N32 hglin,hgcol : lines, columns in showbuf          *
*          N32 upline,downline,uproi,dnroi : bright and dark lines *
*          defining subpicture, and region of interest          *
*          desc : finds x,y coordinates of subpicture           *

```

```

*****/

```

```

face(plin,pcol,datlin,datcol,hglin,hgcol,upline,uproi,dnline,dnroi,show-
buf,bufadr,dtype) N32 *plin,*pcol,showbuf,bufadr,dtype,hglin,hgcol;

```

```

N32 upline,uproi,dnline,dnroi;

```

```

N16 datlin,datcol;

```

```

{

```

```

    N32 lut;

```

```

    N32 xlin,xcol;

```

```

    N32 snwords,nwords;

```

```

    N32 upadr,dnadr,rupadr,rdnadr;

```

```

    N16 scrlin,scrcol;

```

```

    N16 updata[0x200],dndata[0x200],rupdata[0x200],rdndata[0x200];

```

```

    N16 bldata[0x200],whdata[0x200];

```

```

    N16 input;

```

```

    int intval,i,hres;

```

```

    char filnam[80],s[40];

```

```

/*****set parameters*****/

```

```

    xlin=*plin;          /* e8,130 center of screen */

```

```

xcol=*pcol;

scrln=(N16)hglin;

scrcl=(N16)hgcol;


nwords=datcol;

snwords=0x20;


/***** acquire image *****/

upadr=bufadr + xlin * hgcol + xcol; /* mult of 4or8 ? */
dnadr= upadr + datlin * hgcol;
rupadr= upadr + (datlin/2 - 0x10) * hgcol + datcol/2 - 0x10;
rdnadr= rupadr + 0x20 * hgcol;
zmodbf(&uproi,&rupadr);
zmodbf(&dnroi,&rdnadr);
zmodbf(&upline,&upadr);
zmodbf(&dnline,&dnadr);

mark(upline,dnline,uproi,dnroi,updata,dndata,rupdata,rdndata,nwords,snwords-
, dtype);    mon(scrln,scrcl,showbuf);


input=5;
while (input!=0)
{
    printf(" use arrows to move area,0 to accept : ");
    input=getch()-48;
}

```

```

        printf("\n");

        while ((input!=0) && (input!=2) && (input!=4) && (input!=8) &&
(input!=3) && (input !=6) && (input!=9))        {

                printf(" use arrows to move area,0 to accept : ");

                input=getch()-48;

                printf("\n");

        }

        cl_mon();

        if (move_tst(input,&xlin,&xcol,hglin,hgcol,datlin,nwords)==1)

lines(xlin,xcol,upline,dnline,updata,dndata,nwords,uproi,dnroi,rupdata,rdnd-
ata,snwords,dtype,bufadr,hglin,hgcol,datlin,datcol,&upadr,&dnadr,&rupadr,&r-
dnadr);        mon(scrlin,scrcol,showbuf);

        }

        cl_mon();

fill_in(upline,dnline,uproi,dnroi,updata,dndata,rupdata,rdndata,nwords,snwo-
rds,dtype);        *plin=xlin;

        *pcol=xcol;

}

/*****
Name : move_tst(input,pmlin,pmcol,hglin,hgcol,datlin,nwords)
desc : defines possible cursor movement on screen
*****/
move_tst(input,pmlin,pmcol,hglin,hgcol,datlin,nwords)

```

```

N16 input,datlin;
N32 *pmlin,*pmcol,hglin,hgcol,nwords;
{
    int result;
    result=0;
    switch (input)
    {
        case 0 : break ;
        case 2 : if (*pmlin+2 <= hglin-datlin) /* -1 */
            {
                *pmlin=*pmlin+2;
                result=1;
            }
            break;
        case 3 : if (*pmlin+8 <= hglin-datlin)
            {
                *pmlin=*pmlin+8;
                result=1;
            }
            break;
        case 4 :if (*pmcol > 7)
            {
                *pmcol=*pmcol-8;
                result=1;
            }
            break;
    }
}

```

```

        case 6 :if (*pmcol+8+nwords<=hgcol)
        {
            *pmcol=*pmcol+8;
            result=1;
        }
        break;
    case 8 :if (*pmlin > 1)
    {
        *pmlin=*pmlin-2;
        result=1;
    }
    break;
    case 9 :if (*pmlin > 7)
    {
        *pmlin=*pmlin-8;
        result=1;
    }
    break;
}

return(result);
}

/*****
Names : lines fill-in, mark
*   desc : called by u_face
*****/

lines(mlin,mcol,upline,dnline,updata,dndata,nwords,uproi,dnroi,rupdata,rdnd-

```

```

ata, snwords, dtype, bufadr, hglin, hgcol, datlin, datcol, pupadr, pdnadr, prupadr, pr-
dnadr) N32 mlin, mcol, upline, dnlne, nwords, dtype, bufadr, uproi, dnroi, snwords;
N32 hgcol, hglin;
N16 updata[], dndata[], rupdata[], rdndata[], datlin, datcol;
N32 *pupadr, *pdnadr, *prupadr, *prdnadr;
{
    N32 nupadr, ndnadr, nrupadr, nrdnadr;

    printf("\n");

fill_in(upline, dnlne, uproi, dnroi, updata, dndata, rupdata, rdndata, nwords, snwo-
rds, dtype);    *pupadr=bufadr + mlin*hgcol + mcol;

    *pdnadr= *pupadr + datlin * hgcol;

    *prupadr= *pupadr + (datlin/2 - 0x10) * hgcol + datcol/2 - 0x10;

    *prdnadr= *prupadr + 0x20 * hgcol;

    zmodbf(&uproi, prupadr);

    zmodbf(&dnroi, prdnadr);

    zmodbf(&upline, pupadr);

    zmodbf(&dnlne, pdnadr);

mark(upline, dnlne, uproi, dnroi, updata, dndata, rupdata, rdndata, nwords, snwords-
, dtype); }

fill_in(upline, dnlne, uproi, dnroi, updata, dndata, rupdata, rdndata, nwords, snwo-
rds, dtype) N32 upline, dnlne, nwords, dtype, uproi, dnroi, snwords;
N16 updata[], dndata[], rupdata[], rdndata[];
{

```

```

        zput(updata,&upline,&nwords,&dtype);zwdma();
        zput(dndata,&dnline,&nwords,&dtype);zwdma();
        zput(rupdata,&uproi,&snwords,&dtype);zwdma();
        zput(rdndata,&dnroi,&snwords,&dtype);zwdma();
    }

mark(upline,dnline,uproi,dnroi,updata,dndata,rupdata,rdndata,nwords,snwords-
,dtype) N32 upline,dnline,uproi,dnroi,nwords,snwords,dtype;
N16 updata[],dndata[],rupdata[],rdndata[];
{
    N16 whdata[0x200],bldata[0x200];

    int i;

    for (i=0;i<(N16)nwords;i++) whdata[i]=0x7fff;
    for (i=0;i<(N16)nwords;i++) bldata[i]=0x0;
    zget(updata,&upline,&nwords,&dtype);zwdma();
    zget(dndata,&dnline,&nwords,&dtype);zwdma();
    zput(bldata,&upline,&nwords,&dtype);zwdma();
    zput(bldata,&dnline,&nwords,&dtype);zwdma();
    zget(rupdata,&uproi,&snwords,&dtype);zwdma();
    zget(rdndata,&dnroi,&snwords,&dtype);zwdma();
    zput(whdata,&uproi,&snwords,&dtype);zwdma();
    zput(whdata,&dnroi,&snwords,&dtype);zwdma();
}

```

COMMAND.C

```
#include "c:\c\stdio.h"
```

```
#include "c:\c\conio.h"
```

```
/******
```

```
*   Name :commmand(value)                                     *
```

```
*   args: int:value;                                         *
```

```
*   Description : Outputs value:                             *
```

```
*           1=play                                           *
```

```
*           2=rewind;                                         *
```

```
*           4=back1;                                          *
```

```
*           8=forward1;                                       *
```

```
*           16=stop;                                          *
```

```
*           32= ffwd;                                         *
```

```
*           64=pause;                                         *
```

```
*           128=Power;                                        *
```

```
*                                                           *
```

```
*****/
```

```
command(value)
```

```
int value;
```

```
{
```

```
    unsigned base,control;
```

```
    long int i;
```

```
    base=0x300;
```

```
    control=0x80;
```

```
    outp(base+3,control);  
    for (i=0;i<20000;i++) outp(base,value);  
    for (i=0;i<20000;i++) outp(base,0);  
    printf("command sent\n");  
}
```

U_ANGLE.C

```

#include "c:\c\stdio.h"

#include "c:\driver\mcshdr.h"

#include "c:\driver\import.h"

#include "c:\c\math.h"


#define PI 3.142592654


/*****
*   Name : find_coords(roisde,corsde,realsde,val,abs,ord,regions)      *
*   Args : N16 roisde,corsde,realsde : size of roi, of input data    *
*           of corelation                                             *
*           int : abs[],ord : output coordinates                     *
*           val : ldeimensional input coordinate                     *
*           regions : number of subpictures to process               *
*   Desc : converts one dimensional coordinate to two dimansional    *
*           and checks that it is in permissible range; repeats for  *
*           each subimage                                             *
*****/
find_coords(roisde,corsde,realsde,val,abs,ord,regions)
N16 roisde,corsde,realsde;
int abs[],ord[],val[],regions;
{
    int i,range,quot,rem,error,sign[4];
    range=(realsde-roisde)/2;
/*   printf("translating translation from 1 to 2 dimensions\n");*/
    for (i=0;i<regions;i++)

```

```

{
    rem=val[i] % corsde;
    if ((0<=rem) && (rem<=range)) ord[i]=rem;
    else
        if ((corsde-range<=rem) && (rem<=corsde))
            ord[i]=rem-corsde;
        else ord[i]=9999;
    if (ord[i]<0) sign[i]=1;
    else sign[i]=0;
    quot=val[i]/corsde;
    if ((0<=quot) && (quot<= range)) abs[i]=quot+sign[i];
    else
        if ((corsde-range <= quot) && (quot<= corsde))
            abs[i]=quot-corsde+sign[i];
        else abs[i]=9999;
}

error=0;
for (i=0;i<regions;i++)
    if ((ord[i]==9999) || (abs[i]==9999)) error=1;
}

/*****
*   Name : ftune(abs,ord,voisins,r_abs,r_ord,regions)   *
*   Args : int abs[],ord[] : coordinates of maxima   *
*           int voisins : values of correlation for neighbors of max *
*           double : r_abs[],r_ord[] : output real coords of maxima *
*   Desc : interpolates to find real maxima from integer max   *
*****/

```

```

*****/

ftune(abs,ord,voisins,r_abs,r_ord,regions)

int abs[],ord[],voisins[3][3][4],regions;

double r_ord[],r_abs[];

{

    N16 loop;

    int i,itiers;

    long dummy;

    double temp[3],t0x,t0y;

    for (loop=0;loop<regions;loop++)

    {

        t0x=0.;

        t0y=0.;

        for (itiers=0;itiers<3;itiers++)

        {

            for (i=0;i<3;i++)

                fa(t0y,voisins[i][0][loop],voisins[i][1][loop],voisins[i][2][loop],&temp[i-

]);

                t0(temp[0],temp[1],temp[2],&t0x);

                for (i=0;i<3;i++)

                    fa(t0x,voisins[0][i][loop],voisins[1][i][loop],voisins[2][i][loop],&temp[i-

]);

                    t0(temp[0],temp[1],temp[2],&t0y);

        }

        if ((t0x>1.0) || (t0x<-1.0)) t0x=0.0;

        if ((t0y>1.0) || (t0y<-1.0)) t0y=0.0;

    }

}

```

```

        dummy = abs[loop];
        r_abs[loop] = dummy + t0x;
        dummy = ord[loop];
        r_ord[loop]= dummy + t0y;
        printf("x,y translation : %f %f\n",r_abs[loop],r_ord[loop]);
    }
}

```

```

fa(t,f11,f22,f33,prtn)

```

```

double t,*prtn;

```

```

N16 f11,f22,f33;

```

```

{

```

```

    long dummy;

```

```

    double f1,f2,f3;

```

```

    dummy=f11;f1=dummy;

```

```

    dummy=f22;f2=dummy;

```

```

    dummy=f33;f3=dummy;

```

```

    *prtn=(f1- 2. * f2 + f3) / 2. * t * t + (f3-f1)/2. * t + f2 ;

```

```

}

```

```

t0(f1,f2,f3,prtn)

```

```

double f1,f2,f3,*prtn;

```

```

{

```

```

    double var;

```

```

    if (f1 - 2.0 * f2 + f3 == 0.0) var=0.0;

```

```

        else var=(f1-f3)/(2. * (f1-2. * f2 +f3));

        *prtn=var;

    }

/*****

*   Name : r_find_angle(origabs,origord,regions,newabs,neword,      *
*   otheta,ptheta)                                              *
*   Desc : finds rotation from change in coordinates from      *
*   (origabs,origord) to (newabs,neword)                        *
*****/

r_find_angle(origabs,origord,regions,newabs,neword,otheta,ptheta)

int origabs[],origord[],regions;

double newabs[],neword[];

double *ptheta,otheta;

{

    int i;

    long dummy;

    double vector0[2],vector1[2],angle2,angle1,norm;

    printf("angle calculation: \n");

    for (i=0;i<regions;i++)

    {

        dummy=origabs[i];

        newabs[i]= newabs[i] + dummy;

        dummy=origord[i];

        neword[i]= neword[i] + dummy;

    }

    vector0[0]=newabs[1]-newabs[0];

```

```

vector0[1]=neword[1]-neword[0];
vector1[0]=newabs[3]-newabs[2];
vector1[1]=neword[3]-neword[2];
norm=sqrt(pow(vector0[0],2.0) + pow(vector0[1],2.0));
if (norm==0.0) angle1=999.0;
else angle1=asin(vector0[1]/norm)*180/PI;
norm=sqrt(pow(vector1[0],2.0) + pow(vector1[1],2.0));
if (norm==0.0) angle2=999.0;
else angle2=asin(vector1[1]/norm)*180/PI;
*ptheta=angle2-angle1-otheta;
printf("original angle : %lf\n",otheta);
printf("eye wr film : %lf head wr film : %lf diff
: %lf\n",angle1,angle2,angle2-angle1);  printf("deviation : %lf\n",*ptheta);

}

```

AUTIL.C

```
#include "c:\c\stdio.h"
#include "c:\c\stdlib.h"
#include "c:\driver\mcs_hdr.h"
#include "c:\driver\zip.h"
#include "c:\driver\import.h"
```

```
extern N16 data[];
```

```

/*****ROUTINES*****/

```

/*****

```
*
*
*   Name : extend(datlin,datcol,imlin,imcol,strtrow,strtcol,mask_val
*           bufnum);
*
*   Args : N16 : datlin,datcol,imcol,imlin,mask_val,strtrow,strtcol,
*           N32 : bufnum
*
*   Description : Given a datlin x datcol array it puts it in an
*
*           imlin x imcol image, bufnum, with the top left hand corner at
*
*           (strtrow,strtcol). The surrounding pixels are set to mask_val.
*
```

*****/

/*****

```

*
* Name : capture(datlin,datcol,imlin,imcol,strtrow,strtcol,bufnum
*
* data);
*
* Args : N16 : datlin,datcol,imcol,imlin,strtrow,strtcol,data[ ]
*
* N32 : bufnum
*
```

```

*   Description : Captures a datlin x datcol array from an          *
*   imlin x imcol image, bufnum, with the top left hand corner at  *
*   (strtrow,strtcol).                                             *
*                                                                    *
*   *****/
*/
*                                                                    *
*   Name : auto_shift(image,buf1,imlin,imcol,target)              *
*   Args : N32 image,buf1,target; N16 imlin,imcol;                *
*   Desc : shifts image by necessary amount so as to be within     *
*   a factor of 2 of target                                         *
*   *****/
*/
*                                                                    *
*   Name : oq_auto_shift(image,buf1,imlin,imcol,target)           *
*   Args : N32 image,buf1,target; N16 imlin,imcol;                *
*   Desc : same but quicker and quieter                             *
*   *****/
*/
*                                                                    *
*   Name : q_auto_shift(image,buf1,imlin,imcol,target)            *
*   Args : N32 image,buf1,target; N16 imlin,imcol;                *
*   Desc : same as auto_shift but quicker and quieter and can handle *
*   negative inputs                                                 *
*   *****/

```

```

/*****
*
*
*   Name : cmax16
*
*   Args : N16:nelets; N32: inbuff,outbuff; N16:pmax,pmin
*
*   Description: finds max and min of the first nelets words
*
*   in inbuff, puts them in outbuff and brings them back in tab,
*
*   and print their values. NOTE : nelets IS N16 I.E. THERE MUST
*
*   BE LESS THAN 0X100*0X100 ELEMENTS IN IMAGE!!!!
*
*****/
/*****
*
*
*   Name : q_cmax16
*
*   Args : N16:nelets; N32: inbuff,outbuff; N16:pmax,pmin
*
*   Description: same as above but doesn't print values
*
*****/
/*****
*
*
*   Name : cmax32
*
*   Args : N16: nelets; N32: inbuff,outbuff; N32:pmax,pmin
*
*   Description: finds max and min of the first nelets words
*
*   in inbuff, puts them in outbuff and brings them back in tab,
*
*   and print their values  NOTE : nelets IS N16 I.E. THERE MUST
*
*   BE LESS THAN 0X100*0X100 ELEMENTS IN IMAGE!!!!
*
*****/

```

```

*****/
/*****
*
*      Name : q_cmax32
*
*      Args : N16: nelets; N32: inbuff,outbuff; N32:pmax,pmin
*
*      Description: same as above but doesn't print values
*
*****/
/*****
*
*      Name : getadr;
*
*      args : N32 : bufnum , pbufadr
*
*      Description : given a buffer number, getadr puts its
*                    address in *pbufadr
*
*****/
/*****
*
*      Name : ctrans
*
*      Args : s : 80 char string N32 bufnum
*
*      Description : translates buffer s to N32 bufnum
*
*****/
/*****
*
*      Name : ftrans
*
*      Args : s : 80 char string N32 *povly,*pentpt
*

```

```

*   Description : translates function s to N32 overlay,entry_point      *
*****/
/*****
*   Name : interr(intval)                                              *
*   Args : int : intval                                              *
*   desc : request interrupt from zip; waits for it, checks return    *
*****/
/*****
*   Name : q_interr(intval)                                           *
*   Args : int : intval                                              *
*   desc : same as above but quicker and quieter                     *
*****/
/*****
*
*   Name : Zipinit(s)                                                *
*   Args : s : String                                                *
*   Description : opens zip and initializes with file "s"           *
*****/
/*****
*   Name : zipout()                                                  *
*   Desc : closes zip and exits main program                        *
*****/
/*****
*
*   Name : zipcl()                                                  *
*   Args :

```

```

*      Description : close zip
*
*****/

/*****

*
*
*      Name : pad_im
*
*      Args : N16 : datlin,datcol,imcol,imlin,mask_val;  N32 : bufnum
*
*      Description : Given a datlin x datcol array it puts it in
*
*      the middle of an imlin x imcol image, bufnum. The surrounding
*
*      pixels are set to mask_val.
*
*
*****/

/*****

*
*
*      Name : extend(datlin,datcol,imlin,imcol,strtrow,strtcol,mask_val
*
*      bufnum);
*
*      Args : N16 : datlin,datcol,imcol,imlin,mask_val,strtrow,strtcol,
*
*      N32 : bufnum
*
*      Description : Given a datlin x datcol array it puts it in an
*
*      imlin x imcol image, bufnum, with the top left hand corner at
*
*      (strtrow,strtcol). The surrounding pixels are set to mask_val.
*
*
*****
*****/

```

```

extend(datlin,datcol,imlin,imcol,strtrow,strtcol,mask_val,bufnum)

N16 datcol,datlin,imlin,imcol,strtrow,strtcol,mask_val;

N32 bufnum;

{

    N32 imsiz,bufadr,nwords,dtype,try;

    N16 tab[0x1000];

    N16 i,j,k,l;

    nwords=imcol;

    for (i=0; i<0x1000 ; i++) tab[i]=mask_val;

    l=(imcol-datcol)/2;

    imsiz=imlin*imcol;

    getadr(bufnum,&bufadr);

    try=bufadr;

    /***** pad 1st lines up to strtrow *****/

    for (i=0; i< strtrow ; i++)

    {

        nwords=imcol;dtype=0;

        if ZSFAILURE(zput(tab,&bufnum,&nwords,&dtype))

        {

            printf("cannot load image0\n");

            zclose("0");

            exit(1);

        }

    }

}

```

```

        try=try+imcol;
        if ZSFAILURE(zmodbf(&bufnum,&try))
        {
            printf("cannot change buffer address\n");
            zclose("0");
            exit(1);
        }
    }
}

```

/****** enter the picture *****/

```

for (i=0; i< datlin ; i++)
{
    nwords=imcol;dtype=0;;
    for (j=0;j<datcol;j++) tab[strtcol+j]=data[i*datcol+j];
    if ZSFAILURE(zput(tab,&bufnum,&nwords,&dtype))
    {
        printf("cannot load image0\n");
        zclose("0");
        exit(1);
    }
    try=try+imcol;
    if ZSFAILURE(zmodbf(&bufnum,&try))
    {
        printf("cannot change buffer address\n");
        zclose("0");
    }
}

```

```

        exit(1);
    }
}

/***** finish padding *****/

for (i=0; i<4096 ; i++) tab[i]=mask_val;
k=imlin-(strtrow+datlin);
for (i=0; i< k ; i++)
{
    nwords=imcol;dtype=0;;
    if ZSFAILURE(zput(tab,&bufnum,&nwords,&dtype))
    {
        printf("cannot load image0\n");
        zclose("0");
        exit(1);
    }
    try=try+imcol;
    if ZSFAILURE(zmodbf(&bufnum,&try))
    {
        printf("cannot change buffer address\n");
        zclose("0");
        exit(1);
    }
}

if ZSFAILURE(zmodbf(&bufnum,&bufadr))

```

```

    {
        printf("cannot change buffer address\n");
        zclose("0");
        exit(1);
    }
}

/*****
*
* Name : capture(datlin,datcol,imlin,imcol,strtrow,strtcol,bufnum
*       data);
*
* Args : N16 : datlin,datcol,imcol,imlin,strtrow,strtcol,data[ ]
*        N32 : bufnum
*
* Description : Captures a datlin x datcol array from an
*               imlin x imcol image, bufnum, with the top left hand corner at
*               (strtrow,strtcol).
*
*
*****/

capture(datlin,datcol,imlin,imcol,strtrow,strtcol,bufnum,data)
N16 datcol,datlin,imlin,imcol,strtrow,strtcol,data[ ];
N32 bufnum;
{

    N32 imsiz,bufadr,nwords,dtype,try;
    N16 tab[0x1000];

```

```

N16 i,j;

nwords=datcol;

getadr(bufnum,&bufadr);
try=bufadr;
try=try+strtrow*imcol+strtcol;

if ZSFAILURE(zmodbf(&bufnum,&try))
{
    printf("cannot change buffer address\n");
    zclose("0");
    exit(1);
}

for (i=0; i< datlin ; i++)
{
    nwords=datcol;dtype=0;;
    if ZSFAILURE(zget(tab,&bufnum,&nwords,&dtype))
    {
        printf("cannot load image0\n");
        zclose("0");
        exit(1);
    }
}

```

```

        for (j=0;j<datcol;j++) data[datlin*i+j]=tab[j];
        try=try+imcol;
        if ZSFAILURE(zmodbf(&bufnum,&try))
        {
            printf("cannot change buffer address\n");
            zclose("0");
            exit(1);
        }
    }

    if ZSFAILURE(zmodbf(&bufnum,&bufadr))
    {
        printf("cannot change buffer address\n");
        zclose("0");
        exit(1);
    }
}

/*****
*
*      Name : auto_shift(image,buf1,imlin,imcol,target)
*
*      Args : N32 image,buf1,target; N16 imlin,imcol;
*
*      Desc : shifts image by necessary amount so as to be within
*      a factor of 2 of target
*****/
auto_shift(image,buf1,imlin,imcol,target)

```

```

N32 image,target,buf1;

N16 imlin,imcol;

{

    N16 nelets,j,k,stop,process,up,control;

    N32 max32,min32,ii;


    nelets=imlin*imcol;

    printf("before scaling image :");

    cmax32(nelets,image,buf1,&max32,&min32);

    process=1;

    up=0;

    if (max32<=0)

    {

        printf("WARNING : ORIGINAL MAXIMUM LESS THAN ZERO\n");

        printf("IMAGE WILL NOT BE SCALED\n");

        process=0;

    }

    if (max32>target)

    {

        printf("TARGET SMALLER THAN ORIGINAL MAXIMUM\n");

        printf("IMAGE WILL BE SCALED DOWN\n");

        up=1;

    }

    j=0;

    if ((process==1) && (up==0))

    {

```

```

if ((max32<=0x7fffffff) && (max32>=0))
{
    ii=max32*2;
    stop=0;
    while (stop!=1)
    {
        if (ii<0) stop=1;
        else
        {
            if ((ii>target) || (ii<0)) stop=1;
            else j=j+1;
            ii=ii*2;
        }
    }
}

printf("up_scale image");
}

if ((process==1) && (up==1))
{
    if ((max32<=0x7fffffff) && (max32>=0))
    {
        ii=max32/2;
        stop=0;
        while (stop!=1)
        {
            if (ii<target) stop=1;

```

```

        j=j+1;
        ii=ii/2;
    }
}

printf("down scale image");

}

printf("scale by j=%x\n",j);
if (up==0) control=0;
else control=1;
if (j<=15) shfl(image,imlin,imcol,control,j);
else
{
    k=j-15;
    shfl(image,imlin,imcol,control,15);
    shfl(image,imlin,imcol,control,k);
}

printf("after scaling image :");
cmax32(nelets,image,bufl,&max32,&min32);

}

/*****
*
*
*   Name : oq_auto_shift(image,bufl,imlin,imcol,target)
*
*   Args : N32 image,bufl,target; N16 imlin,imcol;
*
*   Desc : same but quicker and quieter
*
*****/

```

```

*****/

oq_auto_shift(image,bufl,imlin,imcol,target)
N32 image,target,bufl;
N16 imlin,imcol;
{
    N16 nelets,j,k,stop,process,up,control;
    N32 max32,min32,ii;

    nelets=imlin*imcol;
/*
    printf("before scaling image :");
    cmax32(nelets,image,bufl,&max32,&min32);
*/
    q_cmax32(nelets,image,bufl,&max32,&min32);
    process=1;
    up=0;
    if (max32<=0)
    {
        printf("WARNING DURING SCALING : ORIGINAL MAXIMUM \n");
        printf("LESS THAN ZERO. IMAGE WILL NOT BE SCALED\n");
        process=0;
    }
    if (max32>target)
    {
/*
        printf("TARGET SMALLER THAN ORIGINAL MAXIMUM\n");

```

```

        printf("IMAGE WILL BE SCALED DOWN\n");
*/
        up=1;
    }
    j=0;
    if ((process==1) && (up==0))
    {
        if ((max32<=0x7fffffff) && (max32>=0))
        {
            ii=max32*2;
            stop=0;
            while (stop!=1)
            {
                if (ii<0) stop=1;
                else
                {
                    if ((ii>target) || (ii<0)) stop=1;
                    else j=j+1;
                    ii=ii*2;
                }
            }
        }
    }
    if ((process==1) && (up==1))
    {
        if ((max32<=0x7fffffff) && (max32>=0))

```

```

        {
            ii=max32/2;
            stop=0;
            while (stop!=1)
            {
                if (ii<target) stop=1;
                j=j+1;
                ii=ii/2;
            }
        }
    }

/*
    printf("scale by j=%x\n",j);

*/

    if (up==0) control=0;
    else control=1;
    if (j<=15) shfl(image,imlin,imcol,control,j);
    else
    {
        k=j-15;
        shfl(image,imlin,imcol,control,15);
        shfl(image,imlin,imcol,control,k);
    }

/*

    printf("after scaling image :");
    cmax32(nelets,image,buf1,&max32,&min32);

```

```

*/

    printf("image scaled\n");
}

/*****
*
*   Name : q_auto_shift(image,bufl,imlin,imcol,target)
*
*   Args : N32 image,bufl,target; N16 imlin,imcol;
*
*   Desc : same as auto_shift but quicker and quieter and can handle
*
*   negative inputs
*****/
q_auto_shift(image,bufl,imlin,imcol,target)
N32 image,target,bufl;
N16 imlin,imcol;
{
    N16 nelets,j,k,stop,process,up,control;
    N32 max32,min32,ii;
    long test;
    long imax32,imin32;
    nelets=imlin*imcol;
/*
    printf("before scaling image :");
    cmax32(nelets,image,bufl,&max32,&min32);
*/
    q_cmax32(nelets,image,bufl,&max32,&min32);
    imax32=max32;

```

```

    imin32=min32;

    if ((imin32<0) && (imax32<-min32))
    {
        test=-imin32;

        printf("negative scaling\n");
    }

    else test=imax32;

/*

    printf("imax32 %ld(dec)%lx(hex) imin32
%ld(dec)%lx(hex)\n",imax32,imax32,imin32,imin32);    printf("test is %ld
or in hex : %lx\n",test,test);

*/

    process=1;

    up=0;

    if (test<=0)
    {

        printf("WARNING DURING SCALING : ORIGINAL MAXIMUM \n");

        printf("LESS THAN ZERO. IMAGE WILL NOT BE SCALED\n");

        process=0;

    }

    if (test>target)
    {

        printf("TARGET SMALLER THAN ORIGINAL MAXIMUM\n");

        printf("IMAGE WILL BE SCALED DOWN\n");

        up=1;

```

```

}

j=0;

if ((process==1) && (up==0))
{
    if ((test<0x40000000) && (test>0))
    {
        ii=test;
        stop=0;
        while (stop!=1)
        {
            if (ii>=0x40000000) stop=1;
            else
            {
                ii=ii*2;
                if (ii>=target) stop=1;
                else j=j+1;
            }
        }
    }
}

if ((process==1) && (up==1))
{
    ii=test;
    stop=0;
    while (stop!=1)
    {

```

```

        if (ii<target) stop=1;
        else j=j+1;
        ii=ii/2;
    }
}

/*
printf("scale by j=%x\n",j);
*/

if (up==0) control=0;
else control=1;
if (j<=15) shfl(image,imlin,imcol,control,j);
else
{
    k=j-15;
    shfl(image,imlin,imcol,control,15);
    shfl(image,imlin,imcol,control,k);
}

/*
printf("after scaling image :");
cmax32(nelets,image,buf1,&max32,&min32);
*/

printf("image scaled\n");
}

```

```

/*****
*
*
*   Name : cmax16
*
*   Args : N16:nelets; N32: inbuff,outbuff; N16:pmax,pmin
*
*   Description: finds max and min of the first nelets words
*
*   in inbuff, puts them in outbuff and brings them back in tab,
*
*   and print their values
*
*
*****/

```

```

cmax16(nelets,inbuff,outbuff,pmax,pmin)
N16 nelets,*pmin,*pmax;
N32 inbuff,outbuff;
{
    N32 argcount;
    N16 arg[10];
    N16 tab[10];
    N32 dtype;
    N32 nwords;

    argcount=3;
    arg[0]=nelets;
    arg[1]=(N16)inbuff;
    arg[2]=(N16)outbuff;
    if ZSFAILURE(ztcall("vmax16a",&argcount,arg))
    {

```

```

        printf("cannot find max\n");

        zclose("0");

        exit(1);
    }

    nwords=8;

    dtype=0;

    if ZSFAILURE(zget(tab,&outbuff,&nwords,&dtype))
    {
        printf("cannot get max\n");

        zclose("0");

        exit(1);
    }

    *pmax=tab[0];

    *pmin=tab[1];

    printf("max= %x, min=%x (in hex) \n",*pmax,*pmin);
}

/*****

*
*
*   Name : q_cmax16
*
*   Args : N16:nelets; N32: inbuff,outbuff; N16:pmax,pmin
*
*   Description: same as above but doesn't print values
*
*****/

q_cmax16(nelets,inbuff,outbuff,pmax,pmin)
N16 nelets,*pmin,*pmax;

```

```

N32 inbuff,outbuff;

{

    N32 argcount;

    N16 arg[10];

    N16 tab[10];

    N32 dtype;

    N32 nwords;


    argcount=3;
    arg[0]=nelets;
    arg[1]=(N16)inbuff;
    arg[2]=(N16)outbuff;
    if ZSFAILURE(ztcall("vmax16a",&argcount,arg))
    {
        printf("cannot find max\n");
        zclose("0");
        exit(1);
    }
    nwords=8;
    dtype=0;
    if ZSFAILURE(zget(tab,&outbuff,&nwords,&dtype))
    {
        printf("cannot get max\n");
        zclose("0");
        exit(1);
    }
}

```

```

    *pmax=tab[0];

    *pmin=tab[1];

    printf("cmax16\n");

}

```

```

/*****

```

```

*
*
*   Name : cmax32
*
*   Args : N16: nelets; N32: inbuff,outbuff; N32:pmax,pmin
*
*   Description: finds max and min of the first nelets words
*
*   in inbuff, puts them in outbuff and brings them back in tab,
*
*   and print their values
*
*

```

```

*****/

```

```

cmax32(nelets,inbuff,outbuff,pmax,pmin)

```

```

N16 nelets;

```

```

N32 inbuff,outbuff,*pmin,*pmax;

```

```

{

```

```

    N32 argcount;

```

```

    N16 arg[10];

```

```

    N32 tab[10];

```

```

    N32 dtype;

```

```

    N32 nwords;

```

```

        argcount=3;
arg[0]=nelets;
arg[1]=(N16)inbuff;
arg[2]=(N16)outbuff;
if ZSFAILURE(ztcall("vmax32a",&argcount,arg))
{
    printf("cannot find max\n");
    zclose("0");
    exit(1);
}
nwords=8;
dtype=1;
if ZSFAILURE(zget(tab,&outbuff,&nwords,&dtype))
{
    printf("cannot get max\n");
    zclose("0");
    exit(1);
}
*pmax=tab[0];
*pmin=tab[1];
printf("max= %lx, min=%lx (in hex) \n",*pmax,*pmin);
}

```

```

/*****
*
*   Name : q_cmax32
*
*   Args : N16: nelets; N32: inbuff,outbuff; N32:pmax,pmin
*
*   Description: same as above but doesn't print values
*
*****/

q_cmax32(nelets,inbuff,outbuff,pmax,pmin)

N16 nelets;
N32 inbuff,outbuff,*pmin,*pmax;

{
    N32 argcount;

    N16 arg[10];

    N32 tab[10];

    N32 dtype;

    N32 nwords;


    argcount=3;

    arg[0]=nelets;

    arg[1]=(N16)inbuff;

    arg[2]=(N16)outbuff;

    if ZSFAILURE(ztcall("vmax32a",&argcount,arg))

    {

        printf("cannot find max\n");

        zclose("0");

        exit(1);
    }
}

```

```

    }

    nwords=8;

    dtype=1;

    if ZSFAILURE(zget(tab,&outbuff,&nwords,&dtype))

    {

        printf("cannot get max\n");

        zclose("0");

        exit(1);

    }

    *pmax=tab[0];

    *pmin=tab[1];

    printf("cmax32\n");

}

/*****

*
*
*   Name : getadr;
*
*   args : N32 : bufnum , pbufadr
*
*   Description : given a buffer number, getadr puts its address
*
*   in *pbufadr
*
*
*****/

getadr(bufnum,pbufadr)
N32 bufnum,*pbufadr;
{

```

```

N16 tab[10],loadr,hiadr;

N32 nwords,dtype;

char lbuf[80],hbuf[80],**endptr,buf[80],buf1[80],buf2[80];

int base,i,j;

/*****transform N32 bufnum to N16 loadr,hiadr*****/

base=2;

ltoa(bufnum,buf,base);

j=strlen(buf);

if (j<17)

{

    hiadr=0;loadr=bufnum;

}

else

{

    sprintf(lbuf,"%s",buf+j-16);

    for (i=0;i<j-16;i++) sprintf(buf1+i,"%c",buf[i]);

    if (j<32)

    {

        for (i=0;i<32-j;i++) sprintf(buf2+i,"%c",'0');

        sprintf(hbuf,"%s%s",buf2,buf1);

    }

    loadr=strtol(lbuf,endptr,base);

    hiadr=strtol(hbuf,endptr,base);

```

```

    }

    /*****peek hiadr,loadr, decipher buffer address*****/

    dtype=0;
    nwords=4;
    if ZSFAILURE(zmpeek(tab,&loadr,&hiadr,&nwords,&dtype))
    {
        printf("cannot get address\n");
        zclose("0");
        exit(1);
    }
    base=2;
    itoa(tab[0],buf1,base);
    j=strlen(buf1);
    for (i=0;i<16-j;i++) sprintf(buf2+i,"%c",'0');
    if (j<16)
        sprintf(lbuf,"%s%s",buf2,buf1);
        else sprintf(lbuf,"%s",buf1);
    itoa(tab[2],buf1,base);
    j=strlen(buf1);
    if (j>7)
        sprintf(hbuf,"%s",buf1+j-7);
        else sprintf(hbuf,"%s",buf1);
    sprintf(buf,"%s%s",hbuf,lbuf);
    *pbufadr=strtol(buf,endptr,base);
}

```

```

/*****

*
*      Name : ctrans
*
*      Args : s : 80 char string N32 *pbufnum
*
*      Description : translates buffer s to N32 bufnum
*
*****/

ctrans(s,pbufnum)

char s[80];
N32 *pbufnum;
{
    N32 dummy,dummy1;
    if ZSFAILURE(ztrans(s,&dummy,&dummy1))
    {
        printf("cannot translate buffer\n");
        zclose("0");
        exit(1);
    }
    *pbufnum=dummy1;
}

/*****
*
*

```

```

*      Name : ftrans                                          *
*      Args : s : 80 char string N32 *povly,*pentpt        *
*      Description : translates function s to N32 overlay,entry_point *
*****/

ftrans(s,povly,pentpt)

char s[80];

N32 *povly,*pentpt;

{
    N32 aovly,aentpt;
    if ZFAILURE(ztrans(s,&aovly,&aentpt))
    {
        printf("cannot translate acquire");
        zclose("0");
        exit(1);
    }
    else printf("%s translated\n",s);
    *povly=aovly;
    *pentpt=aentpt;
}

/*****

*      Name : interr(intval)                                  *
*      Args : int : intval                                    *
*      desc : request interrupt from zip; waits for it, checks return *
*****/

interr(intval)

```

```

N16 intval;

{

    int rc;

    N16 retval;

    if (ZSFAILURE(rc=zint(&intval)))

    {

        printf("zint failure, rc = %d\n");

        zclose("0");

        exit(1);

    }

    else printf("interrupt request sent to zip\n");

    if (ZSFAILURE(rc=zwint(&retval)))

    {

        printf("zwint failure, rc = %d\n");

        zclose("0");

        exit(1);

    }

    else printf("interrupt received from zip\n");

    if (retval != 0x8000 + intval)

    {

        printf(" unexpected interrupt register value.\n");

        printf(" expected = %x, actual = %x\n",0x8000+intval,retval);

        zclose("0");

        exit(1);

    }

}

```

```

        else printf("interrupt value = %x is ok\n",retval);
    }

/*****
*   Name : q_interr(intval)                                     *
*   Args : int : intval                                         *
*   desc : same as above but quicker and quieter               *
*****/
q_interr(intval)
N16 intval;
{
    int rc;
    N16 retval;

    if (ZSFAILURE(rc=zint(&intval)))
    {
        printf("zint failure, rc = %d\n");
        zclose("0");
        exit(1);
    }
    if (ZSFAILURE(rc=zwint(&retval)))
    {
        printf("zwint failure, rc = %d\n");
        zclose("0");
        exit(1);
    }
}

```

```

    if (retval != 0x8000 + intval)
    {
        printf(" unexpected interrupt register value.\n");
        printf(" expected = %x, actual = %x\n",0x8000+intval,retval);
        zclose("0");
        exit(1);
    }

    printf("interrupt\n");
}

/*****
*
*      Name : Zipinit(s)
*
*      Args : s : String
*
*      Description : opens zip and initializes with file "s"
*
*****/
Zipinit(s)
char s[];
{
    if ZSFAILURE(zopen("0"))
    {
        printf("cannot open ZIP\n");
        exit(1);
    }

    else printf("zip open\n");
}

```

```

        if ZSFAILURE(zinit(s))
        {
            printf("cannot download %s\n",s);
            zclose(0);
            exit(1);
        }
        else printf("%s downloaded\n",s);
    }

/*****
*   Name : zipout()                                     *
*   Desc : closes zip and exits main program           *
*****/
zipout()
{
    zipcl();
    exit(1);
}

/*****
*
*   Name : zipcl()                                     *
*   Args :                                             *
*   Description : close zip                           *
*****/
zipcl()

```

```
{  
  
    int rc;  
  
    rc=zcclose("0");  
  
    if (rc !=0)  
    {  
  
        printf("Cannot close zip\n");  
  
        exit(1);  
  
    }  
  
    else  
  
        printf("Zip has been closed\n");  
  
}
```

INUTIL.C

```

#include <c:\c\stdio.h>

#include <c:\driver\mcshdr.h>

/*****

*
*      Name : summary(filnam,nb)
*      Args : char filnam; N16 nb
*      desc : opens file filnam and prints angle values to screen
*****/

/*****

*      Name : file_buf(showbuf,prows,pcolumns,pstrtrrow,pstrtrcol,data)
*      desc : brings data from file to buffer showbuf
*****/

/*****

*
*      Name : y_or_n(quest)
*      Args : char quest[]
*      Desc : prompts quest returns 1 for yes, 0 else;
*****/

/*****

*
*      Name : dy_or_n(quest,def)
*      Args : char quest[], int def
*      Desc : prompts quest returns 1 for Y,y, 0 for N,n, def for <cr>
*****/

/*****

*

```

```

*   Name : get16(name,pval)
*
*   Args : N16 val ; char name[]
*
*   Desc : prompts name; gets N16 value to pointer to val (pval)
*
*****/
/*****
*
*   Name : get32(name,pval)
*
*   Args : N32 val ; char name[]
*
*   Desc : prompts name; gets N32 value to pointer to val (pval)
*
*****/
/*****
*
*   Name : getint(name,pval)
*
*   Args : int *pval ; char name[]
*
*   Desc : prompts name; gets N16 value to pointer to val (pval)
*
*****/
/*****
*
*   Name : getstr(name,pval)
*
*   Args : N16 val ; char name[]
*
*   Desc : prompts name; gets str value to pointer to val (pval)
*
*****/
/*****
*

```

```

*   Name : summary(filnam,nb)                                     *
*   Args : char filnam; N16 nb                                   *
*   desc : opens file filnam and prints angle values to screen  *
*****/

```

```

summary(filnam,nb)
char filnam[];
unsigned int nb;
{
    FILE *fp,*fopen();
    int i,run;
    double theta;
    double r_abs[4],r_ord[4];
    char ch[1],chl[1];

    printf("summary of results\n");
    fp=fopen(filnam,"r");
    fscanf(fp,"%c",&ch[0]);
    printf("%s",ch);
    sprintf(chl,"#");
    while (strcmp(ch,chl) != 0)
    {
        fscanf(fp,"%c",&ch[0]);
    }
}

```

```

        printf("%s",ch);
    }

    fscanf(fp,"%d : %lf %lf %lf %lf %lf %lf %lf %lf
%lf\n",&run,&r_abs[0],&r_ord[0],&r_abs[1],&r_ord[1],&r_abs[2],&r_ord[2],&r_
abs[3],&r_ord[3],&theta);    printf("theta[0] : %3.3lf\n",theta);

    for (i=0;i<nb;i++)
    {
        fscanf(fp,"#%d : %lf %lf %lf %lf %lf %lf %lf %lf
%lf\n",&run,&r_abs[0],&r_ord[0],&r_abs[1],&r_ord[1],&r_abs[2],&r_ord[2],&r_
abs[3],&r_ord[3],&theta);    printf("theta[%d] = %3.3lf\n",run,theta);
    }

    fclose(fp);
}

/*****
*   Name : file_buf(showbuf,prows,pcolumns,pstrtrow,pstrtcol,data)   *
*   desc : brings data from file to buffer showbuf                   *
*****/
file_buf(showbuf,prows,pcolumns,pstrtrow,pstrtcol,data)
N32 showbuf;
N16 *prows,*pcolumns,*pstrtrow,*pstrtcol,data[];
{

```

```

N16 rows,columns,strtrow,strtcol;

N32 lrows,lcolumns,nwords,length,extra,getwds,entry,dtype;

N32 bufadr,nbufadr;

int i,j,intval;

char filnam[80];

FILE *fp,*fopen();


dtype=0;

getadr(showbuf,&bufadr);

nbufadr=bufadr;

printf("image will be recovered from a file\n");

getstr("file name",filnam);

printf("the file is %s\n",filnam);

fp=fopen(filnam,"r");

fscanf(fp,"%5x %5x %5x %5x \n",&rows,&columns,&strtrow,&strtcol);

lrows=rows;

lcolumns=columns;

nwords=lrows*lcolumns;

printf("rows = %lx cols = %lx words=%lx\n",lrows,lcolumns,nwords);

length=nwords/0x4000;

extra=nwords-length*0x4000;

getwds=0x4000;

printf("length %lx extra %lx\n",length,extra);

printf("acquiring image from file\n");

for (entry=0; entry< length;entry++)

{

```

```

printf("iteration # %lx of %lx\n",entry,length);
printf("entry = %lx \n",entry);
for (i=0;i<0x800;i++)
{
    for (j=0;j<0x8;j++)
        fscanf(fp,"%5x",&data[0x8*i+j]);
    fscanf(fp,"\n");
}
intval=1;
q_interr(intval);
printf("will put\n");
zput(data,&showbuf,&getwds,&dtype);
zwdma();
nbufadr=nbufadr+0x4000;
zmodbf(&showbuf,&nbufadr);
}
for (i=0;i<extra/0x8;i++)
{
    for (j=0;j<0x8;j++) fscanf(fp,"%5x",&data[0x8*i+j]);
    fscanf(fp,"\n");
}
zput(data,&showbuf,&extra,&dtype);
zmodbf(&showbuf,&bufadr);
fclose(fp);
*prows=rows;
*pcolumns=columns;

```

```

        *pstrtrow=strtrow;

        *pstrtcol=strtcol;
    }

sdat_file(filnam,srows,scolumns,strtrow,strtcol,snwords,data)
char filnam[];
N16 srows,scolumns,strtrow,strtcol,data[];
N32 snwords;
{
    FILE *fp,*fopen();

    int i,j;

    fp=fopen(filnam,"w");
    fclose(fp);
    fp=fopen(filnam,"a");
    fprintf(fp,"%5x %5x %5x %5x \n",srows,scolumns,strtrow,strtcol);
    for (i=0;i<(N16)snwords/8;i++)
    {
        for (j=0;j<0x8;j++)
            fprintf(fp,"%5x",data[0x8*i+j]);

        fprintf(fp,"\n");
    }
    fclose(fp);
}

```

```

/*****
*
*      Name : y_or_n(quest)
*
*      Args : char quest[]
*
*      Desc : prompts quest returns 1 for yes, 0 else;
*****/

y_or_n(quest)
char quest[];
{
    char message[100],val[100];
    int good_val,i;

    good_val=0;
    while (good_val !=1)
    {
        getstr(quest,val);
        switch (val[0])
        {
            case 'y':
            case 'Y': good_val=1;
                    i=1;
                    break;

            case 'n':
            case 'N': good_val=1;
                    i=0;
                    break;
        }
    }
}

```

```

        default : printf("please choose Y,y,N, or n\n");

    }

}

return(i);

}

/*****
*
*   Name : dy_or_n(quest,def)
*   Args : char quest[], int def
*   Desc : prompts quest returns 1 for Y,y, 0 for N,n, def for <cr>
*****/
dy_or_n(quest,def)
char quest[];
int def;
{
    char message[100],val[100];

    int i,inpt;

    i=def;

    printf(quest);

    inpt=getch();printf("%c\n",inpt);

    while ((inpt!=89) && (inpt!=121) && (inpt!=78) && (inpt!=110) &&
(inpt!=13))    {

        printf(quest);

        inpt=getch();

```



```

*   Name : get16(name,pval)                                     *
*   Args : N16 val ; char name[]                               *
*   Desc : prompts name; gets N16 value to pointer to val (pval) *
*****/
get16(name,pval)
char name[];
N16 *pval;
{
    N16 dummy;
    char message[100];
    dummy=0;
    sprintf(message,"%s (in hex) : ",name);
    printf(message);
    scanf("%x",&dummy);
    *pval=dummy;
}

/*****
*
*   Name : get32(name,pval)                                     *
*   Args : N32 val ; char name[]                               *
*   Desc : prompts name; gets N32 value to pointer to val (pval) *
*****/
get32(name,pval)
char name[];
N32 *pval;

```

```

{
    N32 dummy;

    char message[100];

    dummy=0;

    sprintf(message,"%s (in hex) : ",name);

    printf(message);

    scanf("%lx",&dummy);

    *pval=dummy;
}

/*****
*
*      Name : getstr(name,val)
*
*      Args : char val[] ; char name[]
*
*      Desc : prompts name; gets str value to pointer to val (pval)
*****/
getstr(name,val)
char name[];
char val[];
{
    char message[100];

    sprintf(message,"%s : ",name);

    printf(message);

    scanf("%s",val);
}

```

```

find_max(pmax,pindex,elets,data)
N16 *pmax,*pindex,elets,data[];
{
    N16 i,j,k;

    i=0;
    j=0;
    k=0;
    for (i=0;i<elets;i++)
        if (data[i]>k) {j=i;k=data[i];}
    printf("maximum : data[%x]=%x (in hex)\n",j,k);
    *pmax=k;
    *pindex=j;
}

```

PREP.C

```

#include "c:\c\stdio.h"

#include "c:\driver\mcshdr.h"

#include "c:\driver\import.h"

/*****

*   Name : acq_landmarks(datlin,land,acq_col,acq_row,data,          *
*   buf1,acqovly,acqentpt,nwords,dtype,nelets,showbuf,bufadr,invert, *
*   do_mask,thresh,binary,mexh,r_avg,imlin,strtrow,strtcol,      *
*   mask_val,regions,tland,tmh,wkbuf,lcoef,cmultgt,im_top,fftovly, *
*   fftentpt)                                                    *
*                                                                    *
*   Args : N16 datlin : lines, columns in subpict                *
*   N32 land : array of subpicts (landmarks) to be acquired      *
*   int acq_col,acq_row : array of coords of sub_picts           *
*   N16 data : array                                              *
*   N32 buf1 : buffer for cmax                                    *
*   N32 acqovly,acqentpt : overlay,entry point of "acquire"      *
*   N32 nwords,dtype : # words, data type for zip transfers      *
*   N16 nelets =nwrods                                           *
*   N32 showbuf,bufadr : buffer for display, and its address     *
*   N16 invert,do_mask,thresh,binary,mexh : options              *
*   N16 r_avg : average of rois;                                  *
*   N16 imlin : lines, columns in correlation fct                *
*   N16 strtrow,strtcol : startrow, column for image in          *
*   buffer land                                                  *

```

```

*      N16 mask_val : value of area around image in buffer land      *
*      N16 regions : # of subpictures                                *
*      N32 tland : buffer for fft's if mhf is done                  *
*      N32 tmh,lcoef,wkbuf : buffers for mhf                        *
*      N32 cmultgt,im_top : target for shifts if fft's are done     *
*      N32 fftovly, ffttentpt : entryptpoint, overlay for fft      *
*                                                                    *
*      Desc : Acquires and does preliminary filtering of test subpics *
*** *****/

```

```

acq_landmarks(datlin,land,acq_col,acq_row,data,buf1,acqovly,acqentpt,nwords-
, dtype,nelets,showbuf,bufadr,invert,do_mask,thresh,binary,mexh,r_avg,imlin,-
strtrow,strtcol,mask_val,regions,tland,tmh,wkbuf,lcoef,cmultgt,im_top,fftov-
ly,fftentpt) N16

```

```

datlin,data[],imlin,strtcol,strtrow,mask_val,nelets,r_avg[];

```

```

N16 invert,do_mask,thresh[],mexh,binary;

```

```

N32 land[],buf1,acqovly,acqentpt,nwords,dtype,showbuf,bufadr;

```

```

N32 tland,tmh,wkbuf,lcoef,cmultgt,im_top,fftovly,fftentpt;

```

```

int acq_col[],acq_row[],regions;

```

```

{

```

```

    int loop,i,j,intval,imax16,imin16;

```

```

    N16 max16,min16,datcol;

```

```

    datcol=datlin;

```

```

    q_n_grab(regions,datlin,datcol,land,acq_col,acq_row,acqovly,acqentpt);

```

```

for (loop=0;loop<regions;loop++)
{
    zget(data,&land[loop],&nwords,&dtype);
    printf("land[%d] :",loop);
    intval=1;
    q_interr(intval);
    if (invert==1) /*&& (loop<2))*/
        for (i=0; i<nelets ; i++) data[i]=0x7fff-data[i];
    if (do_mask==1)
    {
        if (binary==1)
            for (i=0; i<nelets ; i++)
                if (data[i]<thresh[loop]) data[i]=0;
                else data[i]=0x7fff;
        else
            for (i=0; i<nelets ; i++)
                if (data[i]<thresh[loop]) data[i]=0;
    }

    if (datlin==imlin) zput(data,&land[loop],&nwords,&dtype);
    else
        extend(datlin,datcol,imlin,imlin,strtrow,strtcol,mask_val,land[loop]);
    if ((mexh==1) && (loop<2))

```

```
mexfilt(land[loop],tland,tmh,wkbuf,lcoef,cmultgt,im_top,buf1,imlin,imlin,ne-
lets,fftovly,fftentpt,data,nwords,dtype);          /*****DC component*****/
```

```
/*      printf("landmark, before dc:\n");
      cmax16(nelets,land[loop],buf1,&max16,&min16);
*/
```

```
      zget(data,&land[loop],&nwords,&dtype);
      for (i=0;i<nelets;i++) data[i]=data[i]-r_avg[loop];
      zput(data,&land[loop],&nwords,&dtype);
```

```
/*      cmax16(nelets,land[loop],buf1,&max16,&min16);
      imax16=max16;
      imin16=min16;
      printf("integer max %d min %d\n",imax16,imin16);
```

```
*/
```

```
/*
```

```
      zget(data,&land[loop],&nwords,&dtype);
      display(imlin,imlin,showbuf,bufadr,data);
```

```
*/
```

```
    }
```

```
}
```

```
/**
*****
```

```
*      Name : acq_roi(datlin,roi,acq_col,acq_row,data,buf1,
```

```
*)
```

```

*   acqovly,acqentpt,nwords,dtype,nelets,showbuf,bufadr,screen,      *
*   scradr,invert,do_mask,thresh,binary,mexh,r_avg,imlin,          *
*   strtrow,strtcol,mask_val,regions,hand,upline,dnline,uproi,dnroi, *
*   troi,tmh,wkbuf,lcoef,cmultgt,im_top,fftovly,fftentpt)          *
*                                                                    *
*                                                                    *
*   Args : N16 datlin : lines, columns in subpict                  *
*   N32 roi : array of reference picts to be acquired               *
*   int acq_col,acq_row : array of coords of sub_picts              *
*   N16 data : array                                                *
*   N32 buf1 : buffer for cmax                                       *
*   N32 acqovly,acqentpt : overlay,entry point of "acquire"        *
*   N32 nwords,dtype : # words, data type for zip transfers         *
*   N16 nelets =nwrods                                              *
*   N32 showbuf,bufadr : buffer for display, and its address        *
*   N32 screen, : buffer containing whole screen;                  *
*   N32 scradr address of above;                                     *
*   N16 invert,do_mask,thresh,binary,mexh : options                *
*   N16 r_avg[] : average value of roi                              *
*   N16 imlin : lines, columns in correlation fct                  *
*   N16 strtrow,strtcol : startrow, column for image in            *
*   buffer land                                                      *
*   N16 mask_val : value of area around image in buffer land        *
*   N16 regions : # of subpictures                                  *
*   N16 hand : option for user interface for acquisition            *
*   N32 upline,dnline,uproi,dnroi : buffers for video acquis       *

```

```

*      N32 troi : array of buffers for fft's if mhf is done      *
*      N32 tmh,lcoef,wkbuf : buffers for mhf                    *
*      N32 cmultgt,im_top : target for shifts if fft's are done  *
*      N32 fftovly, fftentpt : entrypt, overlay for fft         *
*                                                                *
*      Desc : Acquires and does preliminary filtering of reference *
*      subpicts, and extracts ROIs                               *
*****/

```

```

/***** acquire ROI (Regions of interest) *****/

```

```

acq_roi(fp,datlin,roi,acq_col,acq_row,data,bufl,acqovly,acqentpt,nwords,dtype,
nelets,showbuf,bufadr,screen,scradr,invert,do_mask,thresh,binary,mexh,r_avg,
imlin,strtrow,strtcol,mask_val,regions,hand,upline,dnline,uproi,dnroi,troi,
tmh,wkbuf,lcoef,cmultgt,im_top,ftovly,ftentpt) N16
datlin,data[],imlin,strtcol,strtrow,mask_val,nelets,r_avg[];
N16 invert,do_mask,thresh[],mexh,binary,hand;
N32 roi[],bufl,acqovly,acqentpt,nwords,dtype,showbuf,bufadr,screen,scradr;
N32 upline,dnline,uproi,dnroi;
N32 troi[],tmh,wkbuf,lcoef,cmultgt,im_top,ftovly,ftentpt;
int acq_col[],acq_row[],regions;
FILE *fp;
{
    N32 hglin,hgcol,mli[4],mcol[4],roisz;

```

```

N16 lin,col,caprow,capcol,max16,min16,scriin,scrcol,scstrow,scstcol;

N16 tval,snelets,rnelets,datcol,pct[4];

int loop,i,dummy;

/***** set initial parameters, grab screen for acquisition **/

    datcol=datlin;

    snelets=datlin * datcol;

    lin=0x20;

    col=0x20;

    roisz=lin*col;

init_face(&hglin,&hgcol,m1in,mcol,&scriin,&scrcol,&scstcol,&scstrow,dat-
col,regions);  if (y_or_n("Acquire Landmarks ?")) ;

    grab(scriin,scrcol,screen,scstcol,scstrow,acqovly,acqentpt);

/***** loop on landmarks *****/

    for (loop=0;loop<regions;loop++)

    {

    do

    {

/***** find image coordinates *****/

        if (hand==1)

        {

```

```

face(&m1in[loop],&mcol[loop],datlin,datcol,hglin,hgcol,upline,uproi,dnline,-
dnroi,screen,scradr,dtype);          acq_col[loop]=mcol[loop];

        acq_row[loop]=m1in[loop];

        printf("acquired at line %d column
%d\n",acq_row[loop],acq_col[loop]);          )

    else

    {

        getint("acquire at row",&acq_row[loop]);

        getint("acquire at column",&acq_col[loop]);

    }

/***** grab and display sub-picture *****/

re_grab(datlin,datcol,roi[loop],acq_col[loop],acq_row[loop],data,buf1,acqov-
ly,acqentpt);          display(datlin,datcol,showbuf,bufadr,data);

/***** pre-filter and extend to im1in x imcol *****/

/***** invert *****/

if (invert==1)

    for (i=0; i<snelets ; i++) data[i]=0x7fff-data[i];

/***** threshold *****/

```

```

if (do_mask==1)
{
    printf("sub-picture[%d] :",loop);
    cmax16(snelets,roi[loop],buf1,&max16,&min16);
    printf("range is %d\n",max16-min16);
    getint("percent of range to set to zero :",&tval);
if (invert==1) thresh[loop]=0x7fff-min16-tval*((max16-min16)/100);
    else thresh[loop]=min16 + tval * ((max16-min16)/100);
    printf("thresh[%d]=%d\n",loop,thresh[loop]);
    pct[loop]=tval;
    if (binary==1)
        for (i=0; i<snelets ; i++)
            if (data[i]<thresh[loop]) data[i]=0;
            else data[i]=0x7fff;
    else
        for (i=0; i<snelets ; i++)
            if (data[i]<thresh[loop]) data[i]=0;

}

/***** display if processed *****/

if ((invert==1)|| (do_mask==1))
{
    zput(data,&roi[loop],&nwords,&dtype);

```

```

        zwdma();

        printf("roi[%d] :",loop);

        cmax16(snelets,roi[loop],buf1,&max16,&min16);

        display(datlin,datcol,showbuf,bufadr,data);

    }

/***** Mexican Hat Filter *****/

    if ((mexh==1) && (loop<2))
    {

mexfilt(roi[loop],troi[loop],tmh,wkbuf,lcoef,cmultgt,im_top,buf1,imlin,im-
lin,nelets,fftovly,fftentpt,data,nwords,dtype);

        zget(data,&roi[loop],&nwords,&dtype);

        zwdma();

        printf("roi[%d] :",loop);

        cmax16(nelets,roi[loop],buf1,&max16,&min16);

        display(imlin,imlin,showbuf,bufadr,data);

    }

/***** extract landmarks place in imlin x imco image **/

    printf("extracting landmark\n");

    caprow=(datlin-lin)/2;

    capcol=(datcol-col)/2;

    capture(lin,col,datlin,datcol,caprow,capcol,roi[loop],data);

```

```

/*****DC *****/

rnelets=lin*col;

f_avg(rnelets,data,&r_avg[loop]);

for (i=0;i<rnelets;i++) data[i]=data[i]-r_avg[loop];

caprow=(imlin-lin)/2;

capcol=(imlin-col)/2;

/****place in image *****/

extend(lin,col,imlin,imlin,caprow,capcol,mask_val,roi[loop]);

zget(data,&roi[loop],&nwords,&dtype);

zwdma();

printf("roi[%d] :",loop);

cmax16(nelets,roi[loop],buf1,&max16,&min16);

display(imlin,imlin,showbuf,bufadr,data);

}

while (!(y_or_n("satisfactory image ?")));

}

if (do_mask)

{

    for (i=0;i<regions;i++) fprintf(fp,"pct[%d]=%d ",i,pct[i]);

    fprintf(fp,"\n");

}

}

```

```

/*****
*   Name : f_avg(snelets,data,pavg)                                     *
*   Args : N16 snelets,data[],*pavg                                   *
*   Desc : takes first snelets elements of array data, and finds     *
*           their average. This is stored in *pavg                     *
*****/
f_avg(snelets,data,pavg)
N16 snelets,data[],*pavg;
{
    int i,idummy;
    long dummy;
    double rsnelets,ravg,rdummy;

    dummy=snelets;
    rsnelets=dummy;
    ravg=0;
    for (i=0;i<snelets;i++)
    {

        idummy=data[i];
        dummy=idummy;
        rdummy=dummy;
        ravg=ravg+rdummy/rsnelets;
    }
    if (ravg>32767.0)
    {

```

```

        printf("ERROR AVERAGE TOO LARGE average= %lf\n",ravg);
        ravg=32767.0;
    }
/*    if (ravg<0.0)
    {
        printf("ERROR : AVERAGE TOO SMALL\n");
        ravg=0.0;
    }
*/
    dummy=ravg;
    *pavg=dummy;
    printf("elets : %lf avg %lf avg %d\n",rsnelets,ravg,*pavg);
}

```

VIDLIB.C

```

/*****
*   Name : init_vid(hres,lut)
*
*   Args : int hres; N32 lut;
*
*   Description : sets up video; linear ramp lookup table
*
*****/

/*****
*   Name : good_im(datlin,datcol,eye,strtcol,strtrow,data,bufl
*
*   Args : N16 datlin,datcol,strtcol,strtrow,data[]; N32 eye,bufl
*
*   Description : grabs dalin* datcol image starting at (strtrow,
*   strtcol) and puts in eye; repeats until neither middle line is
*   zero
*
*****/

/*****
*
*
*   Name : getim
*
*   Args : N16 rows,columns; N32 image0; N16 strtcol,strtrow
*
*   Description : acquires a rows*columns image into image0. The
*   image starts (upper left hand corner) at (strtcol,strtrow)
*
*****/

/*****
*
*   Name : re_grab(datlin,datcol,eye,strtcol,strtrow,data,bufl,
*   acqovly,acqentpt)

```

```

*   Args : N16 datlin,datcol,strtcol,strtrow,data[]; N32 eye,buf1      *
*           N32 acqovly,acqentpt;                                     *
*   Description : grabs dalin* datcol image starting at (strtrow,      *
*   strtcol) and puts in eye; repeats until neither middle line is    *
*   zero; there is no user interface, and zcall not ztcall is used    *
*****/
/*****
*   Name : n_grab(nb,datlin,datcol,eye,strtcol,strtrow,data,buf1,      *
*   acqovly,acqentpt)                                                 *
*   Args : N16 nb,datlin,datcol,strtcol[],strtrow[],data[];          *
*           N32 eye[],buf1,acqovly,acqentpt;                          *
*   Description : based on re_grab, grabs nb datlin* datcol images    *
*   starting at (strtrow[i],strtcol[i]) and puts in eye[i]           *
*   re_grabs all images until neither middle line is zero on any      *
*   image; there is no user interface, and zcall not ztcall is used   *
*****/
/*****
*
*   Name :grab(rows,columns,image0,strtcol,strtrow,acqovly,acqentpt)  *
*   Args : N16 rows,columns; N32 image0; N16 strtcol,strtrow ;        *
*           N32 acqentpt,acqovly                                       *
*   Description : acquires a rows*columns image into image0. The      *
*   image starts (upper left hand corner) at (strtcol,strtrow).this   *
*   program uses zcall instead of ztcall, and has no user interface   *
*

```

```

*****/
/*****
*
*      Name: display
*
*      Args: N16 : rows,columns, N32 bufnum,bufadr N16 :data[];
*
*      Description : displays the image stored in the "data" array
*      but first pads with zeros the first roffset lines and coffset
*      columns so that all the elements in data actually appear on the
*      screen
*
*****/
/*****
*
*      Name: show(rows,columns,image0)
*
*      Args: N16 : rows,columns, N32 image0
*
*      Description : displays the image stored in image0; first 16
*      rows and 32 columns are not visible
*
*****/
/*****
*
*      Name: mon(rows,columns,image0)
*
*      Args: N16 : rows,columns, N32 image0
*
*      Description : displays the image stored in image0; first 16
*      rows and 32 columns are not visible. Same as show, but not

```

```

*      interactive; use with cl_mon();
*
*****/
/*****
*
*
*      Name: cl_mon()
*
*      Args:      ()
*
*
*      Description : clears monitor,enabling zip; use with
*
*      mon(rows,columns,image0)
*
*****/

```

```

*****/

init_vid(hres,lut)
N32 lut;
int hres;
{
    int bufsz,init,incr,blank;

    bufsz=0x100;init=0;incr=0x80;
    cfill(bufsz,init,incr,lut);
    blank=1;
    cvsetup(hres,blank);
    cload(lut,0);
    cload(lut,1);
    printf("Video initialized\n");
}

cfill(bufsz,init,incr,lut)
int bufsz,init,incr;
N32 lut;
{
    N16 arg[10];
    N32 argcount;

    arg[0]=bufsz;
    arg[1]=init;
    arg[2]=incr;

```

```

    arg[3]=(N16)lut;
    argcount=4;
    if ZSFAILURE(ztcall("FILL",&argcount,arg))
    {
        printf("cannot translate fill\n");
        zclose("0");
        exit(1);
    }
}

cvsetup(hres,blank)
int hres,blank;
{
    N16 arg[10];
    N32 argcount;

    arg[0]=hres;
    arg[1]=blank;
    argcount=2;
    if ZSFAILURE(ztcall("VSETUP",&argcount,arg))
    {
        printf("cannot translate vsetup\n");
        zclose("0");
        exit(1);
    }
}

```

```

cload(lut,i)
N32 lut;
int i;
{
    N16 arg[10];
    N32 argcount;

    arg[0]=(N16)lut;
    arg[1]=i;
    argcount=2;
    if ZSFAILURE(ztcall("LOAD",&argcount,arg))
    {
        printf("cannot translate load\n");
        exit(1);
    }
}

```

```

/*****

```

```

*   Name : good_im(datlin,datcol,eye,strtcol,strtrow,data,buf1      *
*   Args : N16 datlin,datcol,strtcol,strtrow,data[]; N32 eye,buf1  *
*   Description : grabs dalin* datcol image starting at (strtrow,  *
*   strtcol) and puts in eye; repeats until neither middle line is *
*   zero                                                         *

```

```
*****/
```

```
good_im(datlin,datcol,eye,strtcol,strtrow,data,buf1)
```

```
N16 datlin,datcol,strtcol,strtrow,data[];
```

```
N32 eye,buf1 ;
```

```
{
```

```
    N16 acquired,base,max16,min16,nelets;
```

```
    int i,j,k,tries;
```

```
    N32 nwords,dtype;
```

```
    dtype=0;
```

```
    nelets=datlin*datcol;
```

```
    nwords=0;
```

```
    (N16)nwords=nelets;
```

```
    acquired=0;
```

```
    tries=0;
```

```
    while (acquired !=1)
```

```
    {
```

```
        getim(datlin,datcol,eye,strtcol,strtrow);
```

```
        tries++;
```

```
        zget(data,&eye,&nwords,&dtype);
```

```
        printf("try # %x :",tries);
```

```
        cmax16(nelets,eye,buf1,&max16,&min16);
```

```
        acquired=1;
```

```
        for (i=0;i<2;i++)
```

```
        {
```

```
            k=0;
```

```

        base=datlin*datcol/2+datcol*i;

        printf("base=%x\n",base);

        if (data[base]==0)
        {
            for (j=0;j<datcol;j++)

                if (data[base+j]==data[base]) k=k+1;

            printf("zero magnitude points: %x\n",k);

            if (k==datcol) acquired=0;

        }

        if (tries>10)

            if (y_or_n("DIFFICULTY ACQUIRING IMAGE, TYPE Y WHEN READY"));

    }

    }

    printf("image aquired after %x tries\n",tries);

    return(tries);

}

```

/*****

*		*
*	Name : getim	*
*	Args : N16 rows,columns; N32 image0; N16 strtcol,strtrow	*
*	Description : acquires a rows*columns image into image0. The	*
*	image starts (upper left hand corner) at (strtcol,strtrow)	*
*		*

*****/

```
getim(rows,columns,image0,strtcol,strtrow)
N16 rows,columns,strtcol,strtrow;
N32 image0;
{
    N16 arg[10];
    N32 argcount;
    char c,d,str[80];

    printf("acquire image ?\n");
    scanf("%s",str);
    while (str[0] !='y')
        {printf("type y when ready\n");scanf("%s",str);}

    arg[0]=columns;
    arg[1]=rows;
    arg[2]=(N16)image0;
    arg[3]=strtcol;
    arg[4]=strtrow;
    argcount=5;
    if ZFAILURE(ztcall("ACQUIRE",&argcount,arg))
    {
        printf("cannot translate acquire\n");
        exit(1);
    }
}
```

```

    }

    else printf("acquire downloaded\n");

}

/*****
*   Name : re_grab(datlin,datcol,eye,strtcol,strtrow,data,buf1,      *
*   acqovly,acqentpt)
*   Args : N16 datlin,datcol,strtcol,strtrow,data[]; N32 eye,buf1      *
*           N32 acqovly,acqentpt;
*   Description : grabs dalin* datcol image starting at (strtrow,      *
*   strtcol) and puts in eye; repeats until neither middle line is      *
*   zero; there is no user interface, and zcall not ztcall is used      *
*****/
re_grab(datlin,datcol,eye,strtcol,strtrow,data,buf1,acqovly,acqentpt)
N16 datlin,datcol,strtcol,strtrow,data[];
N32 eye,buf1,acqovly,acqentpt ;
{
    N16 intval,acquired,base,maxl6,minl6,nelets;

    int i,j,k,tries;

    N32 nwords,dtype;

    dtype=0;

    nelets=datlin*datcol;

    nwords=0;

    (N16)nwords=nelets;

    acquired=0;

```

```

    tries=0;
while (acquired !=1)
{
    intval=1;

    interr(intval);

    grab(datlin,datcol,eye,strtcol,strtrow,acqovly,acqentpt);

    tries++;

    zget(data,&eye,&nwords,&dtype);

    printf("try # %x :",tries);

    cmaxl6(nelets,eye,buf1,&maxl6,&minl6);

    acquired=1;

    for (i=0;i<2;i++)
    {
        k=0;

        base=datlin*datcol/2+datcol*i;

        if (data[base]==0)
        {
            for (j=0;j<datcol;j++)

                if (data[base+j]==data[base]) k=k+1;

            printf("zero magnitude points: %x\n",k);

            if (k==datcol) acquired=0;

        }

        if (tries>10)

            if (y_or_n("DIFFICULTY ACQUIRING IMAGE, TYPE Y WHEN READY"));
    }
}

```

```

    printf("image aquired after %x tries\n",tries);

    return(tries);

}

/*****
*   Name : n_grab(nb,datlin,datcol,eye,strtcol,strtrow,data,bufl,      *
*   acqovly,acqentpt)                                                *
*   Args : N16 nb,datlin,datcol,strtcol[],strtrow[],data[];         *
*           N32 eye[],bufl,acqovly,acqentpt;                         *
*   Description : based on re_grab, grabs nb datlin* datcol images   *
*   starting at (strtrow[i],strtcol[i]) and puts in eye[i]          *
*   re_grabs all images until neither middle line is zero on any    *
*   image; there is no user interface, and zcall not ztcall is used *
*****/
n_grab(nb,datlin,datcol,eye,strtcol,strtrow,data,bufl,acqovly,acqentpt)
N16 nb,datlin,datcol,strtcol[],strtrow[],data[];
N32 eye[],bufl,acqovly,acqentpt;
{
    N16 intval,acquired,base,maxl6,minl6,nelets;

    int i,j,k,tries,im;

    N32 nwords,dtype;

    intval=1;

    q_interr(intval);

    dtype=0;

    nelets=datlin*datcol;

```

```

nwords=0;

(N16)nwords=nelets;

acquired=0;

tries=0;

while (acquired !=1)
{
    acquired=1;

    tries++;

    printf("try # %x :",tries);

    q_interr(intval);

    for (im=0;im<nb;im++)

grab(datlin,datcol,eye[im],strtcol[im],strtrow[im],acqovly,acqentpt);

    printf("images acquired, will be tested\n");

    for (im=0;im<nb;im++)
    {

        zget(data,&eye[im],&nwords,&dtype);

        cmax16(nelets,eye[im],buf1,&max16,&min16);

        for (i=0;i<2;i++)
        {

            k=0;

            base=datlin*datcol/2+datcol*i;

            if (data[base]==0)
            {

                for (j=0;j<datcol;j++)

                    if (data[base+j]==data[base])

```

```

        k=k+1;

        printf("zero magnitude points: %x\n",k);

        if (k==datcol) acquired=0;

    }

}

if (tries>10)

    if (y_or_n("DIFFICULTY ACQUIRING IMAGE, TYPE Y WHEN
READY"));

    printf("image aquired after %x tries\n",tries);

    return(tries);

}

q_n_grab(nb,datlin,datcol,eye,strtcol,strtrow,acqovly,acqentpt)
N16 nb,datlin,datcol,strtcol[],strtrow[];
N32 eye[],acqovly,acqentpt ;
{

    N16 intval;

    int im;

    intval=1;

    q_interr(intval);

    for (im=0;im<nb;im++)

        grab(datlin,datcol,eye[im],strtcol[im],strtrow[im],acqovly,acqentpt); }

/*****
*
*
*****/

```

```

*      Name : grab(rows,columns,image0,strtcol,strtrow,acqovly,acqentpt      *
*      Args : N16 rows,columns; N32 image0; N16 strtcol,strtrow ;          *
*      N32 acqentpt,acqovly                                                *
*      Description : acquires a rows*columns image into image0. The        *
*      image starts (upper left hand corner) at (strtcol,strtrow).this     *
*      program uses zcall instead of ztcall, and has no user interface     *
*                                                                           *
*****/
grab(rows,columns,image0,strtcol,strtrow,acqovly,acqentpt)
N16 rows,columns,strtcol,strtrow;
N32 image0,acqovly,acqentpt;
{
    N16 arg[10],intval;
    N32 argcount;
    char c,d,str[80];

    arg[0]=columns;
    arg[1]=rows;
    arg[2]=(N16)image0;
    arg[3]=strtcol;
    arg[4]=strtrow;
    argcount=5;
    if ZSFAILURE(zcall(&acqovly,&acqentpt,&argcount,arg))
    {
        printf("cannot translate acquire\n");
        zclose("0");
    }
}

```

```

        exit(1);

    }

    else printf("acquire downloaded\n");

    printf("\n");

/*

    printf("will call interr\n");

*/

/*

    intval = 1;

    q_interr(intval);

*/

}

/*****

*
*
*   Name: display
*
*   Args: N16 : rows,columns, N32 bufnum,bufadr N16 :data[];
*
*
*   Description : displays the image stored in the "data" array
*
*   but first pads with zeros the first roffset lines and coffset
*
*   columns so that all the elements in data actually appear on the
*
*   screen
*
*****/

display(rows,columns,bufnum,bufadr,data)

```

```

N32 bufadr,bufnum;
N16 rows,columns,data[];
{
    N32 bufsiz;
    N32 nwords;
    N32 dtype;
    N32 try;
    N16 tab[1024];
    N16 ncolumns,nrows,coffset,roffset;
    int i,j;

    roffset=16;
    coffset=32;
    ncolumns=columns+coffset;
    nrows=rows+roffset;
    bufsiz=ncolumns*nrows;
    printf("image size with added borders: %lx\n",bufsiz);
    for (i=0; i<1024 ; i++) tab[i]=0;

    try=bufadr;
    for (i=0; i<roffset ; i++)
    {
        dtype=0;
        nwords=ncolumns;
        if ZSFAILURE(zput(tab,&bufnum,&nwords,&dtype))

```

```

    {
        printf("cannot load image0\n");
        zclose("0");
        exit(1);
    }
    try=try+ncolumns;
    if ZSFAILURE(zmodbf(&bufnum,&try))
    {
        printf("cannot change buffer address\n");
        zclose("0");
        exit(1);
    }
}
for (i=0; i<rows ; i++)
{
    for (j=0;j<columns;j++) tab[coffset+j]=data[i*columns+j];
    dtype=0;
    nwords=ncolumns;
    if ZSFAILURE(zput(tab,&bufnum,&nwords,&dtype))
    {
        printf("cannot load image0\n");
        zclose("0");
        exit(1);
    }
    try=try+ncolumns;
    if ZSFAILURE(zmodbf(&bufnum,&try))

```

```

        {
            printf("cannot change buffer address\n");
            zclose("0");
            exit(1);
        }
    }

    if ZSFAILURE(zmodbf(&bufnum,&bufadr))
    {
        printf("cannot change buffer address\n");
        zclose("0");
        exit(1);
    }

    else printf("buffer address changed\n");

    show(nrows,ncolumns,bufnum);
    printf("image displayed\n");
}

show(rows,columns,image0)

N16 rows,columns;
N32 image0;
{
    N16 arg[10];
    N16 val,csr,rc,cr;
    N32 argcount;
    int input;
    char c,d,str[10];

```

```

input=0;
while (input!=13)
{
    printf("press <cr> to display image\n");
    input=getch();
}
arg[0]=columns;
arg[1]=rows;
arg[2]=(N16)image0;
argcount=3;
if ZSFAILURE(ztcall("GIVEVD",&argcount,arg))
{
    printf("cannot translate givevd\n");
    exit(1);
}

input=0;
while (input!=13)
{
    printf("press <cr> to stop display\n");
    input=getch();
}
val=0x1;
csr=0x6;
rc=zwrctl(&csr,&val);

```

```

    if (rc !=ZS_SUCC)
    {
        printf("still display\n");
        cr=zwrcctl(&csr,&val);printf("cr%d,val:%u",cr,val);
        if ZSFAILURE(zclose("0")) printf("cannot close zip\n");
        else printf("zip closed\n");
        exit(1);
    }
    else printf ("display stopped\n");
}

/*****

mon(rows,columns,image0)
N16 rows,columns;
N32 image0;
{
    N16 arg[10];
    N32 argcount;

    arg[0]=columns;
    arg[1]=rows;
    arg[2]=(N16)image0;
    argcount=3;
    if ZSFAILURE(ztcall("GIVEVD",&argcount,arg))
    {

```

```

        printf("cannot translate givevd\n");
        exit(1);
    }
}

cl_mon()
{
    N16 val,csr,rc,cr;

    csr=0x6;

    val=0x1;
    rc=zrdctl(&csr,&val);
    while (val !=0)
    {
        rc=zrdctl(&csr,&val);
        printf("too fast\n");
        if (rc !=ZS_SUCC)
        {
            printf("still display\n");
            printf("cannot read register %x return code : %d",csr,rc);
            if ZSFAILURE(zclose("0")) printf("cannot close zip\n");
            else printf("zip closed\n");
            exit(1);
        }
    }
}

```

```

val=0x1;
rc=zwrc1(&csr,&val);
if (rc !=ZS_SUCC)
{
    printf("still display\n");
    printf("cannot poke %x : return code : %d",csr,rc);
    if ZSFAILURE(zclose("0")) printf("cannot close zip\n");
    else printf("zip closed\n");
    exit(1);
}
}

```

ZIPCALLS.C

```

#include "c:\c\stdio.h"

#include "c:\driver\mcshdr.h"

/*

#include "c:\driver\zip.h"

*/

#include "c:\driver\import.h"

/*****

*   Name : cmul1c                                     *
*   Args : im0,im1,im2,rows,columns,target,bufl,type   *
*   Descr: im2=im0*im1; rows, columns in image; if type=0 then *
*   columns=columns/2 (accounts for packaging real    fft's. Product *
*   is shifted to target                                *
*****/

/*****

*   Name : q_cmul1c                                     *
*   Args : im0,im1,im2,rows,columns,target,bufl,type   *
*   Descr: same as above but quicker                    *
*****/

/*****

*   Name : recip                                         *
*   Args : N16 nelets, N32 im0,im1                      *
*   Descr: im1[i]=1/sqrt(im0[i]); saturated value for im0[i]=0 *

```

```

*****/
/*****
*   Name : Vmul32                                     *
*   Args : N16 nelets, N32 : im0,im1,im2             *
*   Descr: Multiplies im0 by im1 and puts in im2 (buffers can be *
*   identical.                                         *
*****/

```

```

/*****
*   Name : cmul1c                                     *
*   Args : im0,im1,im2,rows,columns,target,buf1,type *
*   Descr: im2=im0*im1; rows, columns in image; if type=0 then *
*   columns=columns/2 (accounts for packaging real    fft's. Product *
*   is shifted to target                               *
*****/

```

```

cmul1c(im0,im1,im2,rows,columns,target,buf1,type)

```

```

N32 im0,im1,im2,target,buf1;

```

```

N16 rows,columns,type;

```

```

{
    N32 max32,min32,argcount;
    N16 arg[10],nelets;
    argcount=5;
    arg[0]=(N16)im0;
    arg[1]=(N16)im1;

```

```

    arg[2]=(N16)im2;

    arg[3]=rows;

    if (type==0) arg[4]=columns/2;
    else arg[4]=columns;

    ztcall("cmulhc",&argcount,arg);

    printf("after multiplication\n");

    nelets=rows*columns;

    cmulhc(nelets,im2,buf1,&max32,&min32);

    auto_shift(im2,buf1,rows,columns,target);

}

q_cmulhc(im0,im1,im2,rows,columns,target,buf1,type)
N32 im0,im1,im2,target,buf1;
N16 rows,columns,type;
{
    N32 max32,min32,argcount;

    N16 arg[10],nelets;

    argcount=5;

    arg[0]=(N16)im0;

    arg[1]=(N16)im1;

    arg[2]=(N16)im2;

    arg[3]=rows;

    if (type==0) arg[4]=columns/2;
    else arg[4]=columns;

    ztcall("cmulhc",&argcount,arg);

/*

```

```

    printf("after multiplication\n");

    nelets=rows*columns;

    cmax32(nelets,im2,buf1,&max32,&min32);

*/

    q_auto_shift(im2,buf1,rows,columns,target);

}

/*****
*   Name : recip                                     *
*   Args : N16 nelets, N32 im0,im1                 *
*   Descr: im1[i]=1/sqrt(im0[i]); saturated value for im0[i]=0 *
*****/
recip(nelets,im0,im1)

N16 nelets;

N32 im0,im1;

{

    N32 argcount;

    N16 arg[10];

    argcount=3;

    arg[0]=nelets;

    arg[1]=im0;

    arg[2]=im1;

    ztcall("rosr32b",&argcount,arg);

}

```

```

/*****
*   Name : Vmul32                                           *
*   Args : N16 nelets, N32 : im0,im1,im2                   *
*   Descr: Multiplies im0 by im1 and puts in im2 (buffers can be *
*   identical.                                             *
*****/
vmul32(nelets,im0,im1,im2)
N16 nelets;
N32 im0,im1,im2;
{
    N32 argcount;
    N16 arg[10];
    int rc;

    argcount=4;
    arg[0]=nelets;
    arg[1]=im0;
    arg[2]=im1;
    arg[3]=im2;
    rc=ztcall("vmul32a",&argcount,arg);
    if (rc!=0)
    {
        printf("cannot call vmul32a rc=%d \n",rc);
        zclose("0");
        exit(1);
    }
}

```

```
else printf("vmul32a\n");  
}
```

R2DFFT.C

```

/*****
Name : lfft2d(im0,im1,rows,columns,control,buf1,target,nelets,
*      wkbuf,lcoef)
*
*      Args : N32 im0,im1,buf1,target,wkbuf,lcoef;
*
*              N16 rows,columns,control,nelets;
*
*      description : does a real forward or backward fft then shifts
*
*              image to target
*
*              control=1 : unpack im0->im1 and forward fft
*
*              control=3 : inverse fft
*****/

/*****
*      Name : q_lfft2d(im0,im1,rows,columns,control,buf1,target,nelets,
*
*      wkbuf,lcoef,fftofly,fftentpt)
*
*      Args : N32 im0,im1,buf1,target,wkbuf,lcoef,fftofly,fftentpt;
*
*              N16 rows,columns,control,nelets;
*
*      description : same as above but quieter and quicker and uses
*
*              zcall instead of ztcall
*****/

/*****
*      Name fft2d(image32,imlin,imcol,control)
*
*      Args N32 image32 ; N16 imlin,imcol,control
*
*      Descr : does a 2d real fft of image 32, puts in image 32
*
*      control=1 :direct fft; control=3 inverse fft
*****/

/*****
*      Name q_fft2d(image32,imlin,imcol,control,fftofly,fftentpt)

```

```

*   Args N32 image32,fftofly,fftentpt; N16 imlin,imcol,control   *
*   Descr : same as above but uses zcall instead of ztcall      *
*****/
/*****
*   Name : unpack(image16,image32,imlin,imcol)                   *
*   Args : N32 image32,image16; N16 imlin,imcol                 *
*   Description : unpacks image16 to image 32                    *
*                                                                *
*****/
/*****
*   Name : pack(image32,image16,imlin,imcol)                     *
*   Args : N32 image32,image16; N16 imlin,imcol                 *
*   Description : packs image 32 to image16 with negative clipping *
*                                                                *
*****/
/*****
*   Name : npack(image32,image16,imlin,imcol)                   *
*   Args : N32 image32,image16; N16 imlin,imcol                 *
*   Description : packs image 32 to image16 with no clipping    *
*                                                                *
*****/
/*****
*   Name : set_shift(image,imlin,imcol)                          *
*   Args : N32 image; N16 imlin,imcol : # of lines ,columns in input *
*   Description : prompts for control,upl and shifts image      *
*   (control=0 :up, 1:dn) by upl bits                           *

```

```

*****/
/*****
*   Name : shfl                                           *
*   Args : N32 image; N16 imlin,imcol : # of lines ,columns in input *
*           N16 :control,upl how much to shift;           *
*   Description : shifts image (control=0 :up, 1:dn) by upl bits *
*****/

```

```

#include "c:\c\stdio.h"
#include "c:\driver\mcshdr.h"
#include "c:\driver\zip.h"
#include "c:\driver\import.h"

```

```

/*****
*   Name : lfft2d(im0,iml,rows,columns,control,bufl,target,nelets, *
*           wkbuf,lcoef)                                           *
*   Args : N32 im0,iml,bufl,target,wkbuf,lcoef;                 *
*           N16 rows,columns,control,nelets;                     *
*   description : does a real forward or backward fft then shifts *
*           image to target                                         *
*           control=1 : unpack im0->iml and forward fft          *
*           control=3 : inverse fft                                *
*****/
lfft2d(im0,iml,rows,columns,control,bufl,target,nelets,wkbuf,lcoef)

```

```

N32 im0,im1,buf1,target,wkbuf,lcoef;
N16 rows,columns,control,nelets;
{
    N32 max32,min32;
    N32 argcount;
    N16 arg[10];
    int rc,intval;
    char s[80];

    if (control==1) unpack(im0,im1,rows,columns);

    argcount=7;          /***** fft *****/
    arg[0]=(N16)im1;
    arg[1]=(N16)im1;
    arg[2]=rows;
    arg[3]=columns;
    arg[4]=control;
    arg[5]=(N16)lcoef;
    arg[6]=(N16)wkbuf;
    rc=ztcall("ltdfft",&argcount,arg);
    if (rc!=0)
    {
        printf("rc= %x; cannot call ltdfft\n",rc);
        exit(1);
    }
}

```

```

    printf("ltdfft\n");
/*
    printf("after fft :");
    cmax32(nelets,iml,bufl,&max32,&min32);
*/
    q_auto_shift(iml,bufl,rows,columns,target);
    intval=1;
    q_interr(intval);
}

/*****
*   Name : q_lfft2d(im0,iml,rows,columns,control,bufl,target,nelets,      *
*   wkbuf,lcoef,fftovly,fftentpt)                                         *
*   Args : N32 im0,iml,bufl,target,wkbuf,lcoef,fftovly,fftentpt;        *
*           N16 rows,columns,control,nelets;                             *
*   description : same as above but quieter and quicker and uses         *
*           zcall instead of ztcall                                       *
*****/
q_lfft2d(im0,iml,rows,columns,control,bufl,target,nelets,wkbuf,lcoef,fftov-
ly,fftentpt) N32 im0,iml,bufl,target,wkbuf,lcoef,fftovly,fftentpt;
N16 rows,columns,control,nelets;
{
    N32 max32,min32;
    N32 argcount;
    N16 arg[10];
    int rc,intval;

```

```

char s[80];

if (control==1) unpack(im0,iml,rows,columns);

argcount=7;          /***** fft *****/
arg[0]=(N16)iml;
arg[1]=(N16)iml;
arg[2]=rows;
arg[3]=columns;
arg[4]=control;
arg[5]=(N16)lcoef;
arg[6]=(N16)wkbuf;
rc=zcall(&ffftovly,&ffftentpt,&argcount,arg);
if (rc!=0)
{
    printf("rc= %x; cannot call ltdfft\n",rc);
    exit(1);
}
printf("ltdfft\n");
/*
printf("after fft :");
cmax32(nelets,iml,buf1,&max32,&min32);
*/
q_auto_shift(iml,buf1,rows,columns,target);
intval=1;

```

```

    q_interr(intval);
}

/*****
*   Name fft2d(image32,imlin,imcol,control)   *
*   Args N32 image32 ; N16 imlin,imcol,control   *
*   Descr : does a 2d real fft of image 32, puts in image 32   *
*   control=1 :direct fft; control=3 inverse fft   *
*****/
fft2d(image32,imlin,imcol,control)
N16 imlin,imcol,control;
N32 image32;
{
    N32 lcoef,wkbuf,argcount;
    N16 arg[10];
    char s[80];

    sprintf(s,"wkbuf");
    wkbuf=ctrans(s);
    sprintf(s,"lcoef");
    lcoef=ctrans(s);

    argcount=7;          /***** fft *****/
    arg[0]=(N16)image32;
    arg[1]=(N16)image32;

```

```

    arg[2]=imlin;
    arg[3]=imcol;
    arg[4]=control;
    arg[5]=(N16)lcoef;
    arg[6]=(N16)wkbuf;
    if ZSFAILURE(ztcall("ltdfft",&argcount,arg))
    {
        printf("cannot call ltdfft\n");
        zclose("0");
        exit(1);
    }
    printf("ltdfft\n");
}

/*****
*   Name q_fft2d(image32,imlin,imcol,control,fftovly,fftentpt)      *
*   Args N32 image32,fftovly,fftentpt; N16 imlin,imcol,control      *
*   Descr : same as above but uses zcall instead of ztcall          *
*****/
q_fft2d(image32,imlin,imcol,control,fftovly,fftentpt)
N16 imlin,imcol,control;
N32 image32,fftovly,fftentpt;
{
    N32 lcoef,wkbuf,argcount;
    N16 arg[10];
    char s[80];

```

```

    sprintf(s,"wkbuf");
    wkbuf=ctrans(s);
    sprintf(s,"lcoef");
    lcoef=ctrans(s);

    argcount=7;          /***** fft *****/
    arg[0]=(N16)image32;
    arg[1]=(N16)image32;
    arg[2]=imlin;
    arg[3]=imcol;
    arg[4]=control;
    arg[5]=(N16)lcoef;
    arg[6]=(N16)wkbuf;
    if ZSFAILURE(zcall(&fftovly,&fftentpt,&argcount,arg))
    {
        printf("cannot call ltdfft\n");
        zclose("0");
        exit(1);
    }
    printf("ltdfft\n");
}

/*****
*   Name : unpack(image16,image32,imlin,imcol)   *
*****/

```



```

*   Args : N32 image32,image16; N16 imlin,imcol   *
*   Description : packs image 32 to image16 with no clipping   *
*   *   *
*****/
npack(image32,image16,imlin,imcol)
N16 imlin,imcol;
N32 image32,image16;
{
    N32 argcount;
    N16 arg[10];
    argcount=4;
    arg[0]=(N16)image32;
    arg[1]=(N16)image16;
    arg[2]=imlin;
    arg[3]=imcol;
    if ZSFAILURE(ztcall("p3216",&argcount,arg))
    {
        printf("cannot call p3216\n");
        exit(1);
    }
}

/*****
*   Name : pack(image32,image16,imlin,imcol)   *
*   Args : N32 image32,image16; N16 imlin,imcol   *

```

```
*      Description : packs image 32 to image16 with negative clipping      *
```

```
*                                                                                      *
```

```
*****/
```

```
pack(image32,image16,imlin,imcol)
```

```
N16 imlin,imcol;
```

```
N32 image32,image16;
```

```
{
```

```
    N32 argcount;
```

```
    N16 arg[10];
```

```
        argcount=4;
```

```
        arg[0]=(N16)image32;
```

```
        arg[1]=(N16)image16;
```

```
        arg[2]=imlin;
```

```
        arg[3]=imcol;
```

```
        if ZSFAILURE(ztcall("p3216b",&argcount,arg))
```

```
        {
```

```
            printf("cannot call p3216b\n");
```

```
            exit(1);
```

```
        }
```

```
}
```

```
*****/
```

```
*      Name : set_shift(image,imlin,imcol)      *
```

```
*      Args : N32 image; N16 imlin,imcol : # of lines ,columns in input      *
```

```

*      Description : prompts for control,upl and shifts image      *
*      (control=0 :up, 1:dn) by upl bits                          *

```

```

*****/

```

```

set_shift(image,imlin,imcol)

```

```

N32 image;

```

```

N16 imlin,imcol;

```

```

{

```

```

    N16 control,upl;

```

```

    get16("up(0) or down (1)",&control);

```

```

    get16("by",&upl);

```

```

    shfl(image,imlin,imcol,control,upl);

```

```

}

```

```

/*****/

```

```

*      Name : shfl                                              *

```

```

*      Args : N32 image; N16 imlin,imcol : # of lines ,columns in input *

```

```

*      N16 :control,upl how much to shift;                      *

```

```

*      Description : shifts image (control=0 :up, 1:dn) by upl bits *

```

```

*****/

```

```

shfl(image,imlin,imcol,control,upl)

```

```

N32 image;

```

```

N16 imlin,imcol,control,upl;

```

```

{

```

```

    N16 arg[10];

```

```

    N32 argcount;

```

```

if (upl != 0)
{
    argcount=5;
    arg[0]=(N16)image;
    arg[1]=(N16)image;
    arg[2]=imlin;
    arg[3]=imcol;
    arg[4]=upl;
    if (control==0)
    {
        if ZSFAILURE(ztcall("upshfl",&argcount,arg))
        {
            printf("cannot call upshfl\n");
            exit(1);
        }
    }
    if (control==1)
    {
        if ZSFAILURE(ztcall("dnshfl",&argcount,arg))
        {
            printf("cannot call dnshfl\n");
            exit(1);
        }
    }
}

```

}

U_MEX.C

```

#include "c:\c\stdio.h"

#include "c:\c\math.h"

#include "c:\driver\mcshdr.h"

#include "c:\driver\import.h"

/*****

*   Name : mex(data,size)                                     *
*   Args : N16 data[], size                                   *
*   descr: fills array data with mexian hat filter; if size=0 *
*           64*64 filter, if size=1 : 128*128                 *
*****/

/*****

*
*   Name : mexfilt(land,tland,tmh,wkbuf,lcoef,cmultgt,im_top,bufl,
*           rows,columns,nelets,fftovly,fftentpt,data,nwords,dtype)
*   Args      N32 : land : 16bit elements input buffer
*           N32 : tland : same as land but unpacked to 32bit
*           N32 : tmh : mexican hat filter
*           N32 : wkbuf,lcoef : buffers for lfft2d
*           N32 : cmultgt : target for scaling before complex multiply
*           N32 : target for scaling for maximum range
*           N32 : bufl : buffer for cmax
*           N16 : rows, columns : rows,columns in image
*           N16 : nelets : elements in image
*           N16 : data[] array;
*           N32.nwords, dtype : # words, type for zip transfers
*
*****/

```

```

*          N32 ffttentpt,fftovly : entry point, overlay of lfft2d          *
*
*
*   Desc : mexican hat filters land, puts result back in land             *
*****/
/*****
*
*
*   Name : dat_shft(datlin,datcol,intab,outtab)                             *
*   Args : N16 intab[],outtab[],datlin,datcol                             *
*   Desc : shifts image by (datlin/2,datcol/2)                             *
*****/
mex(data,size)
N16 data[],size;
{

    double current,max,mult,r,s,t1,t2;
    N16 i,j,im,jm,init,dim;
    long il,jl;
    s=sqrt(2.0)/2.0;
    printf("s= %lf\n",s);
    max=0.0;
    for (i=0;i<0x4000;i++) data[i]=0;
    if (size == 0)
    {
        init=0x618;
        dim=0x40;
    }
}

```

```

else
{
    init=0x1c38;
    dim=0x80;
}
for (i=0;i<0x10;i++)
    for (j=0;j<0x10;j++)
    {
        i1=i;
        t1=i1;
        j1=j;
        t2=j1;
        r=pow(t1-8.0,2.0)+pow(t2-8.0,2.0);
/*
        current=1.0/(s*s*sqrt(6.18)) * (1.0-r/s) * exp(-r/(2*s));*/
        current=1.0/(s*s*s*s*sqrt(6.18)) * (2.0-r/(s*s)) * exp(-
r/(2*s)); /*
        current=1.0/(s*s*sqrt(6.18)) * exp(-r/(2*s));*/
        if (current>max) {max=current;im=i;jm=j;}
        if (current<-max) {max=-current;im=i;jm=j;}
    }
mult=32767.0/max - 1.0;
printf("maximum = %lf mult= %lf %x %x\n",max,mult,im,jm);
for (i=0;i<0x10;i++)
    for (j=0;j<0x10;j++)
    {
        i1=i;
        t1=i1;

```

```

        j1=j;
        t2=j1;
        r=pow(t1-8.0,2.0)+pow(t2-8.0,2.0);
/*
        current=1.0/(s*s*sqrt(6.18)) * (1.0-r/s) * exp(-r/(2*s));*/
        current=1.0/(s*s*s*s*sqrt(6.18)) * (2.0-r/(s*s)) * exp(-
r/(2*s)); /*
        current=1.0/(s*s*sqrt(6.18)) * exp(-r/(2*s));*/
        current=current*mult;
        i1=current;
        data[init+dim*i+j]=i1;
    }

/*
    if (size==0)
    for (i=0;i<0x40;i++) printf("%d %x\n",i,data[0x800+i]);
    else
    for (i=0;i<0x80;i++) printf("%d %x\n",i,data[0x2000+i]);

*/
}
-

mexfilt(land,tland,tmh,wkbuf,lcoef,cmultgt,im_top,buf1,rows,columns,nelets,-
fftofly,fftentpt,data,nwords,dtype) N32
land,tland,tmh,wkbuf,lcoef,cmultgt,im_top,buf1,nwords,dtype;
N32 fftovly,fftentpt;
N16 nelets,rows,columns,data[];
{
    N32 max32,min32;

```

```

N16 control,type;

        /****fft's****/

control=1;

q_lfft2d(land,tland,rows,columns,control,bufl,cmultgt,nelets,wkbuf,lcoef,ff-
tovly,fftentpt);

        /**** convolution,inversion****/

printf("convolution,inversion\n");
type=0;
q_cmullc(tland,tmh,tland,rows,columns,im_top,bufl,type);
control=3;

q_lfft2d(land,tland,rows,columns,control,bufl,im_top,nelets,wkbuf,lcoef,fft-
ovly,fftentpt);    pack(tland,land,rows,columns); /* pack with negative
clipping ! */

unpack(land,tland,rows,columns);
q_auto_shift(tland,bufl,rows,columns,im_top);
pack(tland,land,rows,columns);

/*  npack(tland,land,rows,columns) should theoretically repalce the
four above lines, but in my data it doesn't work as well : this is
apparently becuase the negative clipping eliminates "false negatives" i.e

```

8000 = 7fff+1 see standard algo manual example of ltdfft for details

(p 2-94) */

```
    zget(data,&land,&nwords,&dtype);
    dat_shft(rows,columns,data,data);
    zput(data,&land,&nwords,&dtype);
    printf("mexican hat filter operation completed\n");
}
```

/*

*/

* Name : dat_shft(datlin,datcol,intab,outtab) *

* Args : N16 intab[],outtab[],datlin,datcol *

* Desc : shifts image by (datlin/2,datcol/2) *

*/

```
dat_shft(datlin,datcol,intab,outtab)
```

```
N16 intab[],outtab[];
```

```
N16 datlin,datcol;
```

```
{
```

```
    N16 low_mid,lft_mid,up_mid,offset,tmp;
```

```
    int i,j;
```

```
    low_mid=datlin/2*datcol+datcol/2;
```

```
    lft_mid=datlin/2*datcol;
```

```
    up_mid=datcol/2;
```

```
    for (i=0;i<datlin/2;i++)
```

```

for(j=0;j<datcol/2;j++)
{
    offset=datcol*i;
    tmp=intab[low_mid+offset+j];
    outtab[low_mid+offset+j]=intab[offset+j];
    outtab[offset+j]=tmp;
    tmp=intab[lft_mid+offset+j];
    outtab[lft_mid+offset+j]=intab[up_mid+offset+j];
    outtab[up_mid+offset+j]=tmp;
}
}

```