THE INVESTIGATION AND DESIGN OF A MACHINE VISION SYSTEM FOR THE DETECTION AND CONTROL OF THE SEPARATION IN A SPIRAL ORE CONCENTRATOR

By David Gold

Submitted to the University of Cape Town in partial fulfillment of the requirements for the degree of Master of Science in Engineering.

Cape Town, July 1991.

SYNOPSIS

The objectives of the project were firstly, to propose and implement the hardware needed for a Machine Vision System that is to operate on the Spiral Ore Concentrator. Secondly, to propose and implement an algorithm that would operate together with the hardware, to find the transition point between the two types of material, also to find the optimum position for the blade separator, while the system operates in a continuous repetitive mode. Finally, to have an on line analysis of the efficiency of the refinement process in the Spiral Ore Concentrator.

The hardware of the Machine Vision System was implemented and tested in the laboratory. The hardware included a video recorder, through which a video signal was obtained from the play-back of video material taken of the ore flow in the Spiral Ore Concentrator. The video signal was used as input to an RGB decoder in order to remove the colour modulation. A black and white frame grabber digitized the video signal which was then analyzed by a computer.

Based on the fundamental theory of edge detection used in computer vision systems, an algorithm was designed and implemented to detect the transition point between the two types of material. This algorithm was used to find the difference of averages of grey levels between two global neighborhoods within a specified area of interest window. The algorithm gave consistent results and was robust towards surface irregularities.

The algorithm, operating in a continuous repetitive mode, gave rapid fluctuations in the determined edge position. Because a motor with a slow response time would be used to control the movement of the blade separator, the determined edge position signal was smoothed by a filter. Based on periodogram analysis of the edge position signal, a smoothing filter was implemented which incorporates a median filter, followed by a fading-memory polynomial filter. These filters gave sufficient smoothing with little lag to step changes in ore concentration.

The efficiency of the ore separation was monitored by the determination of losses. These losses consist of, the percentage of black material loss and percentage of white material contamination. Two types of losses could be identified, they were Spraying losses and Filter losses, these were combined to give Total losses. From the true edge position curve, which was obtained by finding the edge position every second

pixel point, the Nyquist sampling frequency could be determined. Because of the slow sampling rate, an error in the calculation of the losses was determined from the true edge curve, and the sub-sampled edge curve. Based on this error, it was shown that the machine vision system could multiplex multiple camera inputs.

The objectives of the project were attained. The hardware together with the edge detection algorithm, accurately and quickly determined the edge position between the two types of material. This system was also able to provide a control motor with the optimum mean edge position, as well as an on line efficiency of the separation process.

ACKNOWLEDGEMENTS

Firstly I would like to thank my supervisor, Professor Gerhard De Jager for his enthusiasm, support and constructive ideas throughout this project. Also for his assistance and guidance in ensuring a smooth flow to this thesis.

I would like to thank Wayne Borchardt, George Horn and Craig White for their constructive ideas and programing skills which made this project a little easier.

I would like to thank Dr Norman Morrison for his help with smoothing techniques.

I would also like to thank MINTEK for their financial support throughout the duration of this project and to Dr Tony Lange for setting realistic and attainable demands for this project.

Finally I would like to thank Yael Stern for her unfailing support and patience throughout the duration of this project.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

Mining is one of the biggest industries in South Africa if not the world. Because it is a large industry, the mining and refinement of huge amounts of ore will yield a large financial return. Hence there is an ever increasing stride towards more sophisticated and more efficient methods of mining and refinement of mineral ore. By increasing the efficiency of the refinement process, a high grade of concentrated mineral ore can be obtained in a much shorter time, thus increasing the yield. Until recently the operation of the refinement process and its efficiency, have been operator controlled. From a prior knowledge of the process the operator would try to optimize the refinement efficiency. However the operator cannot constantly monitor the process and small changes in the process can go undetected.

Due to technological advances the role of the operator can now be automated by a machine. Recent advances in the manufacture of the Microchip and Very Large Scale Integration (VLSI) have made sensory devices reliable, cheap, sensitive to small input changes, and accurate. Also the Microprocessor and Computer, have become a very powerful and cost effective engineering tool. In striving for greater efficiency, sophisticated sensors are now measuring the same features that the operator would normally look for, and computers, based on a prior knowledge of the process, are making the same decisions that the operator would normally make. This results in the refinement process being continually and accurately monitored with optimization changes being almost instantaneous. Even small changes will not go undetected and so the refinement process will have a greatly improved efficiency.

In mining a common method of refinement is through the separation of wanted mineral ore and unwanted material. An example of this is the Spiral Ore Concentrator which separates two types of material of the same particle size, but of different weights. At the output of the spiral, wanted heavy material which is of dark colour, and unwanted light material which is of light colour, can be clearly seen. This visual information is used by an operator who places a dividing blade at the transition point between the two types of material. The refinement process of the spiral concentrator is a dynamic process which results in the transition point constantly changing. However the operator only positions the blade a few times a day, thus resulting in an inefficient process. In order to increase efficiency the action of the operator can be automated and so the refinement process can be made to be more accurate and faster. The optical information can be obtained from a cheap,

robust Charge Coupled Device (CCD) camera, from which it can be digitized by a Frame Grabber and analyzed by a computer. The computer will then decide on the optimum blade position. This forms the basis for a Machine Vision System.

The main objectives of the project were :

1. To propose and implement the hardware needed for a Machine Vision System that is to operate on the spiral ore concentrator.
2. To propose and implement an algorithm that will operate together with the hardware to find the transition point between the two types of material, and also to be able to determine the optimum position for the blade separator while the system operates in a continuous repetitive mode.
3. To have an on line analysis of the efficiency of the refinement process in the spiral ore concentrator.

In industry many spiral ore concentrators are operated together at any one time. In order for large scale implementation of the Machine Vision System the first stage of the project was to implement hardware that would be cheap, reliable, easily obtainable and easily implemented. This hardware would have to accurately digitize the brightness level of the ore material so that it could be correctly analyzed by the algorithm. The second stage of the project was to implement the optimum algorithm that would find the instantaneous transition point between the dark and light ore. This implementation was done under the guidelines of the fundamental principles involved in edge detection, used in computer vision systems. Next, operating in a continuous repetitive mode, the system must be able to provide a control motor that will position a blade separator, with the optimum transition point between the two types of ore. The final stage of the project was to find the operating efficiency of the refinement process. This was done through the determination of the amount of wanted material that was lost, and the amount of contamination from unwanted material. The errors in these readings were also determined. Also examined in the final stage of the project was the feasibility of the system being extended to operate with multiple camera inputs.

The layout of the thesis is as follows :

- Chapter 2 will give a short description of the spiral ore concentrator and its operation.
- Chapter 3 will give the fundamental theory of edge detection used in computer vision systems.
- Chapter 4 will discuss the hardware needed for an accurate representation of the mineral ore's brightness level.

- Chapter 5 discusses the implementation of the optimum edge detector based on the fundamental principles of chapter 3.
- Chapter 6 deals with the determination of the optimum blade position over time, that will the used by a motor to drive the separating blade.
- Chapter 7 discusses the efficiency of the refinement process operating under the machine vision system, as well as the error of this estimate. It also discuses the feasibility of multiplexing multiple camera inputs to the system.
- Appendix A gives the listing used by MCAD for periodogram analysis of the power spectrum used in chapter 6.
- Appendix B, C and D gives the software listing and the users guide to the main programs used in the project. They are, EDGE9GC.C, AOFIC4C.C and EDGETRUE.C.

In order to implement the machine vision system we need to have some knowledge about the spiral ore concentrator and its operation. This is discused in the next chapter.

2. THE SPIRAL CONCENTRATOR

2.1. INTRODUCTION

The *Spiral Concentrator* was first introduced in the early 1940's with the development of the Humphreys spiral. This spiral is an example of a wet gravity concentrator, which is easy to install and operates with minimum supervision and cost, while providing a high degree of ore separation [11]. Today many types of spiral concentrator designs are available, all of which are similar and based on the original Humphreys spiral concentrator [4]. The spiral ore concentrator is used to separate two types of material of the same particle size. The first type of material is of high density and so its particles will be heavy, while the second type of material is of lower density and so its particles will be lighter. This difference in material weights forms the basis of the separation mechanism. In industry some of the heavy materials that are separated by the concentrator are Titanium, Tin, Tungsten minerals, Coal, Magnetite, Chromite, Gold, and Zircon [11].

This chapter will briefly discuss the operation of the spiral concentrator.

2.2. OVERVIEW OF OPERATION OF THE SPIRAL CONCENTRATOR

The spiral concentrator consists essentially of the following elements. A feed box situated at the top of the spiral through which the ore feed enters the spiral. The ore feed is a mixture of the two types of ore material, one heavy and the other light, and water to form a slurry, thus enabling the mixture to flow in the form of a stream. The ore stream then travels down a helical or spiral chute of 5 to 7 turns, with the chute having a semi-circular cross-section. Situated at the bottom of the spiral are 1 or 2 adjustable metal splitters or blades that are molded into take-off ports which separate the material into various streams. Figure 2.1 shows a spiral concentrator used in a closed circuit test unit with the basic elements as described above. In addition there is a tank at the base of the spiral to collect the material output from the take-off ports. This material is then fed into a variable speed sand pump which pumps the material back up to the feed box situated at the top of the spiral for recirculating.

If 2 blades are used the ore stream will be separated into 3 streams. The material obtained from the inner most stream, closest to inner edge of the spiral, will be concentrated heavy material called *Concentrate*. This concentrate is the material that is most wanted. The material obtained from the middle stream will be a mixture of heavy material and light material called *Middling*. This material can be fed back into the spiral for recirculating [11]. Finally the outermost stream called *Tailing* contains only light material which can be discarded. A splitter box with 2 metal blades showing the concentrate (CONC), middling (MIDS), and tailing (TAILS) output streams is shown in figure 2.2. If only 1 blade is used the material is separated into 2 streams. The innermost stream will be the concentrate and the outer stream will the a combination of the middling and tailing. The outer stream can then be retreated to extract all the heavy material, or be discarded.

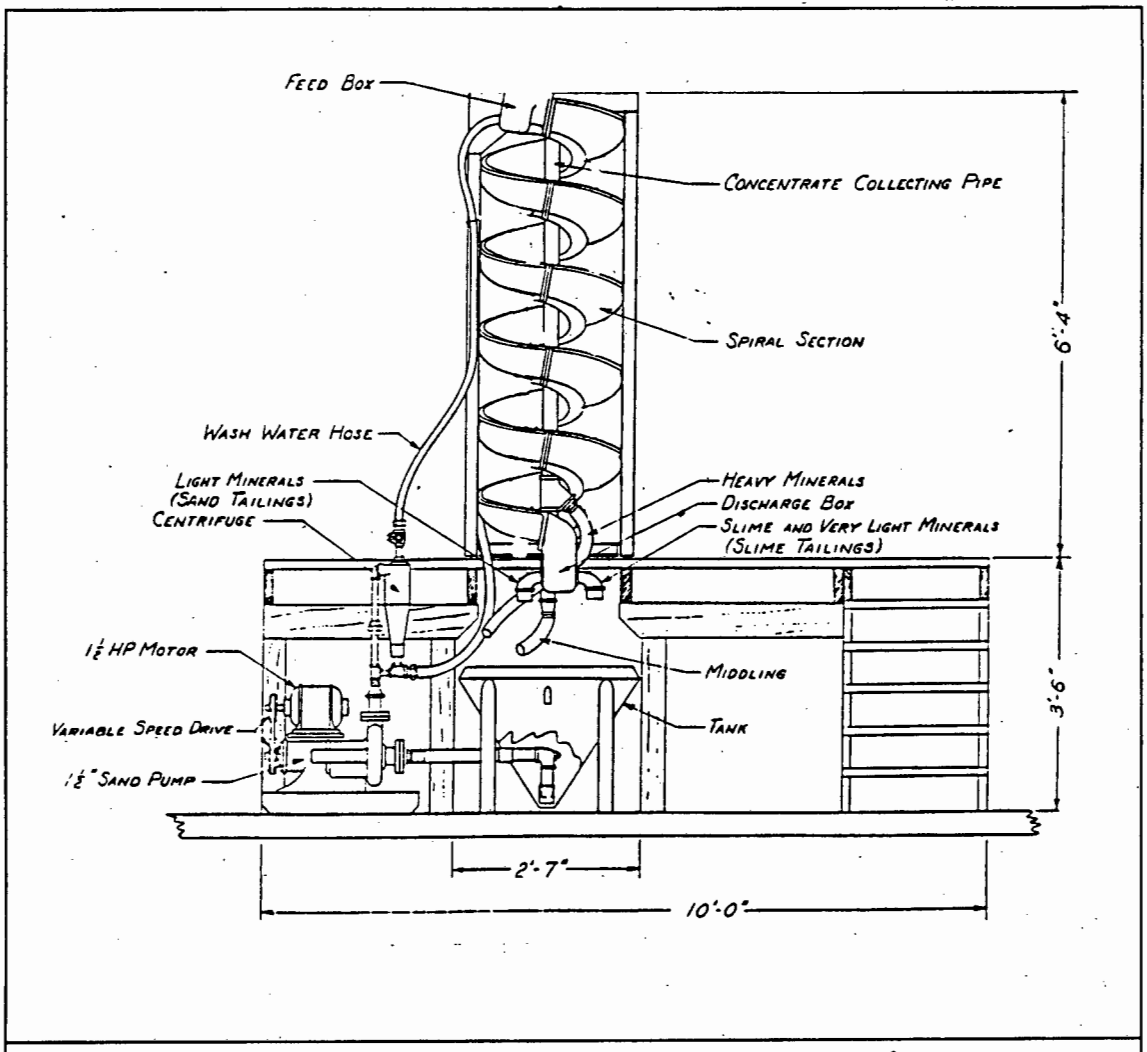


FIGURE 2.1 : A Spiral Concentrator closed circuit test unit.

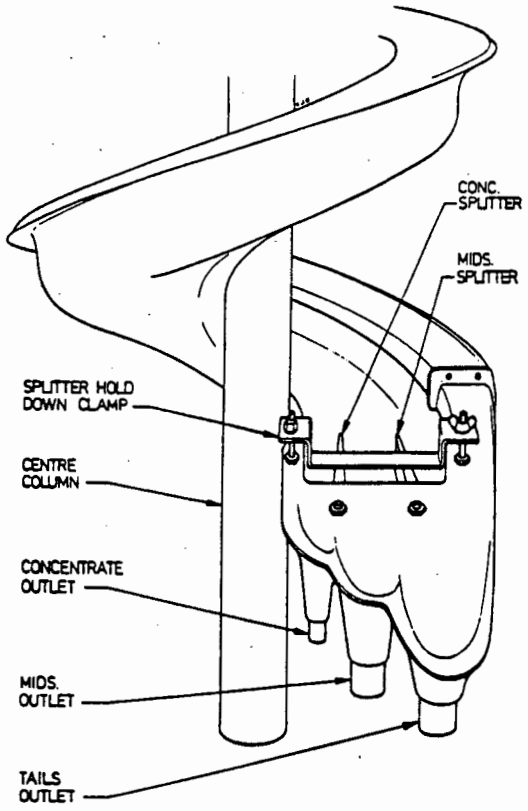


FIGURE 2.2 : A double blade splitter box situated at the base of the spiral chute.

The mechanism of ore separation in the spiral concentrator according to Gleeson [9] is as follows. As the ore stream flows down the spiral chute, each ore particle in the stream will have a centrifugal force acting on it, that is outwardly tangential to the bottom channel of the chute. This force will be directly proportional to the square of the flow velocity of the material and inversely proportional to the radius at which the particle is located. Due to this force, water is piled up on the outer sides of the chute until there is an equilibrium point reached between the gravitational force which pulls the stream downward, and the outward centrifugal force. The flow position of the stream will thus remain constant as long as the spiral is of uniform pitch and radius.

The velocity of the stream will be the greatest just below the surface of the water and will decrease with depth until it is reaches zero at the floor of the channel. The bottom layer of water is retarded by friction and therefore will have much less centrifugal force acting on it. This results in the water flowing sideways along the bottom of the chute, towards the inner edge. The heavy particles situated at the bottom of the chute will thus be carried inward. Because of the inward flow there must also exist an outward flow of water at the surface to replace it. This upper

outward flow will carry the lighter particles situated on the surface, to the outer edge of the spiral chute. The inward and outward flow patterns can be seen in figure 2.3 which shows the spiral flow behavior in a cross-section of the spiral chute.

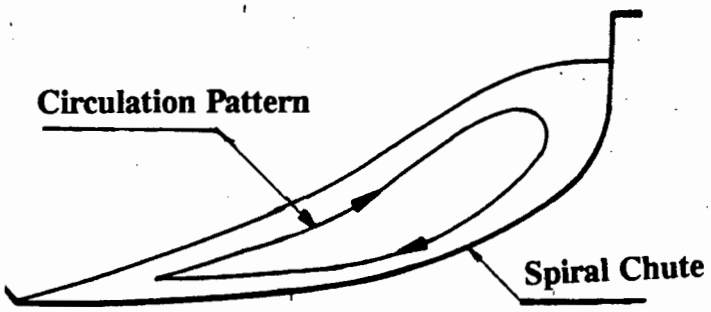


FIGURE 2.3 : Ore flow behavior in a cross-section of the spiral chute.

The inward flow of water will concentrate the heavy material to the inner edge of the chute and the outward flow will concentrate the light material to the outer edge. These can be separated by adjustable metal blades placed at the transition regions from heavy to light, thus splitting the stream into concentrate, middlings and tailings. Figure 2.4 shows a cross-section of the spiral chute with the concentration distribution of the high density heavy particles and the low density light particles. The heavy particles are shown by black dots which are situated predominantly at the inner edge, while that of the light material is represented by white dots and is situated predominantly at the outer edge.

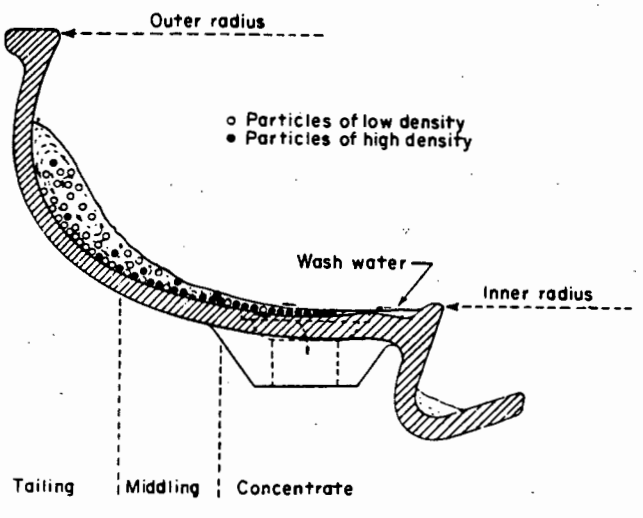


FIGURE 2.4 : Cross-section of a spiral chute showing the distribution of heavy and light material particles.

At the inner edge of the spiral chute there is a groove for wash water. This wash water flows over the surface of the ore and is used to wash off lighter material that is ridding on the heavier material and return it to the outer edge of the chute.

Most types of spirals are based on the Humphreys spiral. The major difference between them is the cross-sectional shape of the spiral chute. For this project a Reichert spiral was used whose cross-sectional shape is seen in figure 2.4. This spiral was set up in a closed circuit test unit as seen in figure 2.1. At the bottom of the spiral only one metal blade was used to separate the ore stream into 2, a concentrate stream, and a middling and tailing stream. The type of ore fed down the spiral was a mixture between titanium and beachsand, titanium being the heavier of the two. The concentrate titanium was almost black while that of beachsand was almost white. The spiral was loaded with various concentrations of ore, starting at 0 and increasing in 0.5 Kg steps to 4 Kg. This was achieved by weighing out 0.5 Kg of pure titanium and adding it to the tank at the base of the spiral. The titanium would then evenly distribute itself throughout the spiral concentrator test unit.

2.3. INDUSTRIAL AUTOMATION FOR THE SPIRAL CONCENTRATOR

Due to different ore concentration loading of the spiral, the outer edge position of the concentrate material will not be fixed but tend to vary over time. Also the instantaneous position will change due to centrifugal force causing the heavy material to be sprayed out in the form of waves. Since the heavy and the lighter materials have different brightness levels, the outer edge position of the concentrate material can be determined by a visual inspection of the ore stream, and the blade placed at the transition point between the two levels. The position of the adjustable metal blade is initially set by the operator and updated a few times a day. Figure 2.5 shown a photograph of the ore mixture of titanium and beachsand in the spiral chute. Titanium is be seen to be the black material and beachsand is the white material. The unevenness of the outer edge of the concentrate material can also be seen.

FIGURE 2.5 : Photograph of an ore mixture of Titanium and Beachsand in the spiral chute.

The inability of the blade to follow the instantaneous and long term changes of the edge, will result in a loss of valuable material, as well as a contamination from the unwanted material. Therefore the instantaneous outer edge of the concentrate needs to be accurately and quickly determined and the blade driven to that position in order to ensure maximum ore recovery with minimal contamination. The heavy and light material is distinguishable through brightness, therefore if an operator can visually make the distinction between them, then a *Machine Vision System* can be implemented that can also make this distinction.

The Machine Vision System will use the brightness information of the ore to determine the correct blade position. The visual brightness information could be obtained through a camera which is positioned at the base of the spiral overlooking the chute. The visual information could then be digitized by a frame grabber and analyzed by a computer. The determined edge position could then be used to control a motor which will control the positioning of the blade. The basic setup of the Machine Vision System applied to the spiral concentrator is seen in figure 2.6.

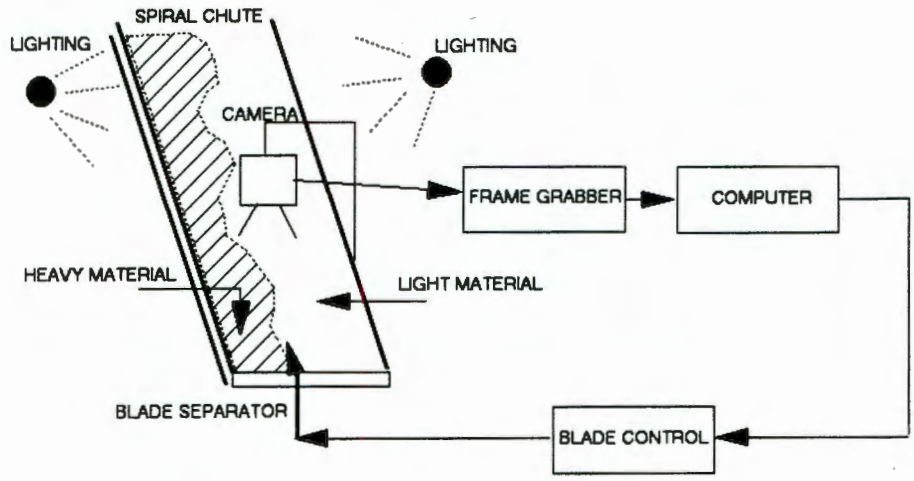


FIGURE 2.6 : A Machine Vision System applied to the spiral concentrator.

The Machine Vision System will be able to determine the blade position accurately and quickly, with the operator only having to initialize the system. This system would thus ensure a maximum ore recovery rate at all times with minimal contamination of unwanted material. The full implementation of the Machine Vision System is discussed in chapter 4.

In order for the Machine Vision System to be able to determine the outer edge position of the concentrate material, fundamental edge detection theory used in computer vision systems, needs to be examined. This is done in the following chapter.

3. EDGE DETECTION THEORY

3.1. INTRODUCTION

The brightness level is observed in order to determine the blade position which will separate the heavy material from that of the light material. The heavy material being dark, has a low brightness or grey level, and the lighter material has a high brightness or grey level. The 2 regions can thus be segmented by examining the grey levels and finding the optimum transition point or *edge* position, from the low grey level to the high grey level. The transition edge position can be found by direct *Thresholding* or the edge can be enhanced by *Edge Enhancement* and *Detection* algorithms.

This chapter will deal with the theory and methods used in edge enhancement and detection.

3.2. EDGE REPRESENTATION

Image segmentation methods can be applied to an image with smooth homogeneous surfaces that correspond to regions of constant grey level, and abrupt grey level changes that correspond to the boundaries between the 2 regions [32]. An image can thus be segmented by edge-based methods that will detect the boundaries or edges which are local features of the image [32, 23]. The definition of an edge according to Rosenfeld and Kak [23] is :

> " An *edge* : the grey level is relatively consistent in each of two adjacent, extensive regions, and changes abruptly as the border between the regions is crossed. An ideal edge has a steplike cross section, while that of a more realistic example will incorporate blur and noise effects. "

Figure 3.1 shows an ideal step edge with a low grey level of 50 and a high grey level of 200. Superimposed on this is the more realistic representation of an edge. Blur effects can be seen by the transition between the 2 grey levels being a ramp, also noise in the form of random variations of the grey level can be seen to affect the whole edge profile.

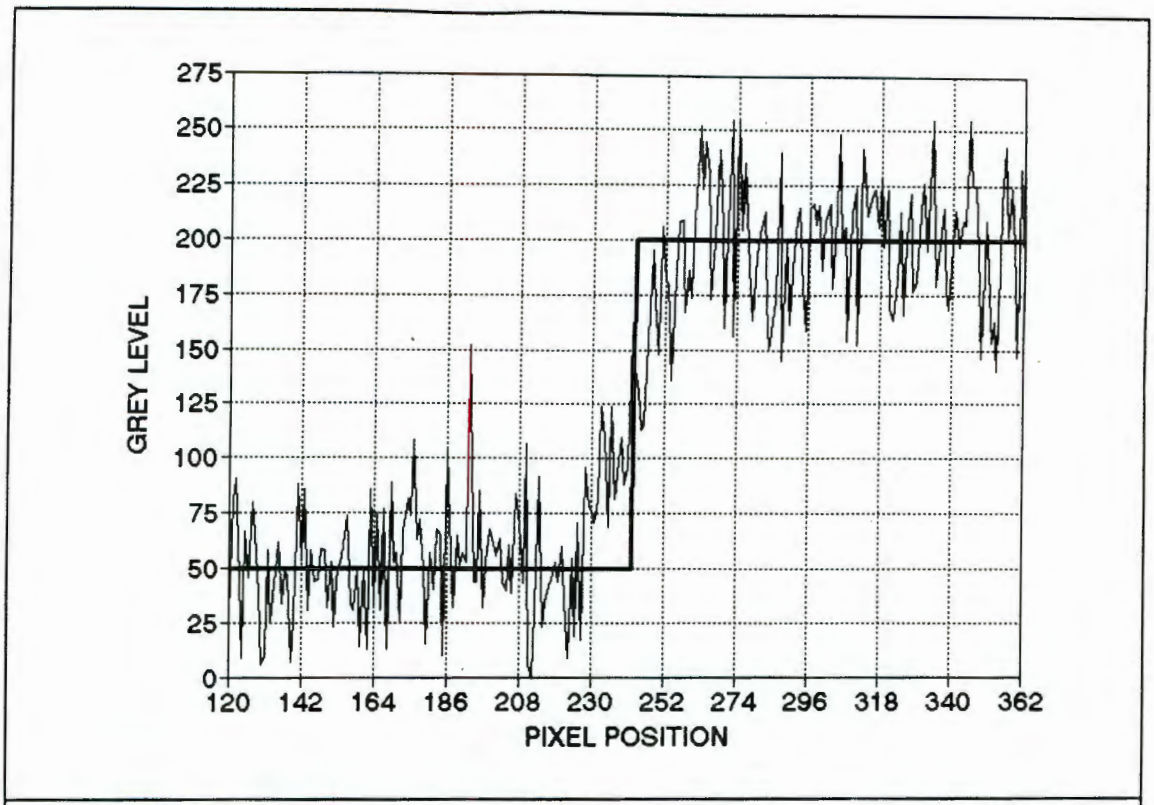


FIGURE 3.1 : An ideal step edge and a more realistic noisy ramped edge.

Edge detection is a large sub-set of image processing. This was evident in a literature search that was done using the *Dialog* database, using the files *Compendex Plus* 1970 - 1990, as well as the *Inspec* 1969 - 1990. The key words were edge detection, or localization, or estimation in pattern recognition, or picture or image processing. The number of articles found was 1987, which was further reduced to 96 by including the key words of review or overview or evaluation, or comparison, or methods or techniques, or assessment or discussion. On conducting a search for edge detection using image or picture processing in the spiral ore concentrator, the number of articles that was found was 0.

From the literature it can be seen that there are many different types of edge detectors. Each edge detector has been designed to cope with a specific application, such as noise suppression or the detection of weak edges in the presence of noise. Because of this the edge detector will work well for its specific application but may not work for another. We thus need to implement an edge detector that is specific to the spiral concentrator. This can be done by examining the fundamental principles of edge detection and then formulating a strategy for the edge detector that is to be implemented.

3.3. EDGE DETECTION

The most common approach to edge detection according to Pratt [21] is shown in figure 3.2. The original image $f(x,y)$ undergoes edge enhancement to form an image field $g(x,y)$ in which the grey level of the edges have been emphasized. The image field is then thresholded to determine the location of the edge and so forms an edge map $e(x,y)$. If the threshold is set too high, edge elements may be excluded, conversely if the threshold is set too low, noise will be falsely detected as edge elements, thus causing multiple crossings of the threshold.

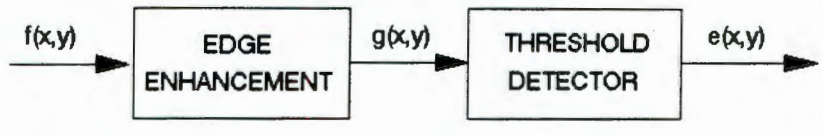


FIGURE 3.2 : A common approach to edge detection.

The optimum threshold value can be obtained by examining the histogram of the edge, the histogram being the number of pixels at a particular grey level [2]. The histogram for the edge shown in figure 3.1, will consist of 2 predominant peaks, centered at 50 and 200. Spreading of these peaks will occur due to the random fluctuation of grey level caused by noise, as well as the blurring of the edge. This type of histogram, because of its 2 peaks, is know as a bimodal histogram, and the threshold can be selected to lie between the 2 peaks. The more the peaks have spread, the more critical the positioning of the threshold level, since it could cause edge elements to be excluded or noise to be included. Many sophisticated methods of thresholding can be adopted and is described in most computer vision literature. In a comparative study of thresholding by Brink [3], the use of moment-preserving thresholding described by Tsai [30], was considered to give the most consistent results.

In order to eliminate the errors that could result from thresholding, the edge is first enhanced to make it distinguishable from its surroundings. The basic principles for the various methods of edge enhancement, are discussed in the following sections.

3.4. EDGE ENHANCEMENT AND DETECTION

The basis of most edge enhancement and detection techniques according to Gonzalez and Wintz [10] is the computation of a local derivative operator. Figure 3.3 from [10] shows a light object placed on a dark background, and a dark object placed on a light background, both having their edges in the vertical direction. It is seen that the profile is ramped due to the blurring of the edge. The first derivative of the edge profile can be seen to have a value of 0 in regions of constant grey level, but has a high value at regions of constant change, such as the ramp of the edge. This value is positive for an edge transition from dark to light and negative for a transition light to dark. For the second derivative of the edge profile, the regions of constant grey level again has a 0 value, but the edge transition or ramp, have positive and negative spikes at the beginning and end of the ramp.

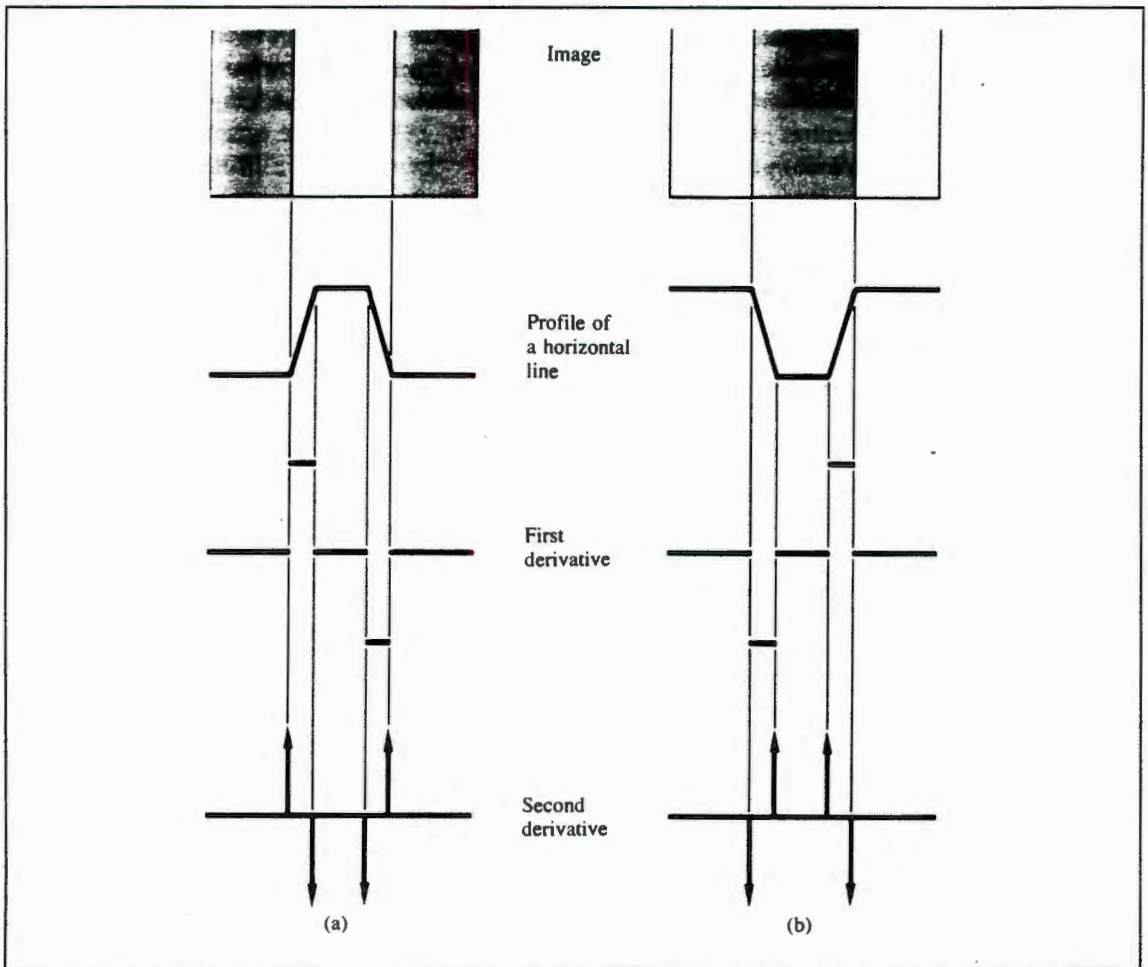


FIGURE 3.3 : Edge detection by First and Second derivative operators.

From the results of the first and second derivatives, the edge location as well as the type of edge (dark to light or vise versa) can be found. The first derivative will give

a high output from the beginning to the end of the ramp. It can be seen that if the ramp spans over many sample points no direct maximum point can be found for the edge location. Since there is a positive or negative peak on either side of the edge, by using the second derivative the edge can be located simply by finding the zero crossing [32]. Again if the ramp spans over many sample points no single zero crossing can be found for the location of the edge. The first derivative of the edge profile can be found by gradient based operators.

3.4.1. GRADIENT OPERATORS

To find the first derivative of an image we must look at the gradient of the grey level from pixel to pixel. For the given image function $f(x,y)$, the gradient of f at the location (x,y) is defined by the vector :

$$GRAD(F) = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \tag{3.1}$$

From this the direction φ at which the gradient has the greatest magnitude is :

$$\varphi = \arctan\left(\frac{\frac{\partial f}{\partial y}}{\frac{\partial f}{\partial x}}\right) \tag{3.2}$$

and has a magnitude $G[f(x,y)]$ of :

$$G[f(x,y)] = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} \tag{3.3}$$

To calculate the first derivative in the x and y directions we can use first differences [23].

$$\begin{aligned} (\Delta_x f)(x,y) &\equiv f(x,y) - f(x-1,y) \\ (\Delta_y f)(x,y) &\equiv f(x,y) - f(x,y-1) \end{aligned} \tag{3.4}$$

These can be implemented by the convolution of the image f with the following masks. Mask (a) for the difference in the x direction and (b) for the difference in the y direction.

$$\begin{matrix} [-1\ 1] \\ \text{(a)} \end{matrix} \qquad \begin{matrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} \\ \text{(b)} \end{matrix} \tag{3.5}$$

In calculating the gradient of f any pair of perpendicular directions can be used.

For the directions of 45° and 135°, the maximum gradient is found by :

$$G[f(x,y)] = \sqrt{(f(x+1,y+1) - f(x,y))^2} + \sqrt{(f(x,y+1) - f(x+1,y))^2} \quad (3.6)$$

using the coordinates of :

f(x,y+1)	f(x+1,y+1)
f(x,y)	f(x+1,y)

which can also be implemented at less computational cost by :

$$G[f(x,y)] \approx |f(x+1,y+1) - f(x,y)| + |f(x,y+1) - f(x+1,y)| \quad (3.7)$$

This operator defined by equation (3.6) and (3.7) is known as the *Roberts* operator [23]. Because this operator only uses 4 points it is extremely sensitive to noise and surface irregularities [8]. This results in the emphasis of high frequency noise as with all derivative operators. The higher the order of the derivative the more pronounced is the effect [16]. Thus in order to reduce the noise, smoothing of the image *f*, first has to be done. This can be achieved by averaging the grey level values in a 3 by 3 neighborhood. This is implemented by the convolution of the image *f* with the mask :

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad (3.8)$$

The greater the size of the mask, the bigger the neighborhood that is averaged and thus the greater the smoothing. Smoothing can also be achieved through the implementation of a median filter, the operation of which is described later in chapter 6 section 6.6. Smoothing can also be incorporated into the difference mask by

averaging in one direction and differencing in the perpendicular direction, this will be discussed later.

3.4.2. THE LAPLACIAN OPERATOR

The second order derivative can be defined by the *Laplacian* operator [10] :

$$L[f(x,y)] = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \qquad (3.9)$$

which can be implemented by the convolution of the image f with one of the following Laplacian masks [21] :

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \qquad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \qquad \begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix} \qquad (3.10)$$

(a) (b) (c)

The more commonly used Laplacian mask (3.10a) averages local pixel points in the x and y directions and is orientation-invariant. As seen in figure 3.3 the operator responds to the start and end, of the ramp of the edge, but the response is even stronger to corners, lines, line ends, and isolated points. Rosenfeld states in [23] that the response of the Laplacian is not truly orientation-invariant with the response to a diagonal edge being twice as strong as that of the response to horizontal and vertical edges. Because the Laplacian is taking second differences it is very sensitive to noise and therefore this operator is usually delegated to a secondary role [10]. According to Schachter and Rosenfeld [26], generally the gradient operator responds better to edges since the Laplacian responds more to isolated pixel points (such as random noise). An operator that is less sensitive to noise while maintaining 0 output for ramped edges is the *PseudoLaplacian*. This operator takes the maximum rather than the sum of the second differences in the x and y directions.

By using second derivative methods the edge can be located by finding the point of zero crossing. Another second derivative operator is proposed by Marr and Hildreth which is modeled on the response of the human eye. This operator is a combination of a Laplacian-Gaussian [32] whose shape is shown in figure 3.4. The Gaussian acts as the smoothing function while that of the Laplacian acts as a nondirectional derivative.

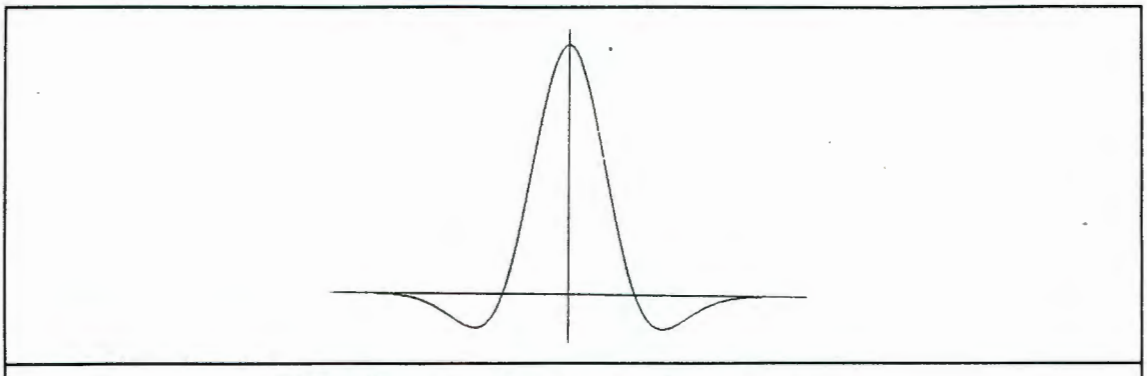


FIGURE 3.4 : A Laplacian-Gaussian mask.

The Laplacian-Gaussian operator can be modeled by taking the difference of 2 Gaussian functions and is commonly referred to as *Difference of Gaussian* (DOG) [32] :

$$DOG(\sigma_e, \sigma_i) = \left[\frac{1}{\sqrt{2\pi}\sigma_e} \exp\left(-\frac{x^2}{2\sigma_e^2}\right)\right] - \left[\frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(-\frac{x^2}{2\sigma_i^2}\right)\right] \quad (3.11)$$

Marr and Hildreth suggested that for the best approximation of the Laplacian-Gaussian was to use the ratio $\sigma_i/\sigma_e = 1.6$. Different sizes of masks are used to detect edges with predominant edges being detected in all masks. The smaller masks will have more detail since the gaussian averaging is smaller. These mask can become large (25 by 25) and are thus computationally expensive.

3.4.3. DIFFERENCE OF AVERAGES

As seen the first and second differential operators will emphasize high frequency noise. One way to reduce the noise is to smooth the image using the mask (3.8) before using the differencing operator. Alternatively we could apply an operator that will find the difference, of the local average of 2 regions. In computing local averages two non-overlapping adjacent neighborhoods are taken in order maximize the response, since if the regions overlap common values would be cancelled out in the difference operation thus weakening the response. Averaging over a local neighborhood of 2 and 4 pixels, and then taking the difference in the x direction is achieved by the convolution of the image f with the following masks :

$$\frac{1}{2}\begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix} \qquad \frac{1}{4}\begin{bmatrix} -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 \end{bmatrix} \quad (3.12)$$

Similarly taking the difference in the y can be achieved by the convolution masks :

$$\frac{1}{2}\begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix} \qquad \frac{1}{4}\begin{bmatrix} 1 & 1 \\ 1 & 1 \\ -1 & -1 \\ -1 & -1 \end{bmatrix} \tag{3.13}$$

The outputs from the above mentioned masks responds blurrily to the edge, but the response can be sharpened by nonmaxima suppression in the direction across the edge [23]. By using the masks that only average over a neighborhood of 2 pixels, parallel to the edge, the blurring effect can be further reduced but the smoothing effect is less. The correct size of averaging neighborhood is important. If a big neighborhood is chosen small edges would be averaged away and the response would be small, alternatively if small neighborhoods are taken the effect of noise will not be sufficiently suppressed.

The masks (3.12) and (3.13) are seen to be symmetric about $(x + ½, y + ½)$. In order to have masks that are symmetric about (x,y) we can use the masks commonly known as the *Prewitt* operator, which are given by [23] :

$$\frac{1}{3}\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \qquad \frac{1}{3}\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \tag{3.14}$$

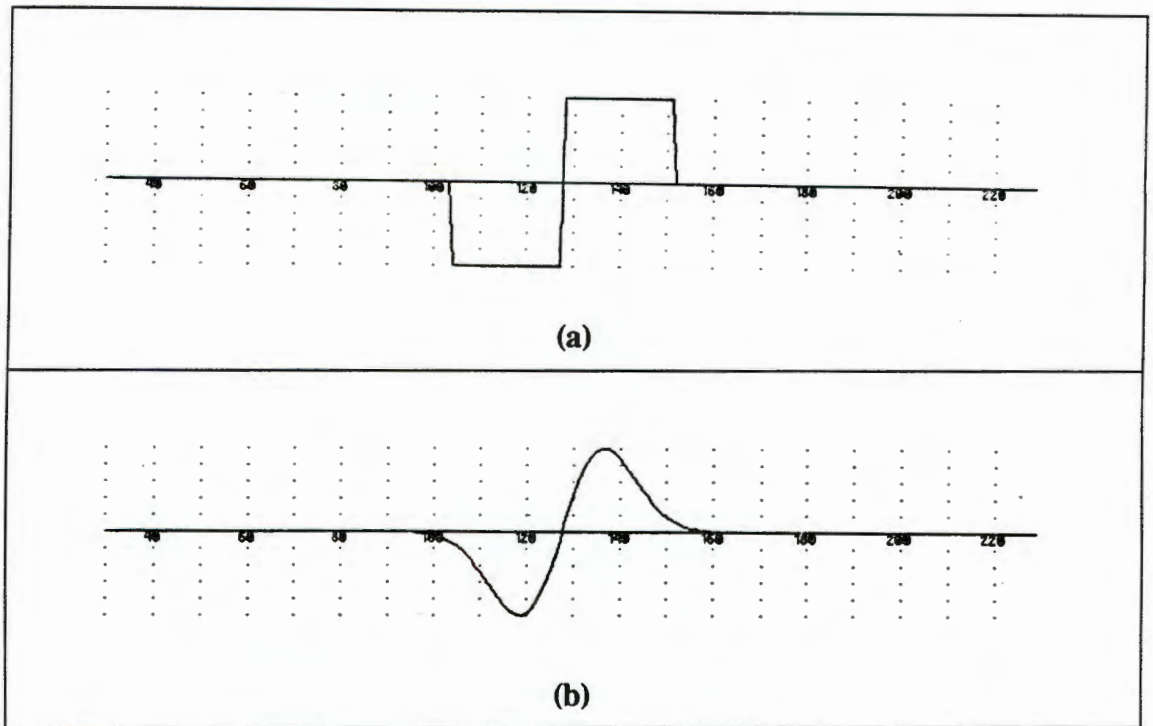
The averaging of pixel values in the masks used so far have all had equal weights of 1. A weighted average can be achieved by the use of the convolution masks commonly known as the *Sobel* operator given by [23] :

$$\frac{1}{4}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{3.15}$$

With these masks a bigger weighting is given to the pixel lying closer to the position (x,y), and will thus have a stronger response for diagonal edges than that of the Prewitt operator. Other forms of uneven weighting can also be implemented.

For the 1 dimensional case the difference of averages can be implemented by the difference of boxes operator, in which all the values have the same weighting. Canny showed in [5] that the optimal step edge operator could be modeled by the first derivative of a Gaussian operator which would have the effect of smoothing the output response of the edge detector. Figure 3.5a shows the 1 dimensional difference of boxes operator in which all the elements have equal weighting, figure 3.5b shows the Canny operator in which the elements in the difference of boxes operator have an

almost Gaussian weighting. In [23] it is also stated that the weakening of the response to a blurred edge can be reduced by operators that use Gaussian weighting patterns.



(a)

(b)

FIGURE 3.5 : (a) Difference Of Boxes operator.
(b) First derivative of a Gaussian operator.

3.4.4. EDGE MATCHING

An ideal edge has a steplike cross-section, therefore one way to detect the edge is to try and match this pattern. The edge pattern can be represented by the second difference of the step edge [23]. A mask can then be formulated from this pattern and then used for convolution with the image. From figure 3.3 we can see that the second difference for an edge from black to white will have a positive spike followed by a negative spike. The convolution mask that will represent the edge pattern will be 1 for the positive spike and -1 for the negative spike. The mask will be thus :

-1	1
-1	1

Note : the mask averages over a neighborhood of 2 pixels, and has also been rotated 180° because of the convolution process. For an edge with a ramp spanning over more sample points, there is a greater distance between the spikes in the second difference of the edge profile. There will thus be a zero crossing between the two

spikes and can be represented by a zero. This will give rise to the convolution mask as shown by [23] to be :

$$\begin{matrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{matrix}$$

This was previously shown to be the Prewitt operator. Many different kinds of masks can be formulated depending on the type and orientation of edge that is trying to be detected. Masks that can detect edges of different orientation are commonly known as *compass gradient* masks, examples of which can be found in [23, 21]. Matching masks can also be used to detect corners, lines, spots, or any other object for that matter.

The choice of mask size as with the difference of average method is also important. If a large mask is used, more noise can be averaged out but so will small edges. The optimal size of mask to use according to Young [32], can be determined by using different sizes of masks and choosing the one that gives the highest output. The output will increase with mask size until a second edge is contained within the same mask. Note : this is a very important principle and will form the basis of the algorithm design discussed in chapter 5.

Many other strategies for edge detection are used (the harder the edge is to detect, the more complex is the edge detector) but only difference based operators have been discussed. The reason for this is that difference operators is a simple, intuitive approach to edge detection. This simplicity leads to an ease of implementation and fast operation of the edge detector. The performance of a difference operator such as the Sobel operator, is equaled if not better than that of more sophisticated methods of edge detection. This was shown in a comparison of various operators carried out by Pratt [21] and an evaluation of edge detectors carried out by Christiansen [6].

In order for the successful operation of the edge detector that is to be proposed, the grey level of the ore material needs to be accurately digitized. The next chapter will discuss the hardware design that is required to do this.

4. HARDWARE DESIGN

4.1. INTRODUCTION

In order to have maximum ore recovery, the positioning of the blade separator at the bottom of the spiral concentrator, needs to be accurately and rapidly determined. From chapter 2 we see that this positioning can be automated by the implementation of a machine vision system. In figure 2.6 the basic machine vision system comprises a camera, which converts the optical image into an analog video signal, a frame grabber which digitizes the analog signal, and a computer which will analyze the digitized image.

This chapter will deal with a detailed discussion of the implementation of the hardware needed for the successful operation of the machine vision system.

4.2. HARDWARE CONFIGURATION

The optical image of the ore in the spiral is converted into an electrical analog signal by the means of a Charge Coupled Device (CCD) camera, which can be either colour or black and white (B/W). Sufficient illumination of the spiral will ensure that the camera output will be a good representation of the brightness level of the ore in the spiral. The camera output will be a composite baseband[1] video signal which will then be used as input into the B/W frame grabber. If a colour camera is being used, the video signal needs to be decoded in order to remove the colour modulation before being used as input into the frame grabber. This decoding is achieved by the use of a Red Green Blue (RGB) decoder. The frame grabber will then digitize the analog input signal to an image array typically of the size, 512 pixels by 512 pixels, and to a resolution of 8 bits. This image array can be transferred to the computer for analysis. Through a digital to analog convertor, the determined edge position can then be used to drive a motor that will position the blade separator.

Figure 4.1 shows the block diagram of the implemented hardware as discussed above.

---

[1] A Baseband signal has its biggest spectral components situated in a band of frequencies around the zero frequency. That is, the signal has not been modulated up in frequency by a high frequency carrier [18].

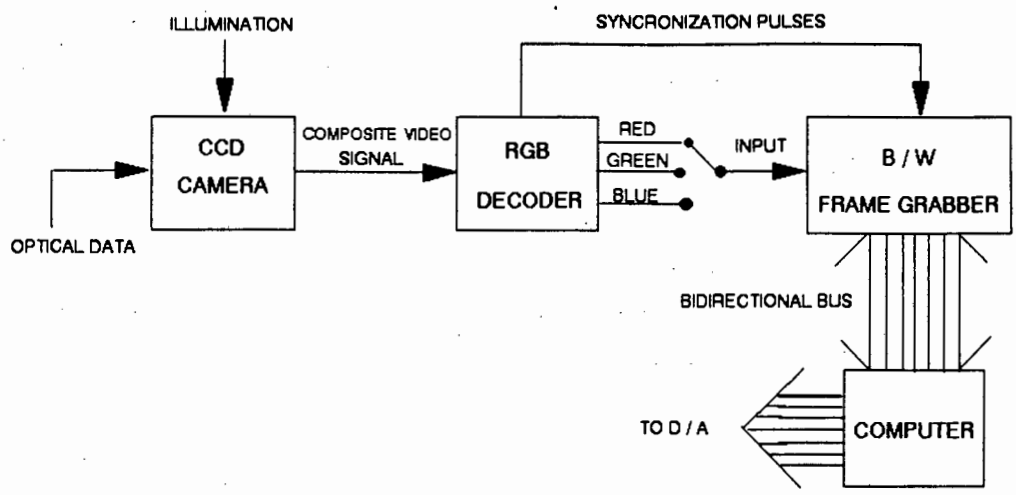


FIGURE 4.1 : Block diagram of the implemented hardware for a Machine Vision System for use on the spiral ore concentrator.

The operation and implementation of each block of hardware will now be discussed in detail.

4.3. CCD CAMERA

In order to convert the optical image into an analog video signal a CCD camera was used. The reason for this is that CCD cameras are inexpensive, small, light, and robust compared to that of Cathode Ray Tube (CRT) cameras. The CCD camera consists of a N * M matrix of photosensitive elements ranging in size, typically from 180 * 180 elements to 1024 * 1024 elements. Bigger sizes than this have also been successfully implemented [12]. As light falls on the array of photosensors each element will be charged to a certain electrical potential. Each element is then read out by a clock pulse, and its stored charge is then transferred to an output amplifier. A video signal is then built up by measuring the electric potentials as they arrive in sequential order at the output [31].

If the different types of ore in the spiral have similar light intensities but of different colour, a colour camera would have to be used, since the two types of ore would look the same using a B/W camera. With the use of a colour camera, the three primary colour components of the ore, Red, Green, and Blue, could be split up and analyzed separately. This however would necessitate the use of an RGB decoder. The reason for this will be further discussed in section 4.5. If the two types of ore in the spiral have light intensities that are distinguishable from one another, a B/W camera would

be sufficient to differentiate between the two. A B/W camera would not need the use of an RGB decoder and so costs would be kept to a minimum in large scale implementation.

For a laboratory analysis of the differing concentration levels of ore in the spiral, a video tape was taken by a commercially available colour camera, the HITACHI VM-500E. This camera was placed as directly overhead as possible to the chute in order not to distort the image. In the laboratory all development was done by using this video tape played back through a JVC HR-D700E video recorder. This recorder was used since it had a very stable pause facility and thus enabled a frame by frame analysis of the ore in the spiral. When the system is ready to be implemented on the spiral, the video recorder can simply be substituted by the CCD camera.

4.4. ILLUMINATION

Illumination is critical for any machine vision system. If the illumination is uneven, shadowing and glinting of the object could occur. This results in complex algorithms that will try and interpret and correct for the lighting inadequacies, and thus make the machine vision system slow. In chapter 2 the ore slurry was seen to be wet and enclosed within the raised sides of the chute. Thus in the implementation, lighting will have to be provided from both sides of the chute in order to eliminate shadowing from the chute sides. Also soft [7] or diffused lighting will have to be used in order to suppress glinting from the wet bubbles that are formed on the top of the slurry.

The video material was taken using two 500 W halogen bulbs which were positioned unevenly on opposite sides of the spiral. The uneven positioning results because no direct attachment points on the spiral could be found for the lighting. This unevenness resulted in a slight shadow on the right edge of the chute, and the halogen bulbs did produce some glinting off the bubbles. In order to have a robust machine vision system the algorithm design would have to take this into account.

4.5. RGB DECODER

For a laboratory analysis of the spiral operation, it was seen in section 4.3 that a colour video tape was made and played back through the video recorder. The composite[1] colour video signal was created using the Phase Alternating Line (PAL)

[1] A composite video signal is made up of the video information, as well as frame and line synchronization pulses.

system. In this system the video signal is made up of a combination of the Red (R), the Green (G), and the Blue (B) channel outputs from a colour camera. These colours are modulated onto a higher carrier frequency and then added to a predetermined brightness or luminance level (L).

If this signal is then used as input into the B/W frame grabber, the digitized image will be a combination of all three colours and not just the brightness or luminance level which is used by the B/W frame grabber. Therefore before being used as input, the video signal needs to be split back into the three primary colours R, G, and B, as well as its luminance level Y. In order to do this we need to have a basic understanding of the PAL transmitter, as well as the receiver. This will now be discussed.

4.5.1. PAL TRANSMITTER

There are three outputs from a colour camera R, G, and B which are used as inputs into the PAL transmitter. The transmitter generates a signal which can be used by colour, as well as B/W systems. Thus a luminance signal is formed for use by B/W systems and then modulated colour information is added to this for use by colour systems. The construction of the luminance signal is achieved by the following :

$$Y = 0.229R + 0.587G + 0.114B \qquad (4.1)$$

The reason for these weightings is that the eye is twice as sensitive to green as that of red, and three times more sensitive to red than that of blue. After combination, the Y channel will have the same spectral response as that of the eye [15]. Since the luminance signal has to be transmitted for B/W systems only two other colour components need to be added to make up the composite colour signal, they are the difference signals (R - Y) and (B - Y). These difference signals are used to ensure no brightness information is transmitted in the chrominance channels [28]. The two chrominance signals are thus formed by the following :

$$U = K_R * (R - Y) \qquad (4.2)$$
$$V = K_B * (B - Y) \qquad (4.3)$$

where K_R and K_B are scaling factors in order to ensure the video signal has a

maximum peak to peak value of 1 V. Their values are :

$$K_R = 0.877$$
$$K_B = 0.493$$

The U and V chrominance signals are band limited to 1.4 MHz and then modulated onto a colour carrier of frequency 4.4336 MHz. The U is modulated by $\sin(\omega t)$ and V with $\sin(\omega t \pm 90°)$. The U, V, and Y signals are combined to form the composite colour video signal COMP expressed as follows :

$$\begin{aligned} & COMP = Y + U + V \\ \text{were} \quad & U + V = \sqrt{U^2 + V^2}.\sin(\varpi\tau + \alpha) \\ \text{were} \quad & \alpha = \arctan\left(\frac{U}{V}\right) = Hue \\ & \\ LET \quad & S = \mathrm{mod}(U + V) = Saturation \\ THEN & \\ & COMP = Y + S.\sin(\varpi\tau + \alpha) \end{aligned} \qquad (4.4)$$

The block diagram for the various stages in the PAL transmitter is shown in figure 4.2.

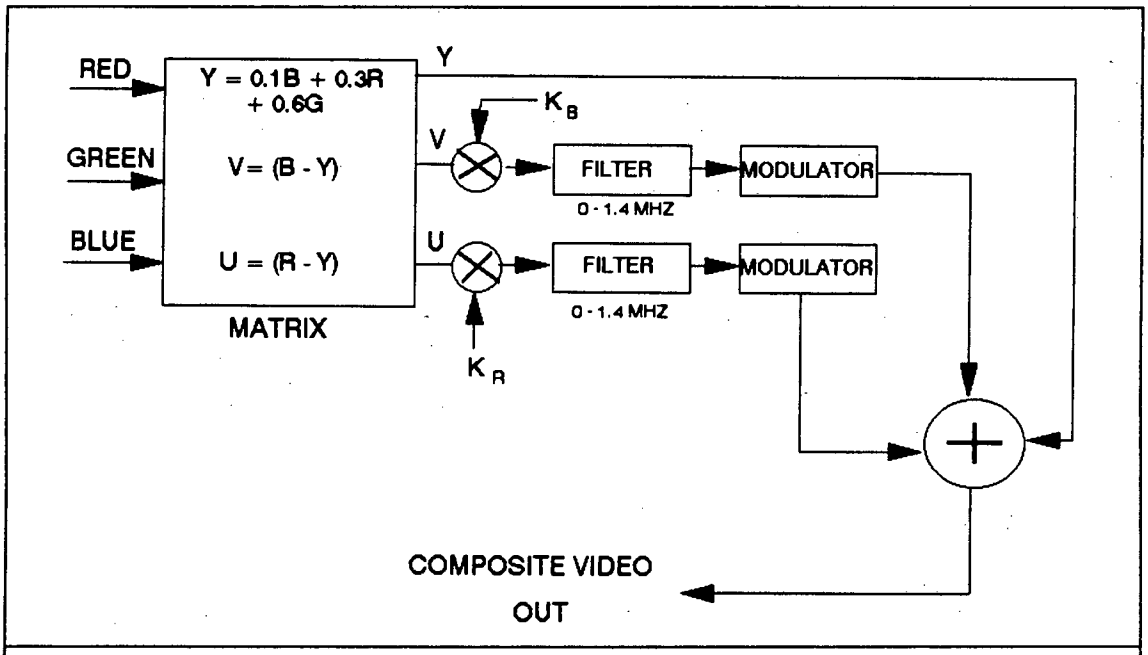


FIGURE 4.2 : Block diagram of PAL transmitter.

It can thus be seen that the input to the frame grabber cannot be the composite colour signal but should be either the luminance signal Y or a colour signal R, G, or B. These components can be split up by an RGB decoder whose operation is the same as that of the PAL receiver.

4.5.2. PAL RECEIVER

The composite video signal COMP is used as an input into the receiver were it is passed through two filters, a carrier reject and carrier pass filter. The output signal from the carrier reject filter will form the luminance Y signal, and the signal from the carrier pass filter will be the modulated U and V signals. The U and V signals are demodulated and scaled back to their original values by the multiplications of :

$$U * 1/K_R = (R - Y) \quad (4.5)$$

$$V * 1/K_B = (B - Y) \quad (4.6)$$

These signals will then be added to the luminance signal Y to yield the R and B signals. The G signal is reconstructed by the following weightings [13] :

$$G = 1.67Y - 0.5R - 0.17B \quad (4.7)$$

It can be seen that the blue signal is attenuated by $1/K_B = 1/0.493 = 2.028$ and the red signal is attenuated by $1/K_R = 1/0.877 = 1.140$. From equation (4.7) the attenuation for the green signal is [13] :

$$G_{atn} = 1.67 - 0.5*1.14 - 0.17*2.02$$
$$= 0.757$$

Therefore the blue signal has the biggest attenuation and thus is the most susceptible to noise, whereas the green signal has the least attenuation and will be the least affected by noise. A block diagram showing the various stages of the PAL receiver is shown in figure 4.3.

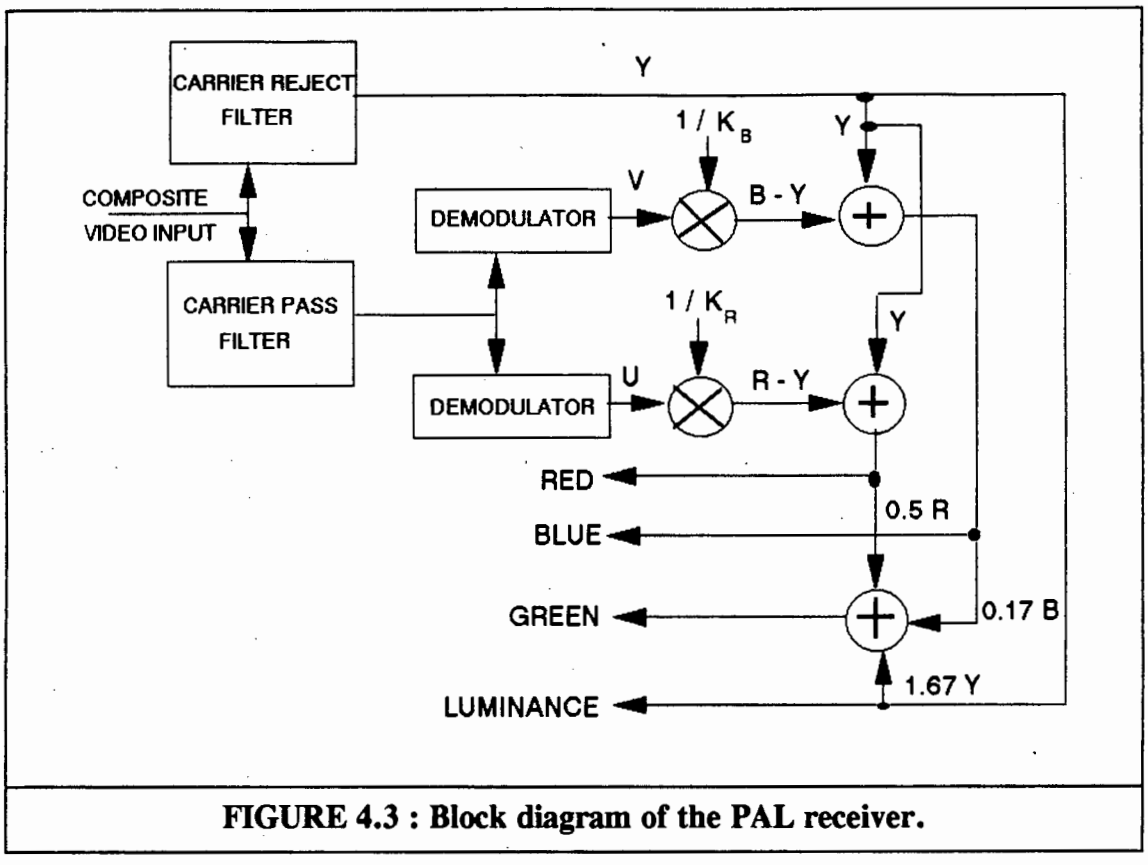


FIGURE 4.3 : Block diagram of the PAL receiver.

The operation of the RGB decoder is the same as that of the PAL receiver and will split the composite video signal into the Y, R, G, or B signals. Either one of these signals can now be used as input into the B/W frame grabber.

The RGB decoder used was the ELV 7000 series VCP 7001 video colour processor. This decoder was used because it was able to be assembled in "kit" form, thus keeping costs low. The decoder outputs the R, G, and B signals, whose gain can be externally controlled. The amount of colour saturation was also controllable and if it was turned completely down, each of the R, G, and B signals will output the luminance signal Y. Decreasing the colour saturation has the same affect as decreasing the values of K_R and K_B, by the same amount. If K_R and K_B are decreased to 0, the difference signals (R - Y) and (B - Y) will be 0, and thus the R, G, and B signals will be seen to only output the luminance signal Y.

4.6. FRAME GRABBER

A colour frame grabber consists essentially of 3 B/W frame grabbers, one to digitize each primary colour. Therefore in order to keep costs low, a B/W frame grabber is used. The frame grabber used was the MATROX PIP-1024B Video Digitizer Board,

which is capable of digitizing the composite video signal into an integer[1] number array of the following formats, 512 * 512, 512 * 576, and 1024 * 1024, all to a resolution of 8 bits or 1 byte. This array is then displayed on an output monitor. The board was used in the 512 * 512 format thus providing an image of 512 * 512 pixels. Each pixel has a value specified by 1 byte, therefore it can take on a value between 0 and 255 or 256 grey levels. This frame grabber has 3 inputs which are software selectable through a multiplexer, thus enabling multiple camera inputs. The frame grabber obtains its synchronization (sync) for digitization from the input composite video signal, from which it extracts the line and frame sync pulses. The frame sync pulse is used as a reference point for start of the number array and the line sync is used as a reference for the start of each row in the array.

As seen from the previous section two types of signals can be used as input into the frame grabber. They are either a B/W signal from a CCD camera, or the R, G, B, or Y signals from the RGB decoder. If the B/W signal is used no external synchronization is required since it is a composite video signal which already contains the line and frame sync pulses. However the output signals of the RGB decoder do not have sync pulses, therefore they need to be externally provided to the frame grabber. The decoder output signals have the same phase as its input signal, therefore the input composite video signal to the RGB decoder, can be used as the composite sync pulse train[2] for the frame grabber. This can be done by changing the jumper of 19-20 to be IN [20] on the Matrox board, thus enabling a separate sync input to be connected to the middle input channel of the board. The video information present in this composite video signal will not affect the frame grabber since it only needs to extract the synchronization pulses which are at set frequencies. Because the video material used in the analysis is colour, the input to the frame grabber will come from the RGB decoder and external synchronization will come from the output video signal, from the video machine. The frame grabber is also software controllable through the programing language Microsoft C.

4.7. MICROPROCESSOR CONTROL

The microprocessor used was the 386-SX personal computer running with a 16 MHz clock. The computer was considered fast enough for the development of the machine vision system.

---

[1] A number with no decimal places.

[2] A combination of frame and line synchronization pulses.

Software was written in Microsoft C version 6 using the small memory model. Microsoft C was used in order to link into the software libraries of the frame grabber. The small memory model was used because it uses 16 bit address pointers to memory rather than 32 as in the large memory model, thus making memory access faster. By linking into the software libraries of the frame grabber the computer will have access to all its software routines, thus having total functional control over it. Once software has been written and successfully compiled to a .OBJ file, it is linked to the small memory model library of the frame grabber (MS.LIB), by using the Microsoft linker (LINK.EXE). Note : the stack size needed for the frame grabber is 4096, but because of large arrays and many variables of the written software it is set to 6000 by using the /ST:6000 option.

The frame grabber is software initialized to operate in the 50 Hz European format, and the non-zoom option is specified thus enabling four 512 * 512 images to be stored. These images of 512 * 512 are held in the frame buffer which is then selected to be displayed to the output monitor. The input channel to the frame grabber is channel 2. The frame buffer is capable of storing four images, but the frame grabbers software will only allow one image to be manipulated at any one time, therefore in the processing only one 512 * 512 frame is used. In the final implementation a cheaper frame grabber with less memory, such as the MATROX PIP-512B Video Digitizer Board could be used.

The frame buffer is updated by taking a snapshot[1] of the input signal, thus displaying a new image on the output monitor. This is done continually in order to have a real time display. In order for the image to be processed, the digitized image or part thereof, is transferred to the computer memory by the function BTRANSFER(). This function transfers a specified number of bytes to a specified location in computer memory from a specified start location in the frame buffer. The shortest transfer time for one pixel using a IBM AT computer is by using the Input/output transfer. The average time to do this is 1.8μs [20]. Once the computer has processed the image and determined the edge position, this value can be used as output through a parallel port to a digital to analog converter. This would then control a motor which would drive the blade to the correct cutting position. This however was not done since the final implementation of the machine vision system onto a spiral concentrator was beyond the scope of this project.

---

[1] A snapshot is invoked by the frame grabber instruction SNAP() which updates the output frame buffer with the new digitized image.

4.8. DETAILED ANALYSES OF THE VIDEO SIGNAL

For the laboratory implementation the hardware is set up as described in the previous sections. To summarize, the video recorder output signal was used as input into the RGB decoder, whose output was then connected to the frame grabber. External synchronization was provided to the frame grabber from the output of the video recorder. One row or 512 pixels of the digitized image was then transferred into computer memory for analyses.

4.8.1. ELIMINATION OF COLOUR MODULATION

To see the effect of the colour modulation introduced by the PAL system, the composite video signal is directly used as input into the B/W frame grabber. Figure 4.4 shows the grey level profile for 1 row of pixels, of a frame taken of the ore in the spiral, for a ore concentration level of 2 Kg. The ore in the spiral can be identified between the pixel locations of 100 - 358, were it ramps up from a low level plateau to a high level plateau. The pixel locations between 0 - 99 and between 359 - 512 can be identified as the sides of the chute.

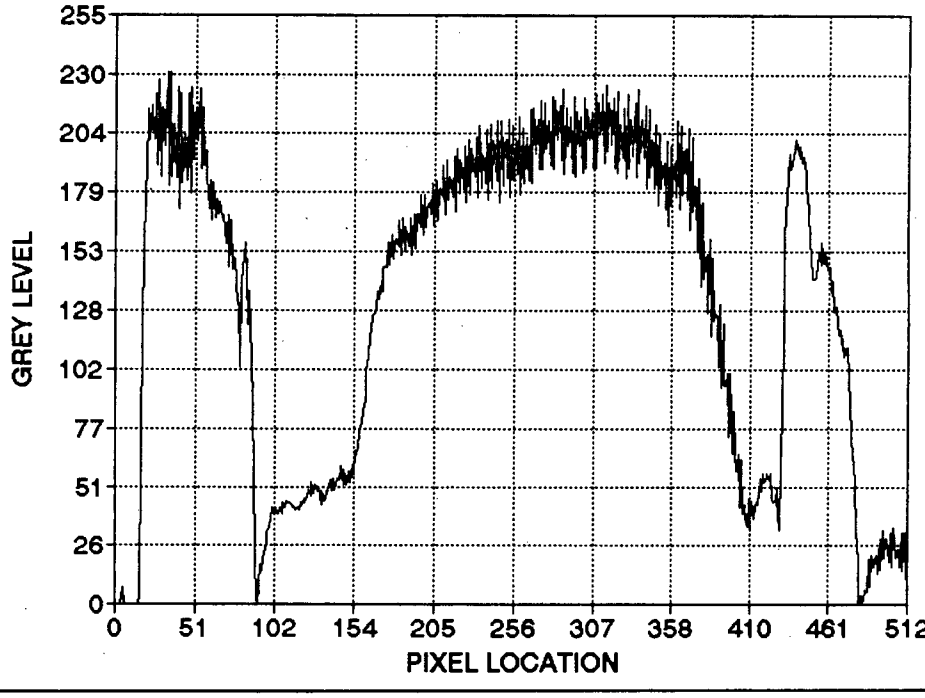


FIGURE 4.4 : Grey level profile for 1 pixel row of a colour frame taken of the ore in the spiral.

From equation (4.4) the composite video signal is made up from the addition of the luminance signal and the modulated colour information. In figure 4.4 the average

height of the ore's grey level profile will be the luminance level, and the high frequency variations in grey level will be the modulated colour information added to it. The pixel locations between 25 - 95 and 165 - 358 are identified on the frame grabber display monitor, to be regions that are almost white, and in figure 4.4 these regions have a high luminance level with a large high frequency variation added to it. For the display to be white, the R, G, and B signals are equal and at a maximum (ie R = G = B = 100%), therefore their addition will result from equation (4.1) in the luminance signal being at a maximum (ie Y = 100%) [28].

The modulated colour information, which is added to the luminance level, is derived from the R and B signals. In a white region, the R and B signals are at a maximum (R = B = 100%), therefore the difference signals U and V as derived from equations (4.2) and (4.3), will also be at a maximum. The amplitude of the modulated colour information will thus be a maximum, since it is derived from the Saturation S level which is obtained from the difference signals U and V.

In order to have an accurate representation of the grey level profile of the ore, the colour modulation is removed by the RGB decoder. The luminance signal is now used as input into the frame grabber. Figure 4.5 shows the luminance output for the same ore profile as in figure 4.4.

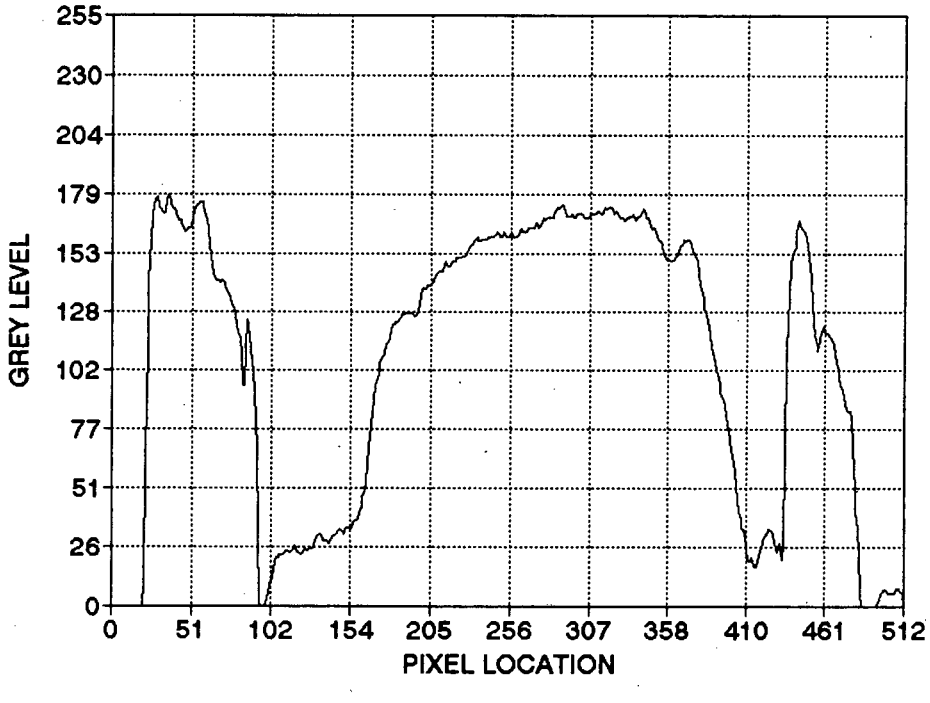


FIGURE 4.5 : Grey level profile for 1 pixel row of the luminance input of a frame taken of the ore in the spiral.

All the high frequency colour modulation. has been removed leaving only the brightness value of the ore in the spiral. The height of the grey level profile is less than that of figure 4.4. This is due to attenuation introduced by the RGB decoder but can be corrected by an adjustment of the gain in the decoder output. Since the decoder has also extracted the 3 primary colours, each colour can now be examined.

4.8.2. RGB CHANNEL RESPONSES

Each of the R, G, and B signals were used as input into the frame grabber, for the same image frame as in figure 4.4. Figure 4.6 shows the ore grey level profile for 1 pixel row of the R input. It is seen that the ore in the spiral is from pixel location 100 - 358. The black material is seen to have no red component as can be seen by a grey level of 0 between the pixel locations 100 - 160. The white material however has a large red component with a high grey level of about 150 between the pixel location 256 - 358.

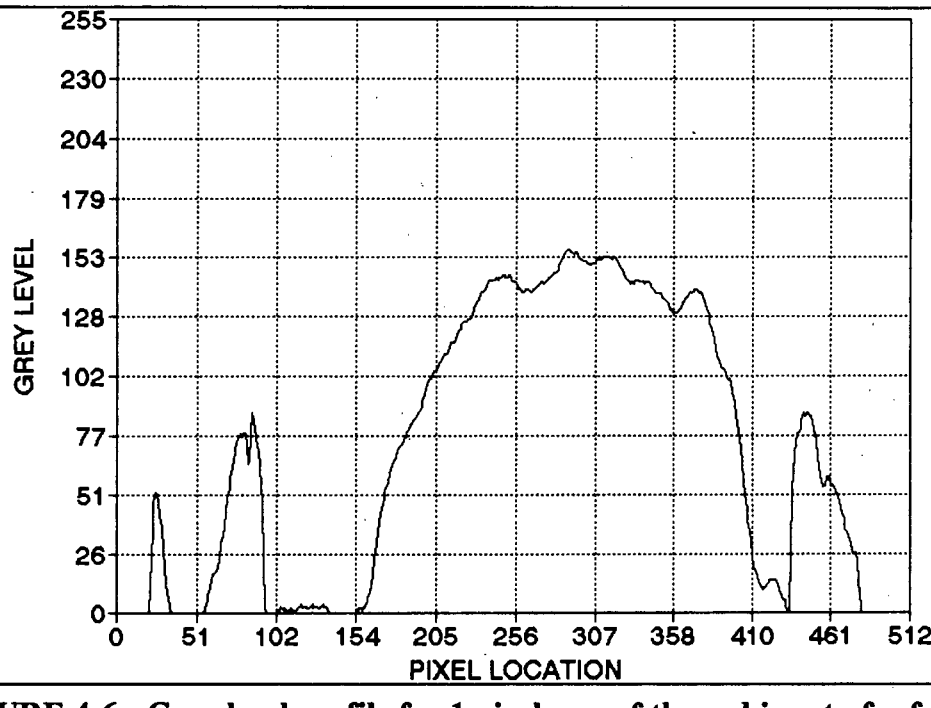


FIGURE 4.6 : Grey level profile for 1 pixel row of the red input of a frame taken of the ore in the spiral.

Figure 4.7 shows the G input. It can be seen that the black material has a small component of green, with a higher grey level compared to that of the R input, but the white material has a smaller component of green compared to that of the R input. In figure 4.8 we see that the black material has a small B component but the white material has none. Interestingly enough the transition from black to white, from

pixel position 154 - 205, is seen to have a higher B component than that of black material alone.

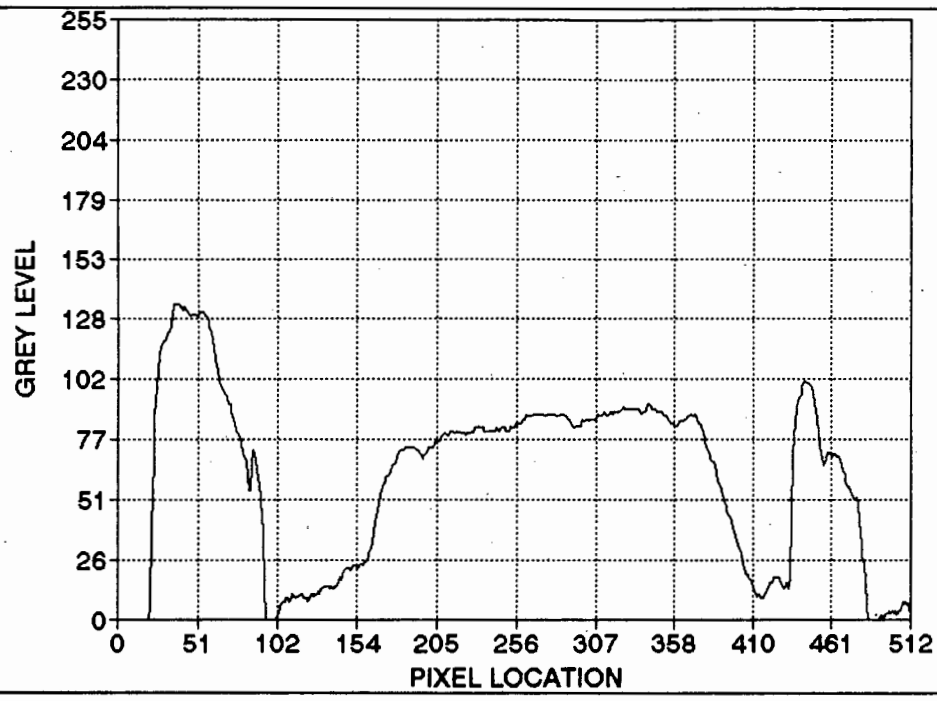


FIGURE 4.7 : Grey level profile for 1 pixel row of the green input of a frame taken of the ore in the spiral.

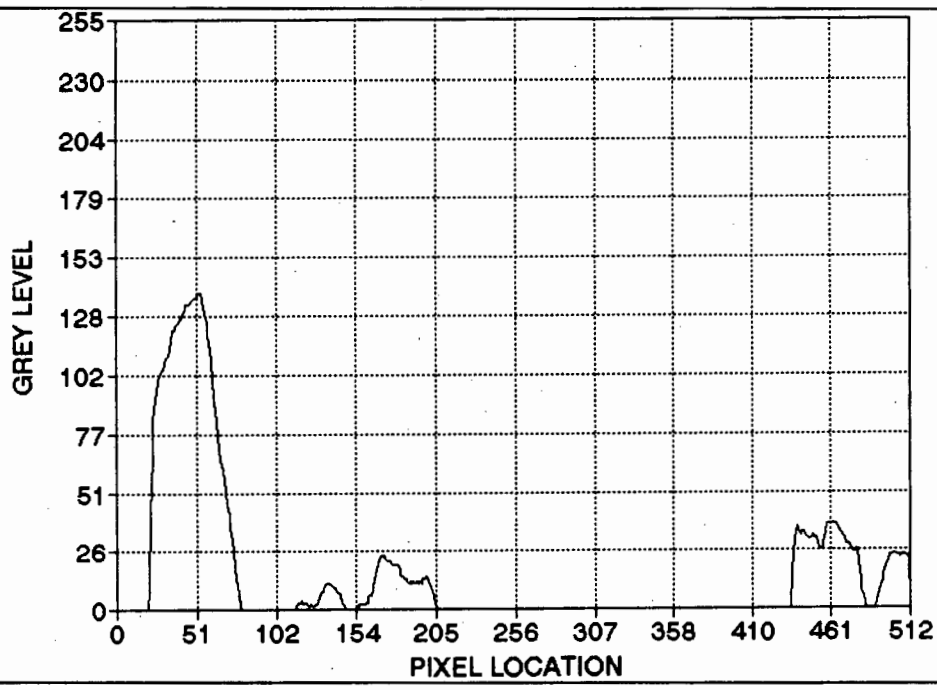


FIGURE 4.8 : Grey level profile for 1 pixel row of the blue input of a frame taken of the ore in the spiral.

It was observed from the displayed images on the frame grabber monitor that the G channel had the least amount of noise while that of R had more, and B the most. This was in accordance with the various attenuation levels as discussed in section 4.5.2. From the above figures, the differing colour responses will yield information about the nature of the ore. But since there is a large intensity difference between the 2 ore types being used, this colour information was not needed. Thus for the rest of the discussion only the luminance signal output from the RGB decoder will be used as an input to the frame grabber. The final hardware setup for the laboratory will thus be the composite colour video signal output, from the video recorder used as an input to the RGB decoder, whose output will then be into the B/W frame grabber. The digitized image will then be transferred into the computer for analyses.

Because the luminance signal is being used, all of the results obtained in the rest of the thesis will hold true if the video source was a B/W camera. Once the final machine vision system is implemented, the video recorder can be replaced by a B/W CCD camera and the RGB decoder can be omitted.

We now have an accurate representation of the grey level of the ore in the spiral, and now need to formulate an algorithm that will detect the outer edge position of the concentrate material. This is discussed in the next chapter.

5. EDGE DETECTION ALGORITHM DESIGN

5.1. INTRODUCTION

From the fundamental theory of edge detection discussed in chapter 3, it was seen that classical edge detectors such as the Prewitt and Sobel operators, average and difference over small local neighborhoods, by using convolution masks that extend over an area of 3 * 3 pixels. These operators work well for edges with well defined high and low plateaus, and transition ramps that span over a few sample points. Since the ore flowing down the chute is subject to many forces, the edge profile of the ore is constantly changing. Sometimes it does have well defined plateaus with a short transition ramp, but a few frames later it will have no lower plateau and a ramp that spans over many sample points. For a ramp with such a large extent, the classical operators will give a poor response since they are specifically designed to operate over small local neighborhoods.

This chapter will deal with the design and implementation of an edge detector that will operate over a global neighborhood and give consistent results even with edges that spread over many sample points. The design of the algorithm will be based on fundamental methods discussed in chapter 3.

5.2. ALGORITHM DESIGN

In figure 4.5 we saw the edge profile for a row of pixels spanning across the whole frame. The edge profile can be seen to be between the pixel locations 100 - 358, with a ramp extending over about 50 sample points. This edge profile was considered to be a good representation of an edge, since it had well defined lower and upper plateaus, as well as a relatively small spanning ramp. Other edge profiles have been seen to have a very small lower level plateau and a ramp that spans over nearly 140 sample points to the upper level plateau.

The Sobel operator described by mask (3.15) is a first difference operator and so detects points of greatest slope. If the ramp of the edge is small, the operator will output a sharp peak at the point of greatest slope, but if the ramp is long the operator output will be high at the start of the ramp and remain high throughout the length of the ramp, thus forming a long high plateau. This would cause uncertainty on the placement of the edge which should be at the middle point of the long high plateau.

A second difference operator such as the Laplacian operator, described by mask (3.10a), will also give an unsatisfactory result. The response will be a positive spike at the start of the ramp, followed by a constant 0 for the duration of the ramp, and the end of the ramp being marked by a negative spike. For ideal small spanning edges, there should only be one zero crossing at the edge location, but for edges that span over many sample points, no single zero crossing will be found. Also high frequency noise will be emphasized by the second difference. We thus need to design an edge detector that will always give a peaked response to an edge that spans over a large number of sample points.

According to Canny [5], when designing the optimal edge detector there are three performance criteria that need to be observed. They are :

1. Good detection. By maximizing the signal-to-noise ratio, there should be a low probability of failing to detect a real edge, and a low probability of falsely marking a nonedge point.
2. Good localization. The detected edge should be as close as possible to the center of the true edge.
3. Only one response to a single edge. Only one dominant peak should be present in the edge detectors output.

From figure 2.5 we see that only 1 edge has to be detected from 2 global neighborhoods, one predominantly black and the other predominantly white. Using small operators that find narrow local edges will not work, we therefore need to use a large operator that compares the 2 global neighborhoods to find the edge. The method proposed is based on finding the *Difference of Averages* between the 2 neighborhoods.

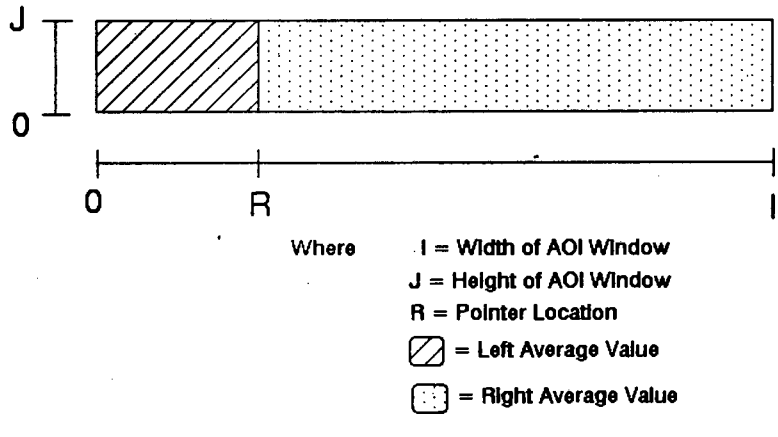
5.2.1. DIFFERENCE OF AVERAGES

According to Rosenfeld [25, 24], edges can be detected by comparing the average grey levels of 2 nonoverlapping neighborhoods. The size of the neighborhood will determine the width of the edge to be detected. To detect small edges, small neighborhoods must be used, but to detect edges between textured or noisy neighborhoods, large neighborhoods must be used, in order for large neighborhood averaging or gross averaging, of the texture and the noise. This method of edge detection forms the basis of all difference based edge detectors where averaging over

large neighborhoods has been successful in detecting edges in textured and noisy regions [27, 19, 8].

In figure 2.5 we see the two neighborhoods of black and white ore are separated by a slow curving edge in the vertical direction. The ore has surface irregularities due to the uneven distribution of the ore, as well as the ore's granular texture. By averaging the 2 nonoverlapping neighborhoods, surface irregularities can be reduced thus maximizing the signal-to-noise ratio. By finding the maximum difference between the 2 neighborhoods in the horizontal direction will yield the edge location. We could do this through a convolution of the image with a mask similar to the one given by mask (3.12). We could start off by taking a neighborhood of 2 pixels, then 4, then 6 and so on. The convolution will yield a high response at the edge position, but the response will become stronger and more peaked, as the mask averages over the whole neighborhood. If the mask is bigger than the neighborhood the response will start to get weaker since the average will incorporate the adjacent neighborhood. The problem with this method is that it assumes the neighborhoods are of the same size, but from figure 2.5 we see that the black material has a smaller extent than that of the white. Also it is computationally expensive to generate bigger and bigger masks and repeat the convolution process. The edge detection algorithm proposed to alleviate these problems is as follows :

Since the camera will be stably mounted over the spiral chute, a stable image of only the sides of the chute and ore inside the chute will be obtained. Because the sides of the chute will cause errors when calculating the average grey level of the neighborhoods, we must first define a fixed *Area Of Interest* (AOI) window that will only include the black and white ore in the spiral chute. The height and width of the AOI window are defined as follows :



(5.1)

If the height J of the window is small enough, the edge may be found by taking the horizontal difference since the curvature of the edge will be negligible compared to that of the window size. The AOI window will extend over the whole of the black and white regions, with the vertical pointer R dividing the AOI window into left and right neighborhoods. The question is, at what point should R be situated ? The optimum position can be found by moving the pointer from left to right, incrementing by 1 pixel position, then after each increment calculating the difference between the 2 average grey levels of the neighborhoods. This is implemented by the following algorithm :

$$DOA(x,y) = \frac{1}{J(I-R)} \sum_{x=R+1}^{x=I} \sum_{y=0}^{y=J} f(x,y) - \frac{1}{JR} \sum_{x=0}^{x=R} \sum_{y=0}^{y=J} f(x,y) \quad (5.2)$$

where

x = *Column Number* = $0 \ldots I$
y = *Row Number* = $0 \ldots J$
R = *Pointer Position* = $1..I-1$

When R = 1 only black material will be present in the left neighborhood or left average value (LAV), but in the right neighborhood or right average value (RAV) there will be black, as well as white material thus bringing the RAV down, so the difference between the RAV and LAV will be small. As the pointer moves further to the right less black material is in the RAV and so its average value will increase, in turn a better estimate is obtained for the black material in the LAV. At the optimum transition point only white material will be present in the RAV, therefore it will be at a maximum, and only black material will be in the LAV, and so be a minimum[1], thus the difference between the two will be a maximum. This will be the edge position. Once the pointer moves past this position white material is now present in the LAV, bringing its average up, thus decreasing the overall difference between the two neighborhoods.

Figure 5.1 shows the *Difference Of Averages* (DOA) edge detector response to a generated edge. The AOI widow is chosen to have width of I = 300 pixels and a height J = 10 pixels. The thick line in the graph shows an edge profile which ramps up slowly and has been corrupted by noise. Since the AOI window has a height of 10 pixels, the edge profile has been obtained by vertically averaging the pixels through the height of J and so makes the edge appear less noisy. The smooth line which peaks at the middle of the ramp is the output of the DOA edge detector, and as a comparison the very noisy output of the Sobel edge detector can be seen at the bottom of the graph. The Sobel operator was used as a comparison since it has been

[1] Remember the black's grey level is 0 and white's grey level is 255.

shown by Pratt [21] and Christiansen [6] to have equaled if not better performance than other sophisticated methods of edge detection. The Sobel operator calculates the edge response in the x and y directions and finds its magnitude by equation (3.3). Because the Sobel edge detector operates over such a small area, it will not sufficiently average the noise and thus has a very noisy output. However the DOA edge detector gross averages neighborhoods, thus smoothing the noise completely away and so gives a clear peak at the edge location.

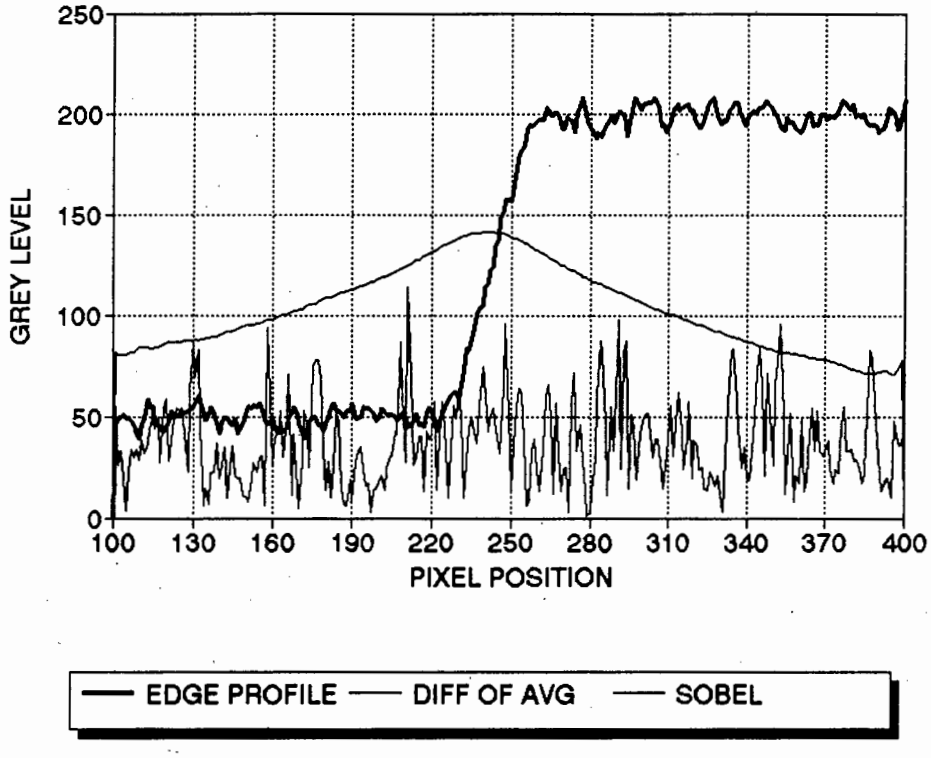


FIGURE 5.1 : A ramped edge with the Difference Of Averages edge detector output.

The output from this edge detector satisfies all the performance criteria given by Canny. It has good detection, by maximizing the signal-to-noise ratio, good localization, with the edge located at the middle of the ramp, and it also has only one response, to a single edge. We see that this method automatically selects the optimum size of the two neighborhoods, by the moving pointer R. Since we are using all the information in the AOI window about the neighborhoods, any texture, irregularities, or noise present in the window will be averaged away and so have negligible effect.

This method is also analogous to template matching were the shape of the edge is modeled by a mask. The AOI window can be considered to be a mask with the values to the left of R to be -1, and to the right 1. By increasing R the lower plateau of the black material will be matched and at the same time so will that of the white material. At the point of highest response, the template will exactly approximate the edge profile with the length R being the length of the black material, and the length (I - R) that of the white material. We need not convolve each mask over the whole frame area (as described in chapter 3 section 3.4.4), since the predefined AOI window already encloses the 2 neighborhoods and convolving outside these neighborhood would be useless and time consuming. We are however performing a convolution within the AOI window but at only one point. By only moving the pointer R and performing the convolution at that point we will ensure the maximum response possible at the correct position of the edge, since the template at position R will best approximate the profile of the edge.

5.3. IMPLEMENTATION OF ALGORITHM AND RESULTS

The algorithm described in equation (5.2) was implemented in Microsoft C software. The software required an initializing stage were the top left, and bottom right coordinates of the AOI window had to be specified. This window could then be moved about until it was in the exact position over the black and white material, were it would remain throughout the test or until it had to be reinitialized. The window width was 300 pixels, which spanned over most of the length of the chute, and the height was 10 pixels (the reason for this height will be discussed later). The window did not extend to the end of the right hand side of the chute since uneven lighting caused a shadow on the right hand side, and so there was an inaccurate representation of the white materials grey level.

If we were to calculate the average value for the left and right window for every increment in R, it would be computationally expensive. To speed up the algorithm it was implemented as follows. All the pixel points in the AOI window were added together to form a total TOTAL. Two variables are initialized, the first is the Right Total RTOT = TOTAL and the second is the Left Total LTOT = 0. The pointer R then starts at column 1. The Column Total COLTOT is then calculated for the column R = 1. LTOT then becomes LTOT = LTOT + COLTOT and the RTOT = RTOT - COLTOT. LTOT and RTOT are then normalized to the number of elements in their respective windows. The difference is then taken between the two, thus giving the edge detectors output. R is incremented and the process is repeated, this is

done for all R. The DOA edge detector will give a positive output for a edge transition from black to white and a negative output for an edge transition from white to black. Since the edge transition of the ore is only black to white, we need only look for a maximum positive peak. The edge location is thus found by locating the position of R were the edge detector output is at a maximum. All arithmetic is done using integer numbers, therefore making the algorithm faster.

Figure 5.2 shows a photograph of the ore in the spiral with the edge ramp spanning over a relatively small number of pixels. The AOI is configured as described above. One difference however is that the left most edge of the AOI window is positioned over the left side of the spiral chute and not at the start of the black ore, the reason for this will be explained shortly. Figure 5.3 shows the edge profile of the ore, seen by the thick line which has been obtained by vertically averaging the pixel points in the AOI window. The DOA output is also seen and is smooth with only one positive peak. This can be seen to be sharper if plotted on a different scale. The edge position is thus the peak of the DOA output and is the vertical line situated at the transition point in the AOI window shown in figure 5.2. As a comparison the Sobel output is also shown. It too gives a peaked response at the edge but it is very noisy. The DOA response is seen to be far better than the response of the Sobel.

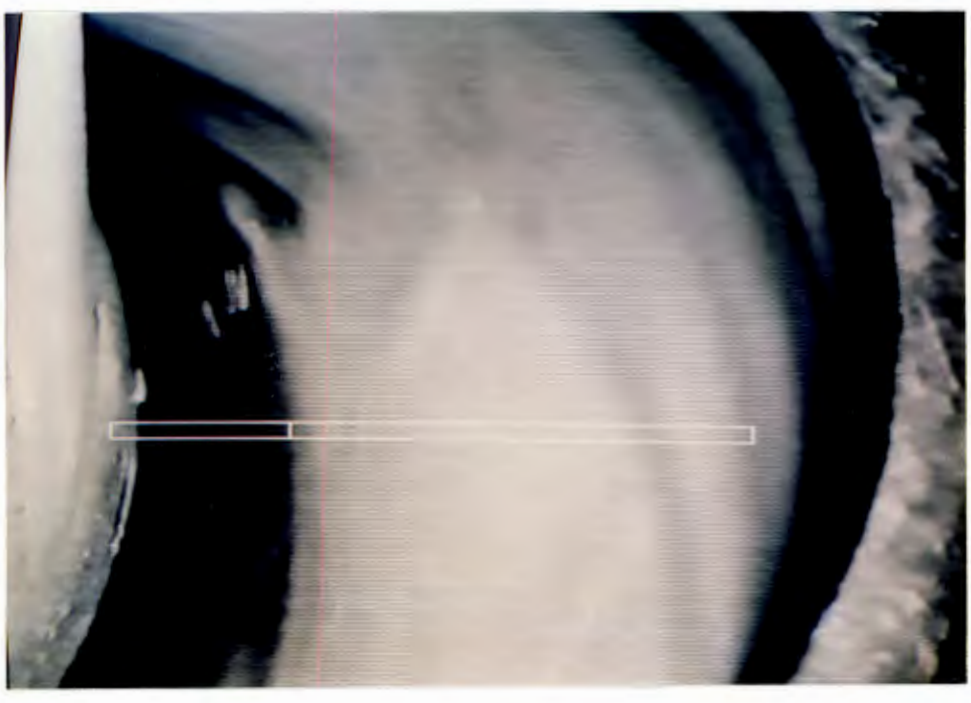


FIGURE 5.2 : Photograph showing the Area Of Interest window over a well defined edge.

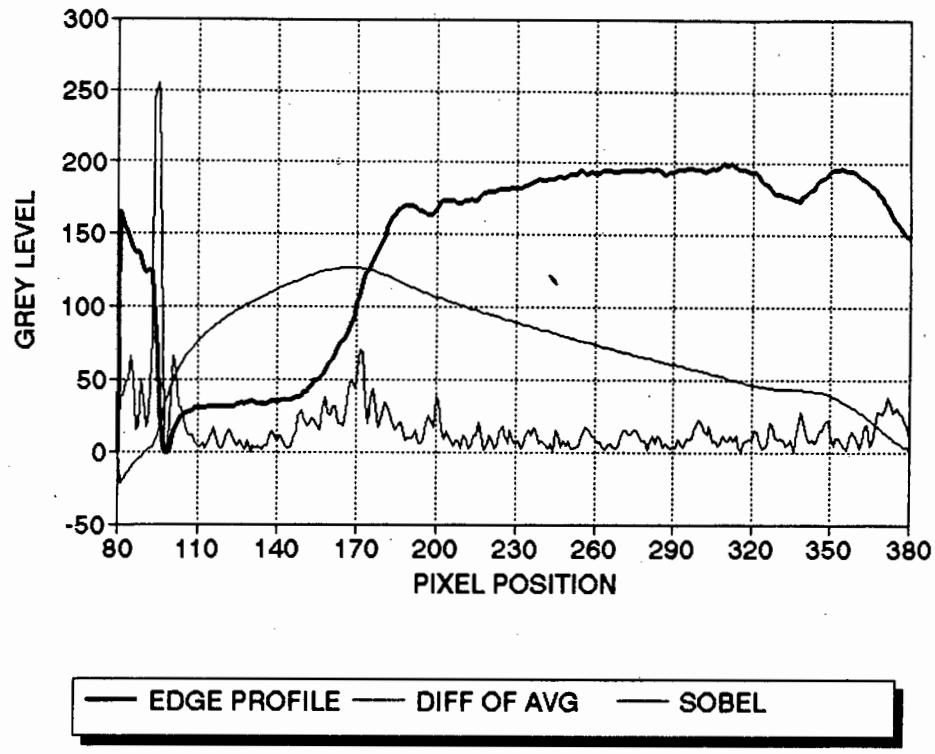


FIGURE 5.3 : Edge profile together with DOA and Sobel outputs for a well defined edge.

The reason for placing the AOI window over the side of the spiral chute is because at the start of the ore there is a prominent dip in grey level. This is seen at pixel position of about 100 (the cause of this dip is discussed in chapter 7 section 7.2.1). If the edge of the AOI window is positioned at the start of the black ore and consequently on this dip, the value of LAV at position R = 1, will equal the low grey level value of the dip. This will cause the difference between the RAV and the LAV at this point to be much larger than that of the maximum response of the DOA output at the edge transition, and so the edge position will be falsely located. To overcome this, the AOI window is placed over the left hand side of the chute which is highly reflective. Thus the transition from white to black, will give a strong negative output, which is ignored by the edge detector. The edge detector output will then climb to a maximum position at the edge transition point. By only including a small amount of the side of the chute, the value for the LAV will quickly converge to the average value of the black material and so the estimate of the left neighborhood will be negligible affected.

From figure 5.3 we can see that the Sobel operator did give a peak at the edge transition, so why use the DOA edge detector ? The DOA edge detector performs well compared to that of the Sobel, for ramps that span over many sample points. Figure 5.4 shows ore material that has been sprayed out over a large area. This is a common site in the spiral chute, since the black material, subjected to centrifugal force, is sprayed out forming finger like waves. Figure 5.5 shows the profile of the sprayed ore, it can be seen to have a long ramp starting from the low level plateau and extending over about 140 sample points. In the middle of the ramp, two humps can be seen, these are small fingers of heavy or black material. We see that the Sobel output is very noisy with no real peak at the edge. The DOA output however is seen to be very smooth from the gross averaging, and peaks at the optimum edge position. This edge position can be seen in figure 5.4 to be at the point were there is no more black material in the left hand window.

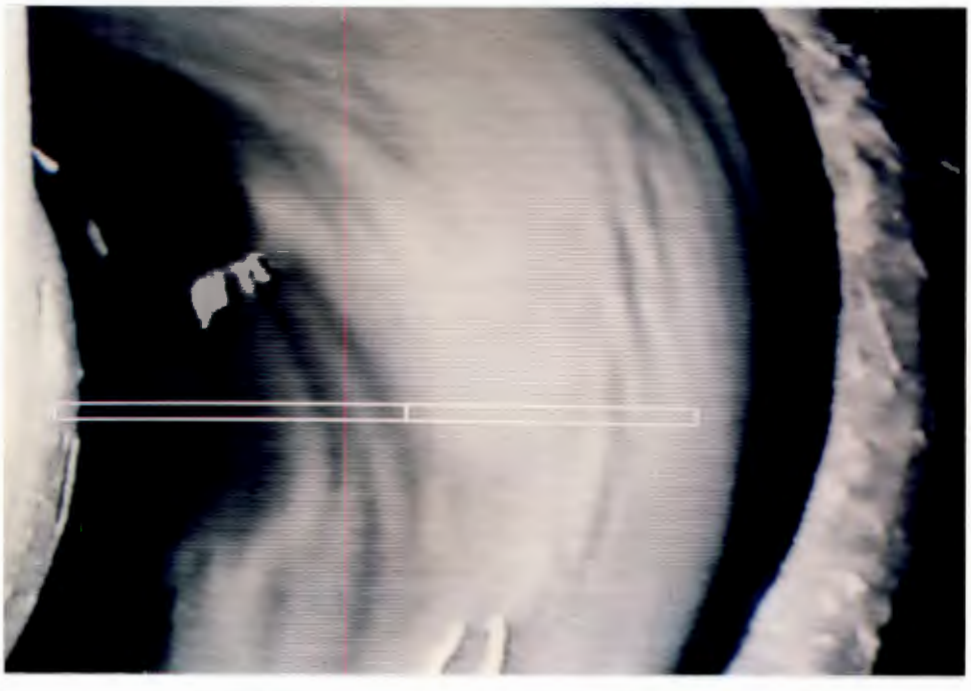


FIGURE 5.4 : Photograph showing the Area Of Interest window over an edge ramp spanning over many sample points.

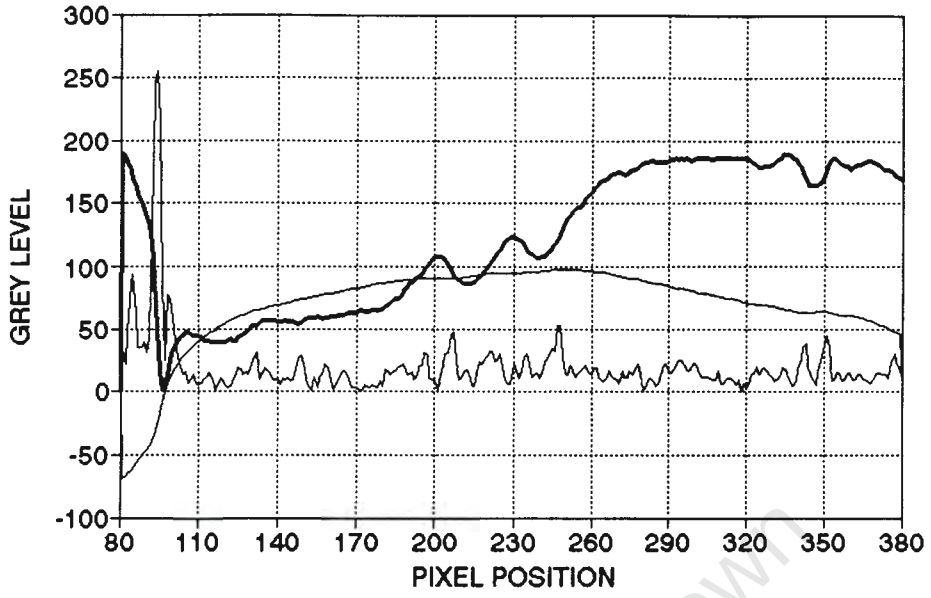


FIGURE 5.5 : Edge profile together with DOA and Sobel outputs for an edge ramp spanning over many sample points.

Because of the gross averaging of the DOA edge detector, any surface irregularities will have negligible effect on the detectors output. We have seen in the previous figures that the detector performs well even when the ore is sprayed out in finger like waves. Another common surface irregularity is highly reflective bubbles that occur on the surface of the wet ore[1]. Because the DOA detector averages over such a large area it is able to suppress the effect of the bubbles, to a certain extent.

Figure 5.6 shows a highly reflective bubble that is seen to be on the top of the black ore. A bubble present on the black material is more likely to cause an error than a bubble on the white material. This is because the black material has a low grey level value and the high value of the bubble will pull its average value up, thus making the difference between the two neighborhoods more difficult to detect. The white material and the bubble both have high grey level values, thus the bubble is less prominent on the white material and therefore it will have less effect.

We see that the Sobel output in figure 5.7 is very strong at the edges of the bubble, but has no real peak at the edge transition point. The DOA output also gives a peak at the bubble position, but the peak at the edge transition point is much stronger. We

[1] It has been mentioned in chapter 4 that the use of soft of diffused lighting can suppress glinting from the bubbles.

can see the further away the bubble is from the transition point the less likely there will be an error. If the bubble were to sit on the edge, the two peaks in the DOA output would be very close together and thus be indistinguishable from one another. This however would not cause an error, since once the system is running continuously, a filter with a time lag will ensure that the blade does not move to any extreme position, which may be caused from the edge being located on a large bubble. The filter will maintain the optimum average value of the edge position. Also a bubble will not always be present in the AOI window, so if in one sample the edge locks onto a bubble, there will be many subsequent samples that do not have the bubble in the AOI window.

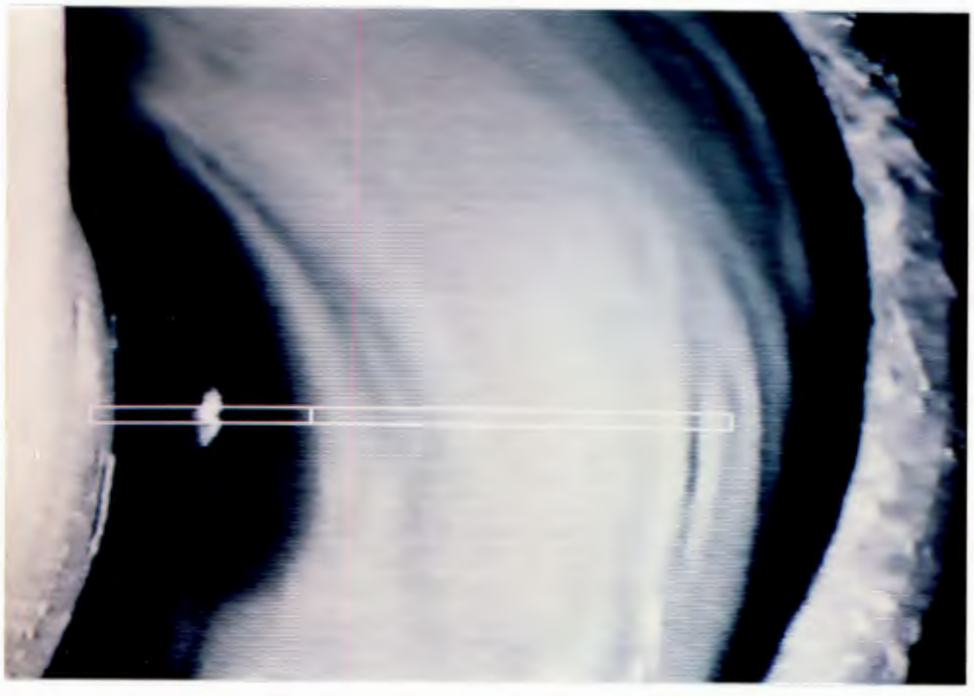


FIGURE 5.6 : Photograph showing the Area Of Interest window over an edge obstructed by a bubble.

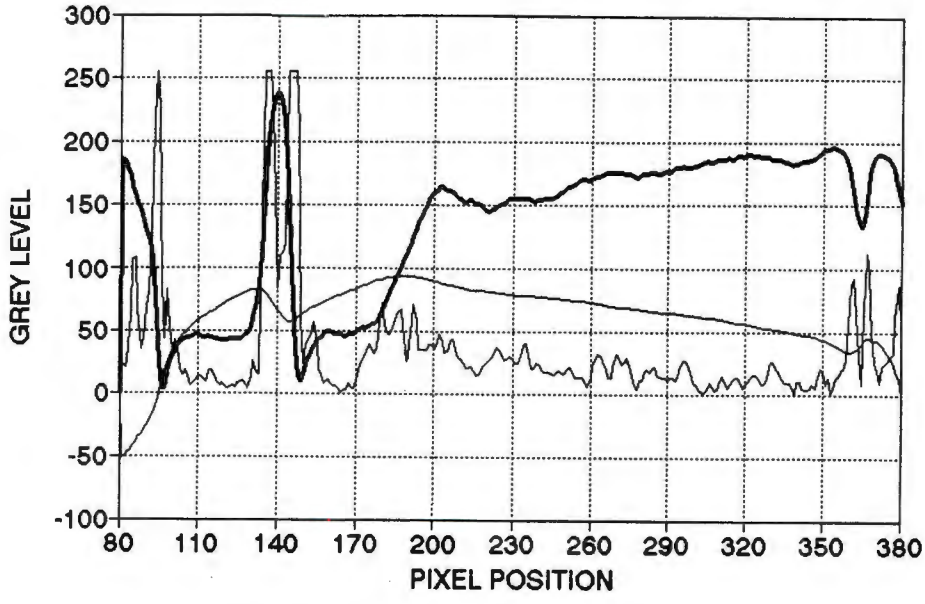


FIGURE 5.7 : Edge profile together with DOA and Sobel outputs for an edge obstructed by a bubble.

In chapter 2 we have seen that some spirals have 2 blades that split the material into 3 streams. The streams are concentrate, middlings, and tailings which is shown in figure 5.8. In figure 5.9 the profile can be seen to have 3 dominant plateaus. The DOA can be seen to have 2 peaks, each situated at the transition point between the plateaus. Also seen is the Sobel output which gives two peaks at the transition points, but no exact peaked point can be determined. The DOA has found the edge to be at the second transition point, because the concentrate and middlings are black and grey respectively and have been averaged together into one low value. By looking for the 2 dominant peaks, the DOA algorithm can be made to operate on edges with 2 transition points. This was not further investigated since the specification of the project was for only one edge to be found and one blade to be controlled.

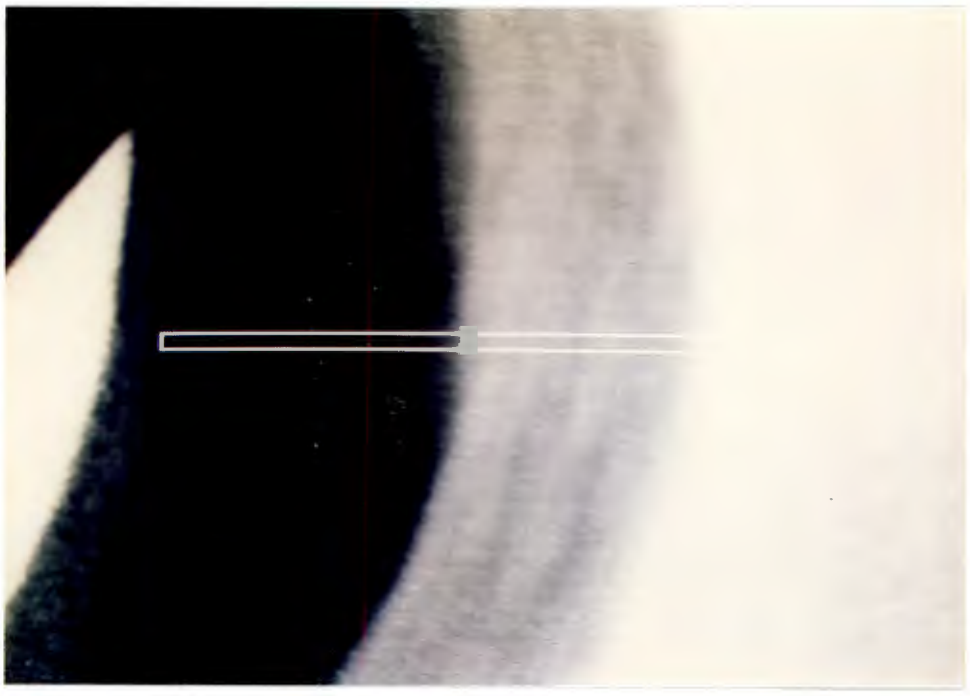


FIGURE 5.8 : Photograph showing the Area Of Interest window over an edge with 2 transition points.

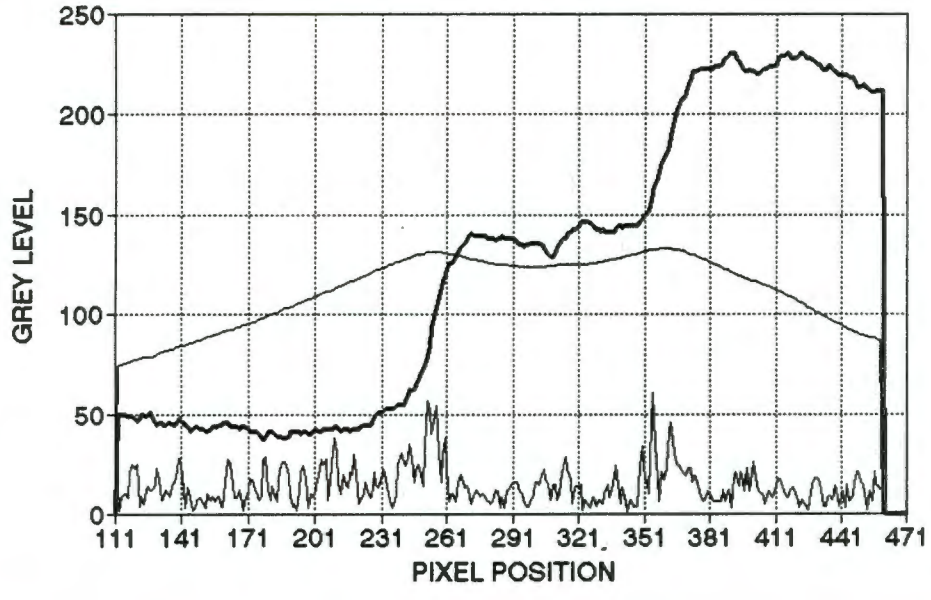


FIGURE 5.9 : Edge profile together with DOA and Sobel outputs for an edge with 2 transition points.

Because the system will operate in an industrial environment it will be subject to vibration, thus moving the AOI window. The AOI window has proved itself to be

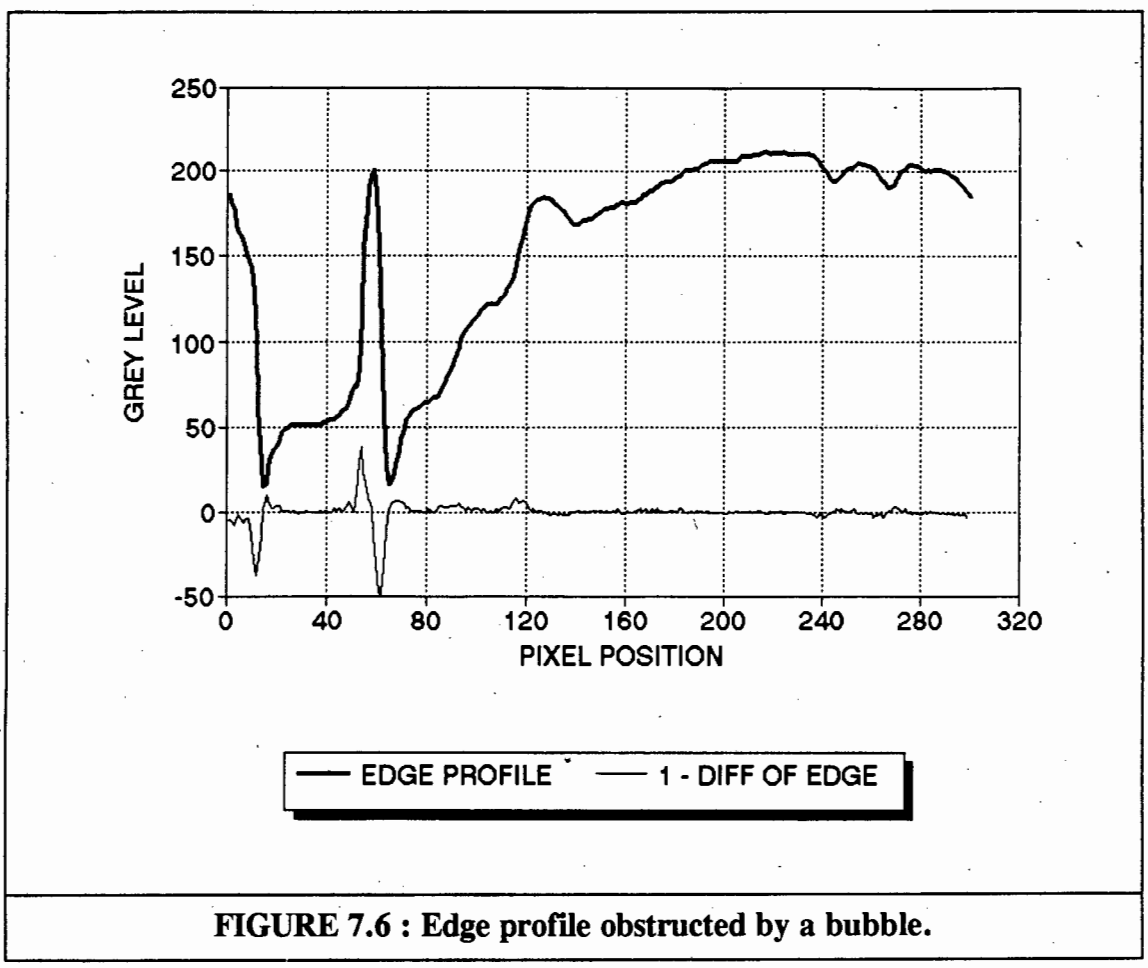


FIGURE 7.6 : Edge profile obstructed by a bubble.

Because the calculations of the losses rely on an accurate edge profile, the bubble presence will cause inaccurate results. We thus need to reject all the grey level profiles with bubbles present. In figure 7.6 we see that the slopes of the leading and trailing edges of the bubble are large compared to that of the slope of the transition ramp of the edge. We thus differentiate the edge profile to obtain an indication of the slope. This is done by a simple point by point subtraction of the present and the preceding point for the length of the edge profile. This is also shown in figure 7.6. The ore's edge transitions slope will rarely go over a value of +10 to -10 while that of the bubble ranges from +50 to -50. In order to determine if a bubble was present, a threshold level is set at 25. If the value of the differential, from the START position to the END position, exceeds this threshold, the edge profile is rejected for the calculation of the losses, and the next sample point or profile is examined. If the threshold is not exceeded the edge profile is used for the calculation of the losses.

7.5. LOSS ACCURACY

The edge position is updated every 0.2s in the calculations of the losses. There is thus errors being introduced because of sub-sampling of the edge position. This will now be discussed.

7.5.1. SAMPLING TIME

In the PAL system a frame is updated 25 times a second[1]. In the calculation of the losses there is a update time of every 0.2s. Thus there will be a sample point every (25*0.2) = 5th frame. Therefore we are sub-sampling and introducing an error. Ideally to have no error we need to sample not only every frame, but at every pixel point. To find a value for this error we compare the true edge location that has been sampled at every pixel point, to that of the sub-sampled edge location that has been sampled every 5th frame.

In order to find the edge location from pixel to pixel, the Difference Of Averages edge detector is used. The AOI window is again set to a width of 10 pixels, thus enabling averaging inside the window to produce a more accurate edge position. The sub-sampled edge location can thus be found by taking periodic points from the determined true edge curve. This remains consistent with the continuous operation of the algorithm, which also determines the sub-sampled edge location by using a AOI window of 10 pixels.

In order to reconstruct the true edge location we need to have a frame by frame analysis of the ore. This was achieved by using the JVC VPV HR-D700E video recorder which has a very stable pause facility. When paused, the current frame is digitized and displayed by the recorder, thus producing a perfectly still image[2]. As pause is pressed again, the next frame is digitized and displayed, thus enabling a frame by frame analysis.

It was seen for successive frames that the ore slurry moves down about 56 pixels. Therefore in order to reconstruct the true edge position, and have continuity from frame to frame, a window height of 56 pixels was chosen. The AOI window was then moved from the top of the 56 point window, to the bottom, calculating the edge location for every second pixel point or row. The edge location was seen to have no

[1] European Standard.

[2] Most video recorders when paused take the image directly from the magnetic tape which causes the image to be unstable.

sharp changes from pixel to pixel so in order to reduce calculations, every second pixel point was taken in the 56 point window. This was then repeated for successive frames thus building up a plot of the true edge location.

From the curve of the true edge location we can now generate the sub-sampled edge location. Each sample point on the sub-sampled curve will be held for a time of 0.2s until the sample point is updated, so the sub-sampled curve can be represented by a zero-order sample and hold. It can be seen that each frame is represented by (56/2) = 28 points. Therefore for an update of every 0.2s or 5th frame, would be after (28*5) = 140 points. This value then is held for the next 5 frames or 140 points. The filter output is then determined from the sub-sampled curve using the filter equation (6.13) as described in chapter 6.

In figure 7.7 we can see the determined true edge location, the sub-sampled edge location, as well as the filter location for an ore concentration of 2 Kg. What can also be seen from the curve of the true edge location, is the spraying of the ore in the form of waves that seem to be equally spaced at a distance of 160 sample points.

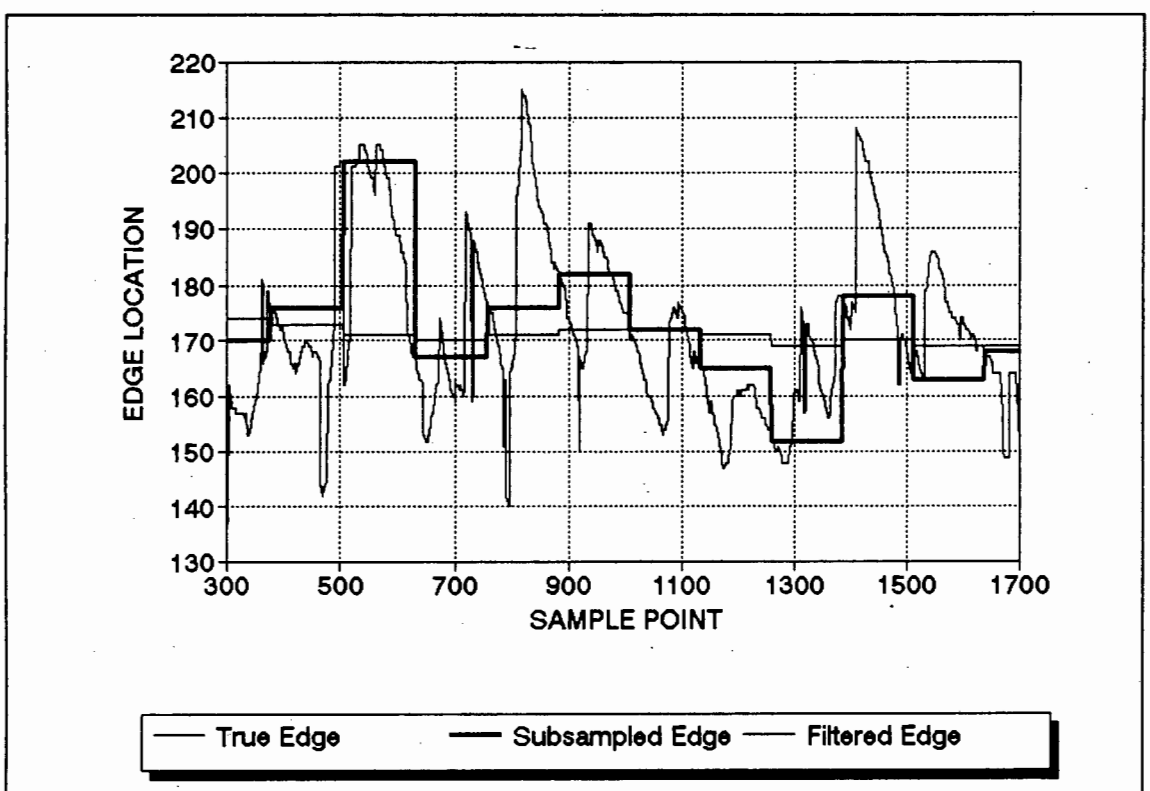


FIGURE 7.7 : Comparison of True edge versus Sub-Sampled edge for an ore concentration of 2 Kg.

Each sample point in the true edge location curve is the sample from every alternate line since we are sampling every second pixel point vertically downwards. The line frequency of the PAL colour television system is [22] :

$$f_{\text{line}} = 15.625 \text{ KHz.}$$

Therefore the time to write 1 line will be (1/15625) = 64μs, and thus for 2 lines will be 128μs. Therefore the separation for each sample point in figure 7.7 is 128μs.

In order to calculate the sub-sampling error we calculate the black loss and white contamination to the sub-sampled curve (Y), as well as to the true edge curve (X), from the filter output. Note : for simplicity of calculation we assume the grey level profile to be a step edge, so the black loss can be determined by using equation (7.6) and the white contamination by (7.7). If we take the difference of X and Y we will obtain the error due to sub-sampling, since X and Y are both calculated relative to the filter position. If we now reduce the filter lag, the loss values of X and Y, are reduced by the same amount. If the filter lag is reduced to zero, the filter curve will follow that of the sub-sampled curve and thus the losses of Y = 0. The error due to sub-sampling will then simply be the black loss percentage and white contamination percentage, between the sub-sampled curve and that of the true edge.

Because the sub-sampling error equally affects both the black loss and the white contamination, their errors will be the similar. In figure 7.8 we see the black loss and white contamination errors for the various concentration levels. It is seen that both errors are predominantly around 9% for the various concentrations, for a sampling time of 0.2s.

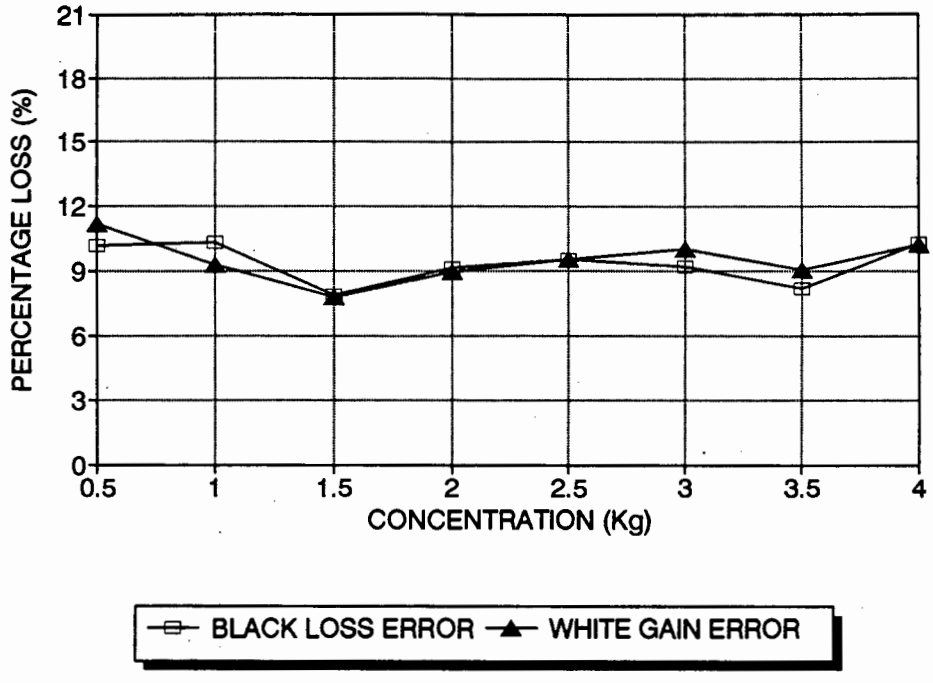


FIGURE 7.8 : Black Loss and White Contamination error for a sample time of 0.2s.

7.5.2. OPTIMUM SAMPLING TIME

Since we have the true edge position for every second pixel point we can determine its frequency components from the use of the periodogram as discussed in chapter 6. From the analysis of the power spectrum the highest frequency component can be located. This will set the sampling rate since, for an accurate reproduction of the true edge position, we must sample at twice this rate. This is known as the Nyquist sampling rate.

Figure 7.9a - 7.9h shows the power spectrum of the true edge position for the ore concentration levels from 0.5 Kg - 4 Kg in steps of 0.5 Kg. The periodogram was obtained by using a data set of 2048 sample points which spans over 73 frames. The data set was divided into 8 windows with 256 sample points in each, this as discussed in chapter 6, gives a better estimate of the power spectrum. From figure 7.9a - 7.9h it is seen that the power spectrum is very similar for each concentration level, with only the power level increasing as concentration increases. Because the power spectrum does not change shape, the period between the waves will remain the same for an increase in concentration level. However the spread of the waves is greater for each concentration level and so the power in the periodogram will increase.

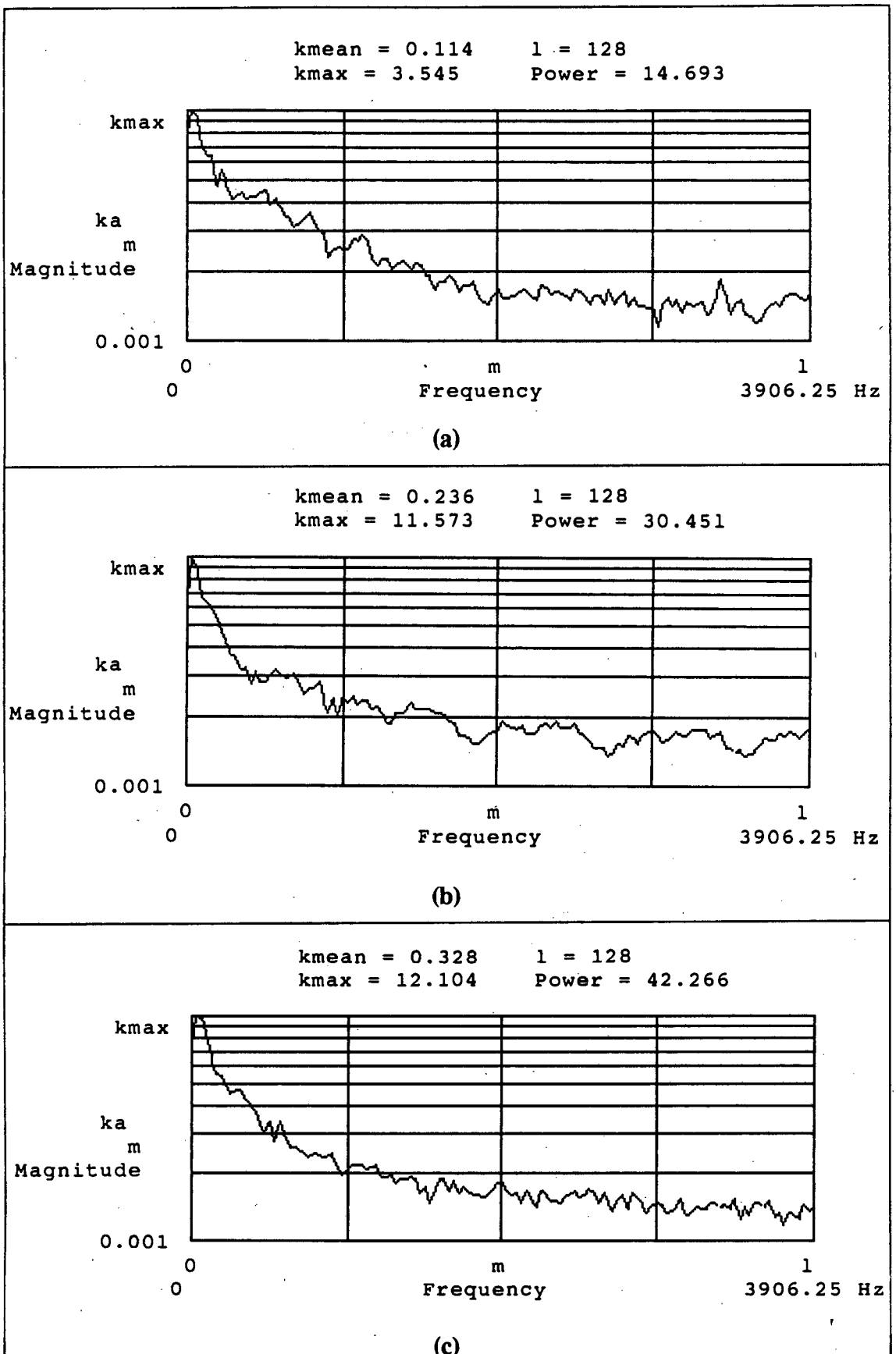
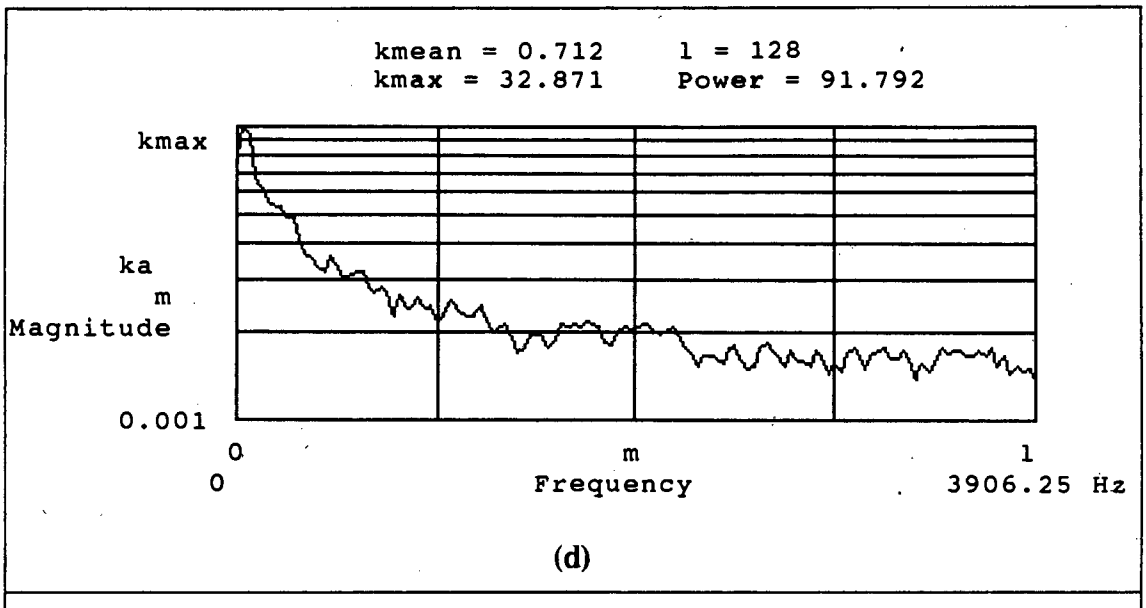
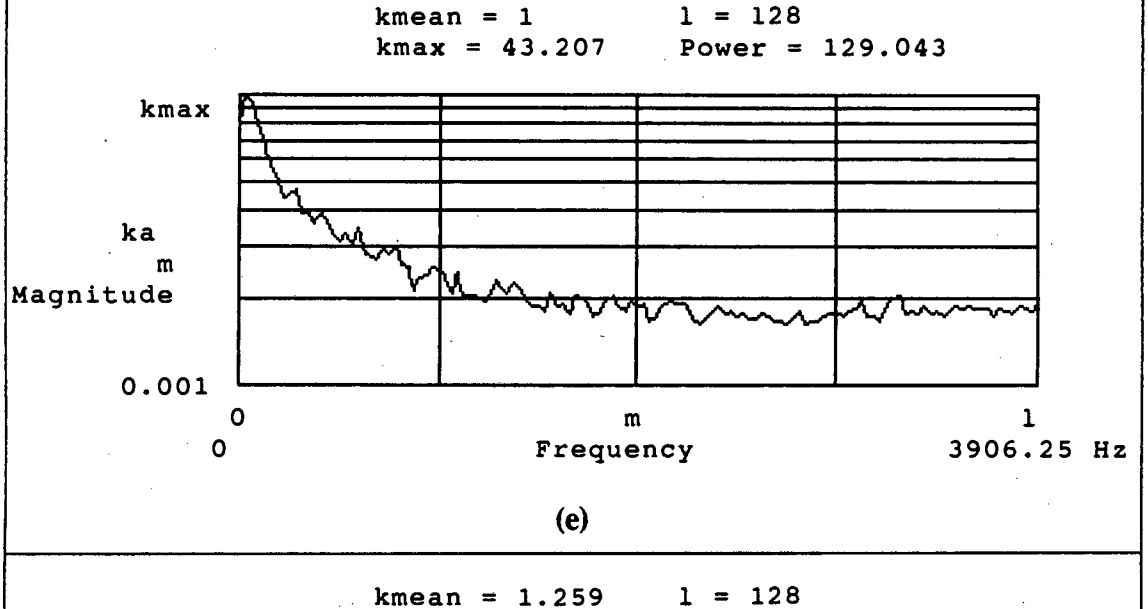


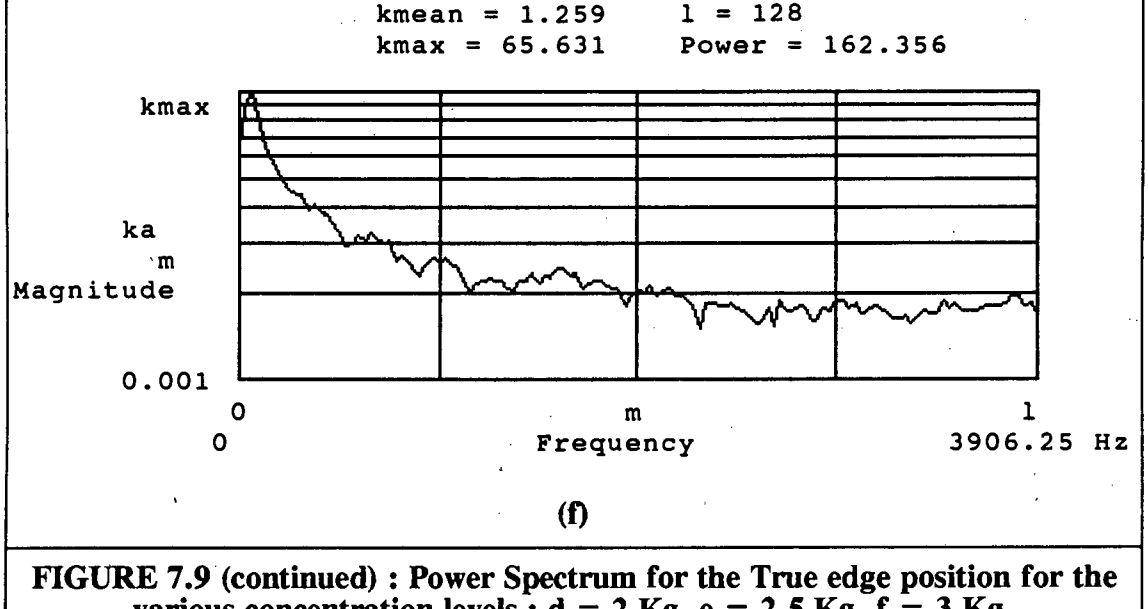
FIGURE 7.9 : Power spectrum for the True edge position for the various concentration levels : a = 0.5 Kg, b = 1 Kg, c = 1.5 Kg.



(d)



(e)



(f)

FIGURE 7.9 (continued) : Power Spectrum for the True edge position for the various concentration levels : d = 2 Kg, e = 2.5 Kg, f = 3 Kg.

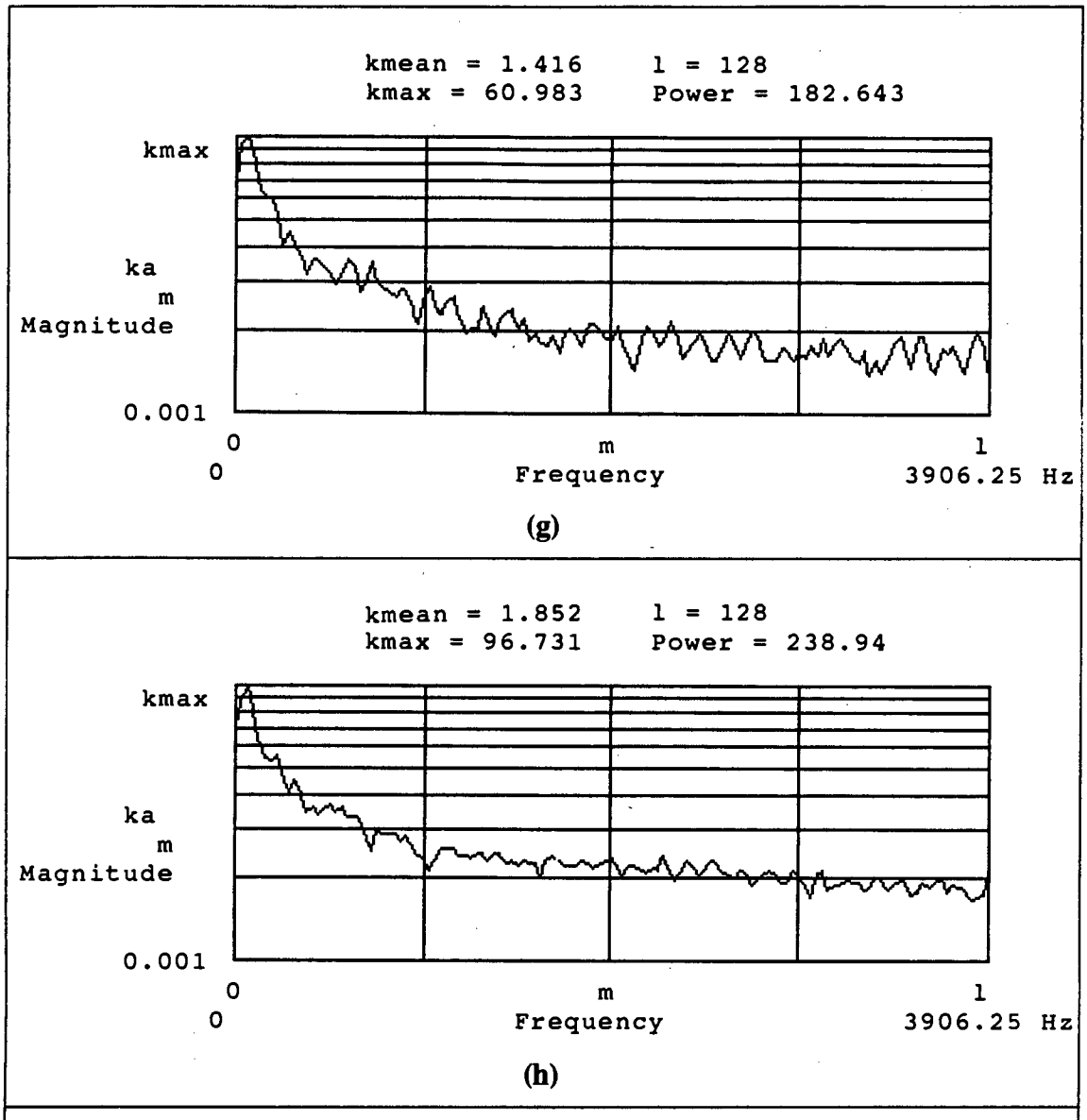


FIGURE 7.9 (continued) : Power Spectrum for the True edge position for the various concentration levels : g = 3.5 Kg, h = 4 Kg.

Because the sampling rate for every second point is 128μs, the sampling frequency is 7812.5 Hz and so the highest frequency seen on the periodogram is half of this, which is 3096.25 Hz. From the above figures the spectrum is seen to flatten out after 976.56 Hz, with no prominent frequency present and so can be regarded to be noise. Therefore the highest frequency component can be taken to be at 976.56 Hz, and the Nyquist frequency to be twice this value which is 1953.12 Hz. Ideally there should be a low pass antialiasing filter to suppress high frequencies, but we cannot low pass filter the ore stream flowing down the chute. In order to accurately reproduce the true edge position, the system must have a sampling time of 0.000512s or sample at every 8th pixel point. For the implementation of the system it would be futile to even try and get close to this sampling rate since the motor that controls the blade

cannot move this fast. We thus sample as fast as we can in order to provide the motor with a good estimate of the mean edge position. Because we are only sampling at 0.2s or once every 140 pixel points the values for the calculated black loss and white contamination will have an error of about 9% for the various concentration levels as seen in figure 7.8.

7.6. IMPLEMENTATION AND RESULTS

The algorithms mentioned in section 7.2, 7.3, and 7.4 were implemented in C software in a continuous repetitive mode. The following was obtained :

7.6.1. DIFFERING EDGE PROFILES

The movement of the slurry down the chute is a dynamic process and thus no two edge profiles are exactly alike. Figures 7.10 to 7.12 show differing edge profile with their determined START and END positions, as well as their UPPER and LOWER levels.

Firstly figure 7.10 shows the ore grey level profile for a concentration of 3.5 Kg. It can be seen that the profile closely represents a step edge. There is a definite low level plateau as well as an upper level plateau, although they are still uneven. The transition from black to white is seen to have a steep ramp. It can be seen that the right hand average value rises and flattens out from pixel position 140 to 235. Over this flat region the first differential of the edge profile can be seen to have an almost 0 value. This region of 0 slope will be were the right average value will be at its maximum value and so the white material's best estimate. The END location is taken to be the first zero crossing which gives good positioning for the UPPER level. The START position was located directly at the knee after the sharp dip located at pixel position 20. This knee represents the start of the black ore.

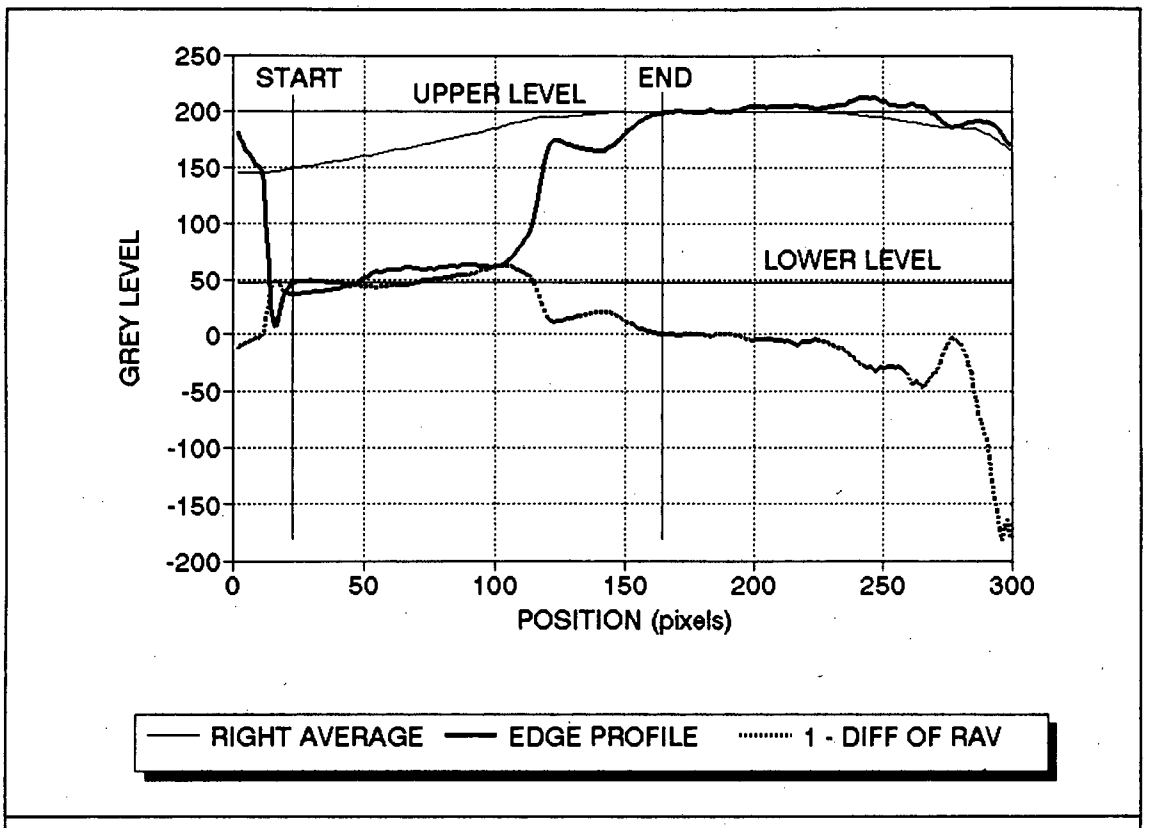


FIGURE 7.10 : Edge profile 1 with determined UPPER and LOWER levels.

Figure 7.11 shows the profile for an ore concentration of 3.5 Kg. Again the upper and lower plateaus can be identified although they are more uneven than that figure 7.10. The ramp is also seen to be not as steep as that of the previous graph. Spraying of the ore will produce wave like protrusion of the ore as seen in figure 7.7. If one of these waves is in the AOI window, then the black material will not evenly ramp up to the white material, but it will rather be black material followed by white and then again black. The presence of white material can be seen as a hump in the middle of the ramp in figure 7.11. The determined UPPER and LOWER levels and START and END positions are seen to be very similar to figure 7.10, despite the unevenness of the edge profile.

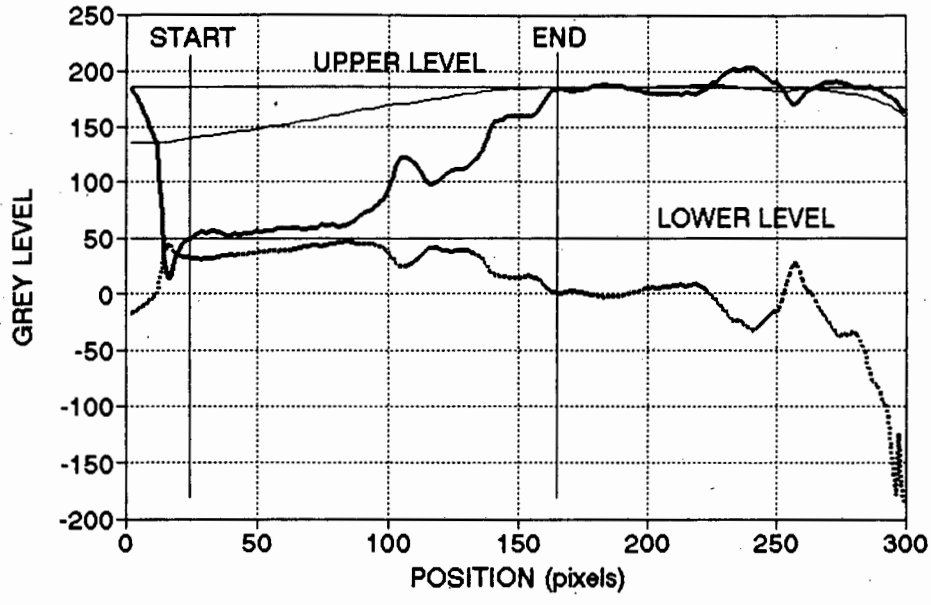


FIGURE 7.11 : Edge profile 2 with determined UPPER and LOWER levels.

Finally figure 7.12 is shown for a concentration of 1 Kg. This figure shows two dominant waves present in the AOI window, and can be seen by 2 humps present in the ramp of the ore profile. Each hump represents white material found between the waves of the black material. The right hand average value is seen to reach a maximum and then drop off. This can be seen as only one zero crossing in the curve of the first differential of the ore profile. The END location will be taken at the point of zero crossing, thus giving the UPPER level as shown figure 7.12. The START and LOWER level are once again placed at the knee after the sharp dip located at pixel position 20.

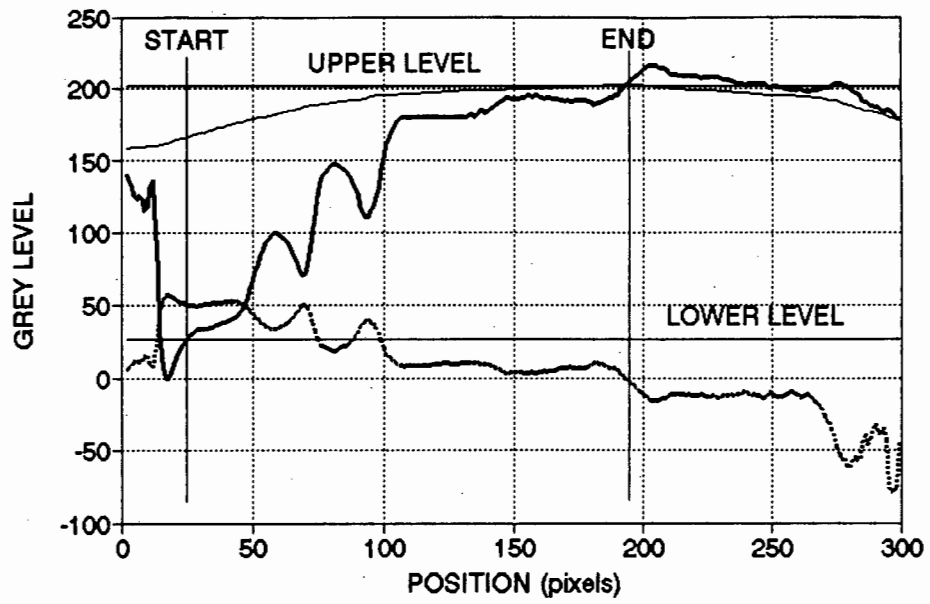


FIGURE 7.12 : Edge profile 3 with determined UPPER and LOWER levels.

It can be seen that the methods for determining the START and END positions, as well as their UPPER and LOWER levels as described in sections 7.2 and 7.3, are robust and give consistent results for the differing edge profiles. As seen no two ore profiles are alike, therefore the value of the spraying losses from sample to sample will be very different. In order to have an indication of the mean value of the spraying losses, 30 values of the determined spraying losses were averaged together and displayed as output. The values for the total losses will be stable since they are calculated over a window of 40 points.

7.6.2. LOSSES FOR DIFFERING CONCENTRATIONS

Once the algorithms for the determination of the spraying losses and total losses had been implemented in C software, a real time and on line measurement of these losses could be taken. Because the spaying losses and total losses calculations were determined over windows of 30 and 40 points respectively, their values were still not completely stable. Therefore in order to get an accurate mean value of the losses for the various ore concentration levels, the loss and contamination values were further averaged over a time window of 2 minutes. The percentages of the black loss and white contamination for the spaying of the ore, as well as the total black loss and total white contamination, for the various ore concentration levels is shown in figure 7.13.

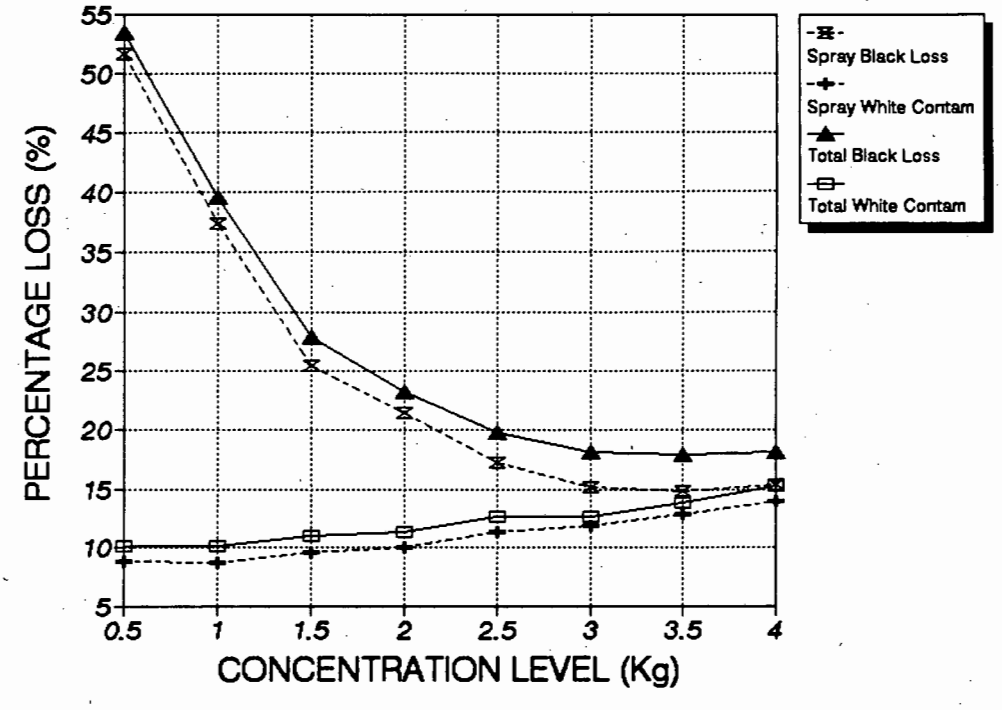


FIGURE 7.13 : Total and Spraying Losses for differing ore concentration levels.

We can see that the black loss at low concentration will be high. This is due to the fact that at low concentrations, the width[1] of the black material gained is small compared to that of the spread caused by the spraying. At higher concentrations, the black width increases[2] and becomes large compared to that of the spraying spread. The white contamination remains initially small because it is not dependant on the width of the black material but rather the ratio of black versus white material gained. It does however increase as the concentration increases. This is due to an increase in spraying spread of the ore as the concentration increases.

7.7. MULTIPLEXING MULTIPLE CAMERA INPUTS

Spiral ore concentrators are not usually operated by themselves but are "ganged" together in groups. The Machine Vision System must ultimately be able to have more than one camera input and so detect the edge position in many spirals. The scope of the project only called for the feasibility of multiplexing multiple camera inputs. The governing factor on determining how many cameras can be multiplexed together, is the sampling time. If the sampling time was very fast we could

[1] Distance from START to EDGE.

[2] The mean value and standard deviation of the edge position, for differing concentration is shown in figure 5.11 of chapter 5.

determine 1 edge position or sample point for each camera in turn, cycling through all the cameras. We have seen in figures 5.10a - 5.10h that the edge position can change radically from sample point to sample point, thus by only taking 1 sample of the edge position from each camera will not give the optimum mean edge position. Therefore to determine the optimum mean position we must average together many edge position points, sampled from 1 camera, then move onto the next camera. We thus need to know how many sample points from each camera will give the best estimate of the mean edge position, and after how long must we cycle back to that same camera, to update the edge position.

The cycle time from camera to camera can be obtained from the error in the readings of the black loss and white contamination. With a fast sampling time these errors would be small, but as the sampling time gets slower, causing sub-sampling, these errors would increase. The errors are calculated the same way as discussed in section 7.5.1, but by varying the sample time or sample point from 0 to say every 500 pixel points, we can see the trend of the black loss error and the white contamination error as a function of sample time. Figure 7.14 shows the black loss error and the white contamination error as a function of sampling time or sample point, for the ore concentration of 2.5 Kg. Note : every sample point in figure 7.14 represents every second pixel point in the curve of the true edge location.

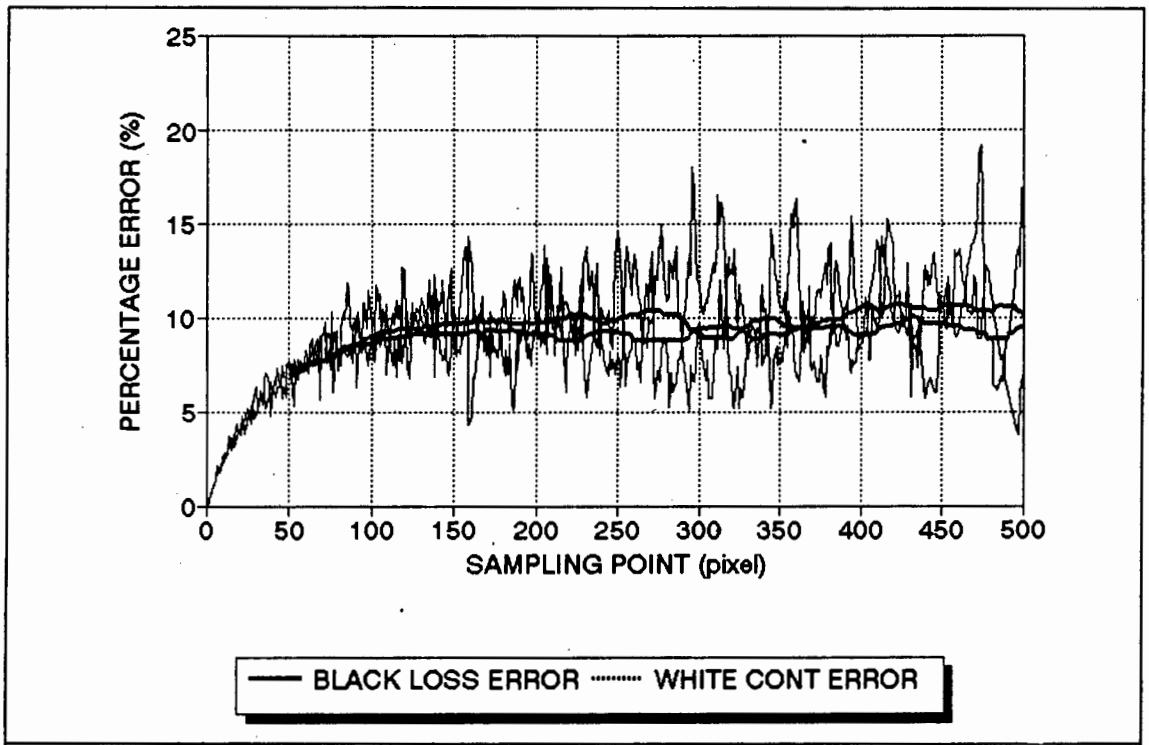


FIGURE 7.14 : Black Loss error and White Contamination error as a function of sample point.

We can see that as the sampling time is slowed down the percentage errors in the reading of black loss error and white contamination error is increased. These error readings become very noisy as the sampling interval is increased. This is because the sample point interval that is taken is much larger than the width of the waves of the sprayed ore. Therefore one sample point may lie on the crest of a wave, then for the next sample point, much greater than the length of the wave, may lie on the trough of the next wave. These noisy percentage error readings are smoothed by a median filter with a window size of 50. It is seen that both errors have almost the same value and both converge to a common percentage error of 10%. This common percentage error of 10% will occur when the edge position is at the optimum mean edge position. These maximum percentage errors were seen to range from 14% for an ore concentration of 0.5 Kg, to 10% for an ore concentration of 4 Kg. We can thus see that no matter how slow the sampling rate is, the error in the readings will always be from about 10% to 14%. Note : for this to be true the determined edge position over time must be a stationary[1] process, also the determined edge position must be at the optimum mean value.

If the edge position was stationary, an infinite number of cameras could be multiplexed together since the errors in the readings would remain between 10% - 14% as long as the edge position was at an optimum mean value. We however know from figure 5.11 that mean and standard deviation of the edge position changes with a step increases in ore concentration. From chapter 6 section 6.7.1 the time taken for the ore to reach a steady state level from a step change in concentration was 44.32s. Therefore in order to maintain the optimum mean edge position, each camera must be sampled in less than 44s.

Since the variance of the edge position is high, the optimum edge position for each camera can be obtained by averaging a number of sample points together. This will reduce the variance and so reduce the error in the estimate of the optimum mean value. Ideally we would want the error of the mean estimate value to be 0, which can be achieved by a large averaging window and hence a very small variance reduction factor. By plotting the VRF for a equally weighted averaging window given by equation (6.16), as a function of window size, the VRF can be seen in figure 7.15 to exponentially decrease, with the biggest reduction in the VRF between the values of 0 and 45, wereafter it slowly approaches 0 as the window size approaches infinity.

---

[1] The mean value and standard deviation of the edge position must not change with time.

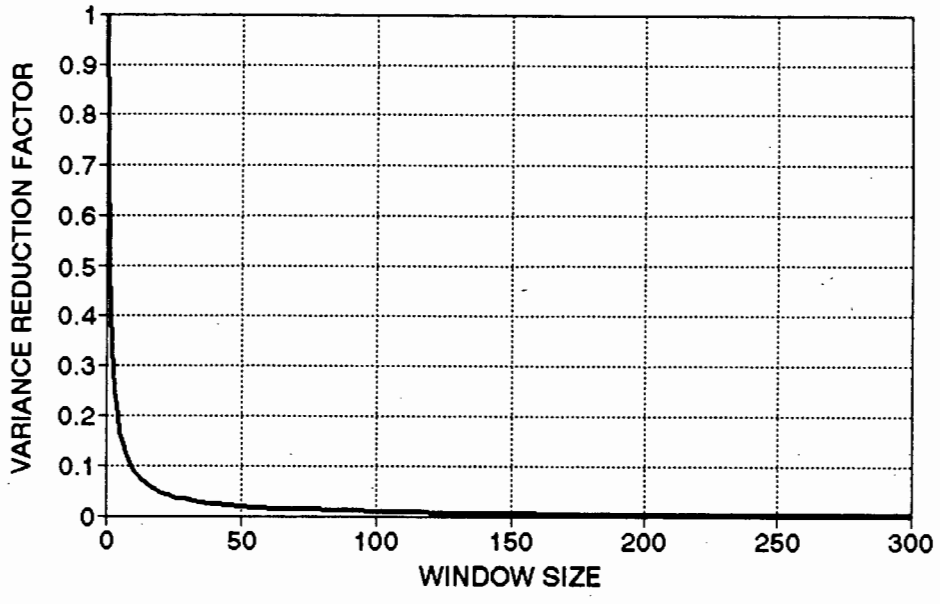


FIGURE 7.15 : Variance Reduction Factor for an equally weighted averaging window as a function of window size.

The number of sample points averaged together, to find the optimum mean edge position is thus taken to be 45. The reason for this is that a window size of 45 gives a small VRF of 0.0244 for a relatively small averaging window, thus minimizing the error of the mean edge value estimate. A window size greater than 45 will minimize this error even more but would require larger window sizes for only a small decrease of the VRF. More time would then have to be spent on each camera therefore limiting the number of cameras that can be multiplexed together.

With 45 sample points taken for each camera to find the optimum edge position and each edge position being updated every 0.16s, the time taken on each camera needs to be (45*0.16) = 7.2s. Because each camera must be cycled in less than 44s, the number of cameras that can be multiplexed together, will be (44/7.2) = 6. This value can be increased by reducing the sampling time. For example, the system was used on a 386 computer which reduced the time for an edge update from 0.16s to 0.11s, therefore this would increase the amount of cameras that can be multiplexed together, to be 8.

The objectives of the project have been accomplished. Conclusions and recommendations of the project will be discussed in the next, final chapter.

8. CONCLUSIONS AND RECOMMENDATIONS

8.1. CONCLUSIONS

The aim of the project was to automate the action of the operator, who positions the blade separator in the spiral ore concentrator. This was achieved through the implementation of a Machine Vision System. The proposed hardware of the system consisting of a CCD camera, a black and white frame grabber and a computer, was successfully implemented and tested in the laboratory.

The proposed and implemented algorithm that was used to find the outer edge of the concentrated material was based on finding the difference of averages between two global neighborhoods within a specified Area of Interest window. This algorithm was found to give consistent results and be robust towards any surface irregularities. This algorithm operating in a continuous repetitive mode gave a fast edge update time of 0.16s which was considered adequate, since the motor that will be used to control the blade separator will only have a slow response time. In order for an accurate reproduction of the edge curve, the Nyquist sampling time should be every 0.000512s or every 8th pixel point. This Nyquist criteria need not be achieved since the system only requires the optimum mean edge position and not a reproduction of the edge curve.

The algorithm operating in a continuous repetitive mode gave rapid fluctuations in the determined edge position. Due to the slow response time of the control motor, the rapid fluctuations needed to be smoothed by a filter. Based on periodogram analysis of the power spectrum it was found that smoothing can be attained through a simple averaging filter. The proposed and successfully implemented filter was a median filter followed by a fading-memory polynomial filter. The median filter excludes all high and low spikes that could corrupt the fading-memory polynomial filter. The fading-memory polynomial filter reduces the time lag of the filter while maintaining a sufficient amount of smoothing. These filters provide a smooth signal of the optimum mean edge position for the blade separator, with only a small delay to step changes.

An on line efficiency of the system can be obtained through the determination of losses, which comprise of, the amount of black material loss, as well as the amount of white material contamination. Two types of losses were found to contribute to the

Total losses of the spiral. They were, Spraying losses caused from the extent of the ore spraying which is dependent on ore concentration, and Filter losses caused from the time lag introduced from the filter. These losses were found to vary with concentration. Due to the sub-sampling of the system, the error in the calculation for the losses was found to be about 9% for the various concentration levels. From an examination of the error as a function of sampling time it was found that the system could multiplex 6 camera inputs together, thus reducing cost in a large scale implementation.

The objectives of the project were attained. The hardware needed for the successful operation of the Machine Vision System should consist of the following elements : A black and white CCD camera or colour CCD camera. If a colour camera is used the colour information must be removed by the use of an RGB decoder. A frame grabber such as the MATROX PIP-1024B video digitizer board or a cheaper version, the MATROX PIP-512B video digitizer board. A computer such as the 386-SX or a faster version, such as a full 386. This hardware together with the algorithms described in the thesis, will accurately and quickly determine the edge position between the two types of ore. This system will also be able to provide a control motor with the optimum mean edge position, as well as an on line efficiency of the separation process.

8.2. RECOMMENDATIONS

The system should be optimized for speed. This can be done by writing the most commonly used algorithms in assembler language, as well as using a faster computer such as a 386 or higher. The result of a faster system will be a faster estimate of the mean edge position, as well as a decreased error in the calculation of the losses. Also more camera inputs can be multiplexed due to the decreased sampling time.

A line scan camera can be used as input since only 10 pixel rows are being used out of the whole frame. The line scan camera could be used to only digitize 10 rows instead of the whole frame, thus minimizing redundancy and increasing the speed of the system. Since the proposed system called for hardware that was cheap and easily implemented, a line scan camera was not used since its cost and complexity is high. In the future, the cost and complexity may be reduced, enabling it to be used as an input source to the system.

9. REFERENCES

[1] Baase S, COMPUTER ALGORITHMS INTRODUCTION TO DESIGN AND ANALYSIS, Addison-Wesley Publishing Company, Second Edition, 1988.

[2] Boyle R.D, Thomas R.C, COMPUTER VISION A First Course, Blackwell Scientific Publication, 1988.

[3] Brink A.D, A SELECTION AND EVALUATION OF GREY LEVEL THRESHOLDS APPLIED TO DIGITAL IMAGES, Msc Thesis, Rhodes University, 1987.

[4] Burt R.O, SPIRAL CONCENTRATION : CURRENT TRENDS IN DESIGN AND OPERATION, Mineral Processing And Extractive Metallurgy, Institute Of Mining And Metallurgy, London, 1984, pp 117 - 127.

[5] Canny J, A COMPUTATIONAL APPROACH TO EDGE DETECTION, IEEE Transactions On Pattern Analysis And Machine Intelligence, Vol PAMI-8, No 6, Nov 1986, pp 679 - 698.

[6] Christiansen P.L, ON THE EVALUATION OF EDGE DETECTORS AND RELATED EVALUATION TECHNIQUES, Msc Thesis, Rand Afrikaans University, 1990.

[7] Davies E.R, VISUAL INSPECTION, AUTOMATIC (ROBOTICS), Encyclopedia of Physical Science and Technology, vol 14, 1987.

[8] Davis L.S, A SURVEY OF EDGE DETECTION TECHNIQUES, Computer Vision, Graphics, And Image Processing, Vol 4, 1975, pp 248 - 270.

[9] Gleeson G.W, WHY THE HUMPHREYS SPIRAL WORKS, Engineering and Mining Journal, Vol 146, March 1945, pp 85 - 86.

[10] Gonzalez R.C, Wintz P, DIGITAL IMAGE PROCESSING, Second Edition, Addison-Wesley Publishing Company, 1987.

[11] Goodman R.H, Brown C.A, Ritchie I.C, ADVANCED GRAVITY CONCENTRATORS FOR IMPROVING METALLURGICAL PERFORMANCE, Minerals and Metallurgical processing, May 1985, pp 79 - 86.

[12] Janesick J, Elliot T, Dingizian A, Bredthauer R, Chandler C, Westphal J, Gunn J, NEW ADVANCEMENTS IN CCD TECHNOLOGY - SUB-ELECTRON NOISE & 4096*4096 PIXEL CCD, SPIE Proceedings, Vol 1242, Symposium on Electronic Imaging, Santa Clara, CA, Feb 1990.

[13] Janse Van Rensburg C, NON-INTRUSIVE MEASURING ON A TELEVISION PICTURE USING SPECTRUM ANALYSIS, MSc thesis, University of Cape Town, 1990.

[14] Morrison N, INTRODUCTION TO SEQUENTIAL SMOOTHING AND PREDICTION, McGraw-Hill Book Company, 1969.

[15] Mothersole L.M, PAL COLOR TV...THE PAL SYSTEM AND MULLARD CIRCUITS DESCRIBED, Mullard LTD, London, 1967.

[16] Nalwa V.S, Binford T.O, ON DETECTING EDGES, IEEE Transaction On Pattern Analysis And Machine Intelligence, Vol PAMI-8, No 6, November 1986, pp 699 - 714.

[17] Oppenheim A.V, Schafer R.W, DISCRETE-TIME SIGNAL PROCESSING, Prentice Hall, 1989.

[18] Peebles P.Z, DIGITAL COMMUNICATION SYSTEMS, Prentice-Hall International Editions, 1987.

[19] Peli T, Malah D, A STUDY OF EDGE DETECTION ALGORITHMS, Computer Graphics and Image Processing, Vol 20, 1982, pp 1 - 21.

[20] PIP VIDEO DIGITIZER BOARD for the IBM PC, XT, and AT HARDWARE MANUAL 289-MH-00, Rev.4, March 22, 1989.

[21] Pratt W.K, DIGITAL IMAGE PROCESSING, John Wiley & Sons, 1978.

[22] Reed C.R.G, PRINCIPLES OF COLOUR TELEVISION SYSTEMS, first edition, Sir Isaac Pitman and Sons Ltd, 1969.

[23] Rosenfeld A, Kak A.C, DIGITAL PICTURE PROCESSING, second edition, Academic Press Inc, Volume 2, 1982.

[24] Rosenfeld A, PERSONAL COMMUNICATION, University Of Cape Town, 1990.

[25] Rosenfeld A, Thurston M, EDGE AND CURVE DETECTION FOR VISUAL SCENE ANALYSIS, IEEE Transactions On Computers, Vol C-20, No 5, May 1971, pp 562 - 569.

[26] Schachter B.J, Rosenfeld A, SOME NEW METHODS OF DETECTING STEP EDGES IN DIGITAL PICTURES, Communications of the ACM (USA), Vol 21, No 2, Feb 1978, pp 172 - 176.

[27] Simon J.C, Haralick R.M (eds), DIGITAL IMAGE PROCESSING Proceeding of the NATO Advanced Study Institute held at Bonas France June 23 - July 4 1980, Reidel Publishing Company, 1981, pp 155 - 176.

[28] Sims H.V, PRINCIPLES OF PAL COLOUR TELEVISION and related systems, Iliffe Books London, 1969.

[29] Stremler F.G, INTRODUCTION TO COMMUNICATION SYSTEMS, Second Edition, Addison-Wesley Publishing Company, 1982.

[30] Tsai W.H, MOMENT-PRESERVING THRESHOLDING : A NEW APPROACH, Computer Vision, Graphics, And Image Processing, Vol 29, 1985, pp 377 - 393.

[31] Weimer P.K, TELEVISION IMAGE SENSORS, Encyclopedia of Physical Science and Technology, vol 13, 1987.

[32] Young T.Y, Fu K, HANDBOOK OF PATTERN RECOGNITION AND IMAGE PROCESSING, Academic Press Inc, 1986.

LIST OF APPENDICES

APPENDIX A

Appendix A contains the MCAD listing for the periodogram analysis of the power spectrum. The periodogram is implemented according to equations (6.9) and (6.10). The input data set of 1024 points is split into 8 windows of 128 points. The mean value in each window is then removed so that the DC component does not dominate the higher frequency components. Each window is multiplied by the Hamming window function and the Fourier transform of each is taken. The magnitude of the resultant Fourier transform of each window is summed and scaled to give the resultant averaged periodogram.

$d := 7$

$L := 2^d$ Window Length $n := 0 \ldots L - 1$ input vector range

$$wb(n) := .54 - .46 \cdot \cos\left[2 \cdot \pi \cdot \frac{n}{L - 1}\right]$$ Hamming Window

$xi := 0 \ldots 1023$

$xx_{xi} := READ[e2\ prn]$ Input sequence

$x0_n := xx_n$ $x1_n := xx_{n+L}$ $x2_n := xx_{n+2 \cdot L}$ $x3_n := xx_{n+3 \cdot L}$

$x4_n := xx_{n+4 \cdot L}$ $x5_n := xx_{n+5 \cdot L}$ $x6_n := xx_{n+6 \cdot L}$ $x7_n := xx_{n+7 \cdot L}$

$tm := mean(x0)$ $x0_n := x0_n - tm$ $tm := mean(x1)$ $x1_n := x1_n - tm$

$tm := mean(x2)$ $x2_n := x2_n - tm$ $tm := mean(x3)$ $x3_n := x3_n - tm$

$tm := mean(x4)$ $x4_n := x4_n - tm$ $tm := mean(x5)$ $x5_n := x5_n - tm$

$tm := mean(x6)$ $x6_n := x6_n - tm$ $tm := mean(x7)$ $x7_n := x7_n - tm$

window sequence

$v0_n := wb(n) \cdot x0_n$ $v1_n := wb(n) \cdot x1_n$ $v2_n := wb(n) \cdot x2_n$

$v3_n := wb(n) \cdot x3_n$ $v4_n := wb(n) \cdot x4_n$ $v5_n := wb(n) \cdot x5_n$

$v6_n := wb(n) \cdot x6_n$ $v7_n := wb(n) \cdot x7_n$

$$U := \frac{1}{L} \cdot \sum_n wb(n)^2$$ $U =$ $Df := U \cdot 8 \cdot L$ $Df =$

$k0 := fft(v0)$ $k1 := fft(v1)$ $k2 := fft(v2)$ $k3 := fft(v3)$

$k4 := fft(v4)$ $k5 := fft(v5)$ $k6 := fft(v6)$ $k7 := fft(v7)$

$$l := \frac{L}{2}$$ $m := 0 \ldots l$

$k0_m := \left[\left|k0_m\right|\right]^2$ $k1_m := \left[\left|k1_m\right|\right]^2$ $k2_m := \left[\left|k2_m\right|\right]^2$ $k3_m := \left[\left|k3_m\right|\right]^2$

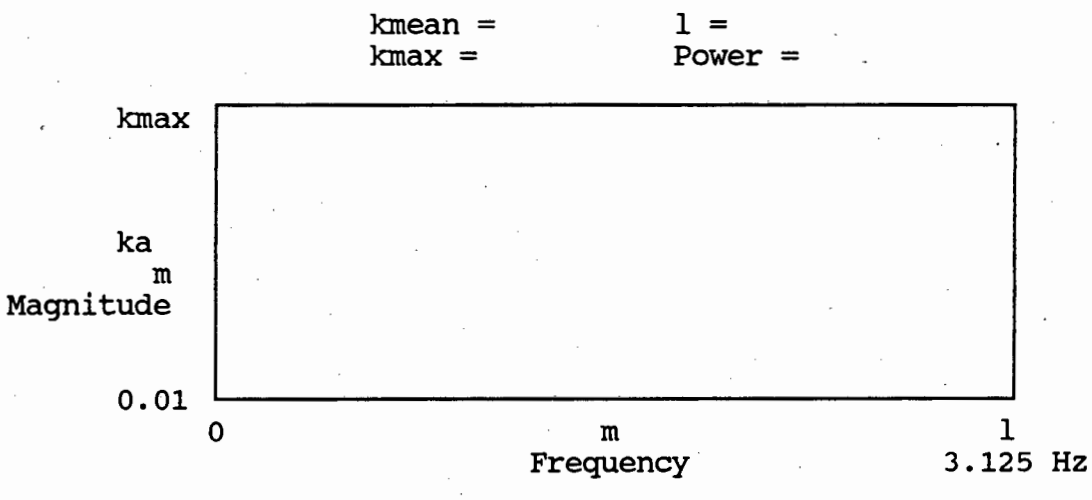
$k4_m := \left[\left|k4_m\right|\right]^2 \qquad k5_m := \left[\left|k5_m\right|\right]^2 \qquad k6_m := \left[\left|k6_m\right|\right]^2 \qquad k7_m := \left[\left|k7_m\right|\right]^2$

$$ka_m := \frac{1}{Df}\left[k0_m + k1_m + k2_m + k3_m + k4_m + k5_m + k6_m + k7_m\right]$$

$$Power := \sum_m ka_m$$

kmax := max(ka) kmean := mean(ka)

Averaged periodogram- power density spectrum



APPENDIX B

Users Guide to

Program : EDGE9GC.C

Description

This program controls the operation of the Machine Vision System that is to operate on the Spiral Ore Concentrator. It will find and display the determined edge position, as well as the filtered edge position. Also determined are the Spraying losses, the Filter losses and the Total losses. The data of the edge position and the losses can also be saved to a text file.

Hardware Required

IBM computer, preferably an AT or higher with a VGA monitor. A MATROX PIP 1024B or 512B Video Digitizer Board.

Operation

To start program type EDGE9GC [Enter]. The first stage of the program is the initialization of the Area Of Interest (AOI) window.

Initialization

To start the user must enter in the coordinates of the AOI window. This is done by entering in the coordinates of the top left hand corner, denoted by X1, Y1 and the bottom right hand corner, denoted by X2, Y2. Due to the transfer capability of the frame grabber, the AOI window cannot have a height greater than 100 pixel rows. The program will prompt the user to enter in the values for X1, Y1, X2 and Y2 respectively. The default values for X1, Y1, X2 and Y2 are 80, 315, 380 and 325 respectively. These values can be enter by simply pressing [Enter] at the appropriate prompt. Once this has been done it will ask if these values are correct, the default is Y. If Y is specified, the program will proceed to the next stage of initialization else if N is specified the program will return to the beginning of the initialization so that the coordinate values can be retyped.

The second stage of initialization involves the initialization of the smoothing filter. There are 3 filters from which 1 must be chosen. They are : 1. MEAN, 2. MEDIAN, 3. WEIGHTED MEAN MEDIAN.

Filter 1 is a simple averaging filter with an equally weighted window. If selected the MEAN WINDOW SIZE or length must be specified. Filter 2 is a median filter and if selected the MEDIAN WINDOW SIZE must be specified. Filter 3 is a combination of the median filter and the Fading-Memory Polynomial filter. The Fading-Memory Polynomial filter is calculated from the equation (6.13). If selected the MEDIAN WINDOW SIZE must first be specified followed by the weighting factor θ or TIME CONSTANT for the Fading-Memory Polynomial filter. Note the limits of θ are $0 < \theta < 1$. The default is filter 3 with a median window size of 3 and the value for $\theta = 0.89$. After entering in the filter specification the program will proceed to the next stage of initialization.

The third stage of initialization is the fine movement of the AOI window. The AOI can now be seen on the frame grabber output monitor. The following options are available to the user :

Arrow Keys

By pressing one of the arrow keys, the AOI rectangle will move in the direction specified by which arrow key was pressed.

L

L will lock the top left hand corner of the AOI window, and if pressed again it will unlock. The status of the point X1, Y1 can be seen in the middle of the screen to be either FREE or LOCKED.

Grey -

Grey - will select fine movement of the AOI window, which will move 1 pixel at a time in the direction of the selected arrow key. The movement status can be seen in the middle of the screen to be STEP SIZE = 1.

Grey +

Grey + will select coarse movement of the AOI window, which will move 20 pixels at a time in the direction of the selected arrow key. The movement status can be seen in the middle of the screen to be STEP SIZE = 20.

B

B will enable the drawing index (grey level) of the AOI rectangle to be changed. Once selected the current drawing index can be seen at the top right hand corner of the screen, DRAW INDEX = 255. By pressing the grey - or grey +, the drawing index can be decreased or increased between the values of 0 and 255. Pressing B again will exit the drawing index initialization.

Once the AOI window is in the correct position, the initialization routine can be exited by pressing the [Esc] key.

The program has now completed the initialization and the following output screen can be seen.

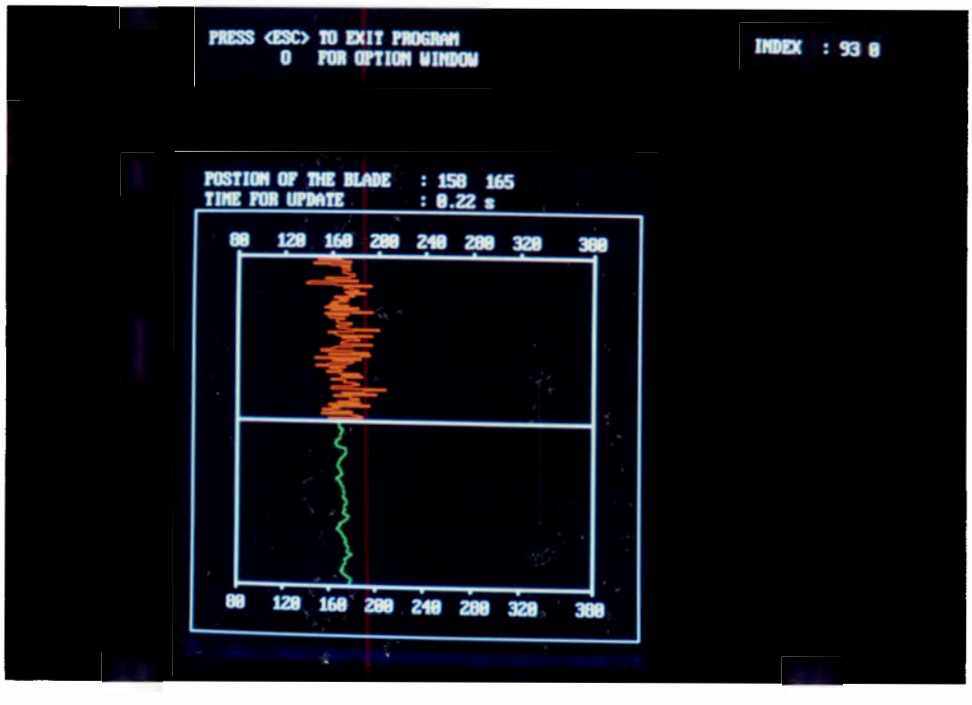


FIGURE B.1 : Main output screen of program EDGE9GC.C

Two graph boxes can be seen. The top box will show in red, the determined edge position and the bottom box will show in green, the filtered edge position. Each box has a height of 128 points and their length ranges from X1 to X2. These boxes will thus show the edge position and filtered edge position over a time frame of 128 sample points. At the top of the graph box, the POSITION OF THE BLADE can be seen, the first number shown is the edge position and the second shown is the filtered

edge position. Underneath shows the TIME FOR UPDATE, which is the time taken for each sample update. The output monitor of the frame grabber will show the AOI window and the determined edge position.

The edge position and the filtered edge position are both stored in 255 circular buffers. The top right hand corner of the screen shows the position of the pointer or INDEX in the circular buffer. The first number shows the current pointer position in the circular buffer. The second number is used when saving the values of the circular buffers and acts as a flag to the position in the buffers from were it must save from.

By typing O, the top left hand corner will shown the options that are available to the user. Note : the option window halts the program, by pressing an option key the program will go to that specific option or else by pressing any other key, the program will be restarted. The following options are available :

B

B will enable the drawing index (grey level) of the AOI rectangle to be changed. Once selected the current drawing index can be seen at the top right hand corner of the screen, DRAW INDEX = 255. By pressing the grey - or grey + the drawing index can be decreased or increased between the values of 0 and 255. Pressing B again will exit the drawing index initialization.

F

F will reset the filter parameters. The options are 1. MEAN, 2. MEDIAN, 3. WEIGHTED MEAN MEDIAN. If filter 1 is selected the MEAN WINDOW SIZE has to be given, and if filter 2 is selected the MEDIAN WINDOW SIZE has to be given. If filter 3 is selected the MEDIAN WINDOW SIZE and the TIME CONSTANT (0 -> 1) has to be given. If filter 3 has been selected it first has to be initialized. INITIALIZING FILTER can be seen in the middle right hand side of the output screen of figure B.1, after the filter parameters have been specified. Also shown is the FILTER WINDOW LENGTH which is the initializing window length L (equation 6.18) derived from θ. The first number is the value of L and the second value shows the length of the recursive averaging window. When these two number are equal the initialization of the filter is complete.

G

G will turn the graph off, and by pressing it again, will turn the graph back on. Note : the program runs slower when it has to draw the graph, so in order for the program to run at optimum speed the graph should be turned off.

I

I will restart the initialization at stage three, where the fine movement of the AOI rectangle is specified.

S

S will allow the edge data and the loss data to be stored. By pressing S, the middle right hand side of the screen will show, DO YOU WISH TO STORE LOSS DATA. If Y is selected, the file name has to be entered, ENTER IN FILENAME. The black loss and white contamination of the current selected losses (Spraying losses, Filter losses or Total losses) will be saved. The file format will be a text file containing two columns separated by a space, the first column will hold the value for the black loss and the second will hold the value for the white contamination. The file is updated at every sample point. Next the program will ask DO YOU WISH TO STORE EDGE DATA. If Y is selected the file name has to be entered, ENTER IN FILENAME. The file format will be a text file with two columns separated by a space, the first column will hold the edge position and the second will hold the filtered edge position. The top right hand corner will show the pointer index to the circular buffer. The first number shows the current pointer index position and the second number shows the flag of the position were saving must take place. Below this, the number of points stored for the edge position and filtered edge position is shown. The file is updated with 255 points of the circular buffers when the two numbers of the INDEX are equal. The edge data file is closed when initialization is restarted and the loss data file is closed when initialization is restarted or different type of losses are selected.

W

W shows the losses due to spraying of the ore, the losses being the amount of black loss and the amount of white contamination. The percentages for each of the values of black loss and white contamination are shown at the middle left of the screen. These values are calculated from equations (7.4) and (7.5). These values are averaged together over a window of size 30, so it will take 30 sample points for the losses to converge to a stable value. To exit this function press W again.

L

L shows the losses brought about purely from the filter lag. The percentages for each of the values of black loss and white contamination are shown at the middle left of the screen. These values are calculated from equations (7.9) and (7.10). These values are calculated from a window of 40 points taken from the circular buffer holding the edge and filtered edge values. The program shows FILTER LOSS INITIALIZATION while 40 new values of the edge and filtered edge are obtained. To exit this function press L again.

T

T shows the total losses which is derived from a combination of the spraying losses and the filter losses. The percentages for each of the values of black loss and white contamination are shown at the middle left of the screen. These values are calculated from equations (7.14) and (7.15). These values are also calculated over a window size of 30 so it will take 30 sample points for the losses to converge to a stable value. To exit this function press T again.

To quit the program press [Esc] from the main output screen of figure B.1.

```
/*************************************************************/
/* PROGRAM EDGE9GC.C TO CALCULATE BLADE POSITION IN SHUTE */
/*  ================================================  */
/*                                                           */
/*       AUTHOR : DAVID GOLD                                 */
/*       DATE   : 1 MARCH 1991                               */
/*************************************************************/



/************************************************************
  *                  Header Files                            *
  ************************************************************/
#include <stdio.h>
#include <ctype.h>
#include <math.h>
#include <conio.h>
#include <float.h>
#include <graph.h>
#include <stdlib.h>
#include <time.h>
#include <malloc.h>
#include <ctype.h>

#define LOC(i,j,cols) (i * cols + j)


/************************************************************
  *                  Function Declarations                   *
  ************************************************************/
void initialize_fgrab(void);
void initialize_variables(int *X1,int *X2,int *Y1,int *Y2,int *wsz,int *ft,
                          int *sflg,int *sps,int *snp,float *th,int *werrflg,
                          int *ferrflg,int *terrflg,int *slossflg);
void initialize_graph(int X1,int X2);
void initialize_circbuff(unsigned int m[],int X1,int X2,unsigned int f[],
                         unsigned int g[],long int r[]);
void initialize_filter(int *wsz,int *ft,float *th,int *n,float *kn,int *fflg);
void initialize_loss_buff(float *bloss,float *wloss,int *lossflg);
void initialize_totalloss_buff(long int *blk,long int *wht,long int *blkloss,
                               int *lossflg);
void get_coord(int *X1,int *X2,int *Y1,int *Y2);
void move_rect(int *X1,int *X2,int *Y1,int *Y2);
void draw_index(int X1,int X2,int Y1,int Y2);
void draw_graph(unsigned int m[],unsigned int fbuff[],int X1,int X2);
void save_data(unsigned int m[],unsigned int fbuff[],int *sflg,int *sps,
               int *spnt,int *slossflg);
void update_data(unsigned int m[],unsigned int fbuff[],int *spnt);
void update_graph(int *snp,int X1,int X2);
void wave_error(int X1,int X2,int ed,unsigned int *grlevel,long int *rav,
                float *bloss,float *wloss,int *lossflg,int slossflg);
void filter_error(unsigned int m[],unsigned int fbuff[],int X1,int ed,
                  unsigned int *grlevel,int flossflg,int slossflg);
void total_error(int X1,int X2,int ed,unsigned int *grlevel,long int *rav,
                 int fltv,long int *blk,long int *wht,long int *blkloss,
                 int *lossflg,int slossflg);
void erralloc(void);
void main(void);
char option_window(void);
int find_edge(int X1,int X2,int Y1,int Y2,unsigned char far *matrix,
```



```
              unsigned int *grlevel,long int *rav);
int get_num(int a);
int flag(int a);
int filter();
int heap_sort(unsigned int f[],int w);



/************************************************************
  *                  Global Variables                        *
  ************************************************************/
unsigned char ptindx;
int cnt;
FILE *fp,*fpl;
char filename[25],filename1[25];



/************************************************************
  *                  Start of Main Program                   *
  ************************************************************/
void main(void)
{
  int x1,x2,y1,y2,edge,wsize,ftype,med,edgecnt,lossflag,slossflag;
  int sflag,spos,spoints,delay,snap,n0,fflag,werrflag,ferrflag,terrflag,
      flossflag;
  long int rave[512],black[32],white[32],blackloss[32];
  unsigned int circbuff[256],sz,filterbuff[256],greylevel[512];
  unsigned int *filterindx;
  char ch;
  unsigned char far *indx;
  float theta,knn,blkloss[32],whtloss[32];
  time_t cstart,cend;

  initialize_fgrab();
  initialize_variables(&x1,&x2,&y1,&y2,&wsize,&ftype,
                       &sflag,&spos,&snap,&theta,
                       &werrflag,&ferrflag,&terrflag,&slossflag);
  get_coord(&x1,&x2,&y1,&y2);
  _clearscreen(_GCLEARSCREEN);
  initialize_filter(&wsize,&ftype,&theta,&n0,&knn,&fflag);
A :
  move_rect(&x1,&x2,&y1,&y2);
  _setvideomode(_VRES16COLOR);
  initialize_graph(x1,x2);
  initialize_circbuff(circbuff,x1,x2,filterbuff,greylevel,rave);
  initialize_loss_buff(blkloss,whtloss,&lossflag);

  sz = (y2 - y1) * 512;
  if (sz == 0)
       sz = 512;
  indx = (unsigned char far *) _fmalloc(sz * sizeof(unsigned char));
  if (indx == NULL)
       erralloc();
  filterindx = (unsigned int *) calloc((wsize + 1),sizeof(unsigned int));
  if (indx == NULL)
       erralloc();

  ch = 0;
  edge = x1 + ((x2 - x1) / 2);
```

```
med = edge;
edgecnt = 0;
ptindx = 0;
flossflag = 0;

_settextposition(1,1);
printf(" PRESS <ESC> TO EXIT PROGRAM     \n");
printf("        O   FOR OPTION WINDOW    \n");
_settextposition(8,2);
printf("POSTION OF THE BLADE   : %d  %d",edge,edge);
_settextposition(9,2);
printf("TIME FOR UPDATE        :  ");

while ((ch != 27) && (ch != 'i') && (ch != 'I'))
{
      for (;;)
      {
            if (kbhit())
                  break;

            cstart = clock();
            fg_snap(snap);
            if (snap == 0)
            {
                  draw_graph(circbuff,filterbuff,x1,x2);
                  for (delay = 0;delay <= 8000;delay++)
                        ;  /* delay to weight for framegrabber to finish*/
            }  /* digitizing 1 image from snap to end of delay*/
            /* must be >= 0.06s*/
            fg_btrans(0,y1,sz,0,1,1,indx,-1);
            edge = find_edge(x1,x2,y1,y2,indx,greylevel,rave);
            ptindx--;
            *(circbuff + ptindx) = edge;

            if ((edgecnt < wsize) && (ftype == 3))
                  edgecnt++;
            else
                  med = (filter(circbuff,filterindx,wsize,ftype,theta,n0,&knn
                              ,&fflag,med));
            *(filterbuff + ptindx) = med;

            if ((flossflag <= 40) && (fflag >= n0))
                  flossflag = flossflag + 1;
            if (werrflag)
                  wave_error(x1,x2,edge,greylevel,rave,blkloss,whtloss,&
                              lossflag,slossflag);
            if (ferrflag)
                  filter_error(circbuff,filterbuff,x1,edge,greylevel,
                              flossflag,slossflag);
            if ((terrflag) && (fflag > n0))
                  total_error(x1,x2,edge,greylevel,rave,med,
                              black,white,blackloss,&lossflag,slossflag);
            if ((werrflag == 0) && (ferrflag == 0) && (terrflag == 0))
            {
                  _settextposition(8,2);
                  printf("POSTION OF THE BLADE   : %d  %d ",edge,med);
            }
            _settextposition(1,60);
            printf("INDEX  : %d %d    ",ptindx,spos);
            fg_rect(x1,y1,x2,y2);
```



```
            fg_rect(x1,y1,edge,y2);
            if ((sflag == 1) && (spos == (int) ptindx))
                  update_data(circbuff,filterbuff,&spoints);
            _settextposition(9,27);
            cend = clock();
            printf("%4.2f s  ",((float) cend - cstart) / CLK_TCK);
      }
      ch = (char) getch();
      if ((ch == 'o') || (ch == 'O'))
            ch = (option_window());
      switch (ch)
      {
            case 'b':
            case 'B':
                  draw_index(x1,x2,y1,y2);
                  break;
            case 's':
            case 'S':
            {
                  if ((sflag == 0) || (slossflag == 0))
                        save_data(circbuff,filterbuff,&sflag,&spos,&spoints,&
                                    slossflag);
                  break;
            }
            case 'f':
            case 'F':
            {
                  initialize_filter(&wsize,&ftype,&theta,&n0,&knn,&fflag);
                  edgecnt = 0;
                  flossflag = 0;
                  break;
            }
            case 'g':
            case 'G':
                  update_graph(&snap,x1,x2);
                  break;
            case 'w':
            case 'W':
                  if ((ferrflag == 1) || (terrflag == 1))
                        break;
                  werrflag = flag(werrflag);
                  if (werrflag == 0)
                  {
                        _settextposition(6,1);
                        printf("                                        ");
                        _settextposition(7,1);
                        printf("                                        ");
                        _settextposition(8,1);
                        printf("                                        ");
                        if (slossflag == 1)
                        {
                              fclose(fpl);
                              slossflag = 0;
                        }
                  }
                  else
                        initialize_loss_buff(blkloss,whtloss,&lossflag);
                  break;
            case 'l':
            case 'L':
```

```
                if ((werrflag == 1) || (terrflag == 1))
                        break;
                ferrflag = flag(ferrflag);
                if (ferrflag == 0)
                {
                        _settextposition(6,1);
                        printf("                                        ");
                        _settextposition(7,1);
                        printf("                                        ");
                        _settextposition(8,1);
                        printf("                                        ");
                        if (slossflag == 1)
                        {
                                fclose(fpl);
                                slossflag = 0;
                        }
                }
                break;
        case 't':
        case 'T':
                if ((ferrflag == 1) || (werrflag == 1))
                        break;
                terrflag = flag(terrflag);
                if (terrflag == 0)
                {
                        _settextposition(6,1);
                        printf("                                        ");
                        _settextposition(7,1);
                        printf("                                        ");
                        _settextposition(8,1);
                        printf("                                        ");
                        if (slossflag == 1)
                        {
                                fclose(fpl);
                                slossflag = 0;
                        }
                }
                else
                        initialize_totalloss_buff(black,white,blackloss,&
                                                  lossflag);
                break;
        }
}
_ffree(indx);
if (sflag == 1)
        fclose(fp);
if (slossflag == 1)
        fclose(fpl);
if ((ch == 'i') || (ch == 'I'))
{
        sflag = 0;
        slossflag = 0;
        spos = 0;
        snap = 0;
        knn = 0;
        fflag = 0;
        edgecnt = 0;
        goto A;
}
_clearscreen(_GCLEARSCREEN);
```



```
  _setvideomode(_DEFAULTMODE);
}



/*********************************************************************
  *  PROCEDURE :  get_coord()                                         *
  *                                                                   *
  *  PURPOSE :   Reads in from the keyboard the values for the        *
  *              area of interest rectangle (X1,X2,Y1,Y2)             *
  *                                                                   *
  *  INPUT : X1,X2,Y1,Y2  : Initialized rectangle position            *
  *                                                                   *
  *  OUTPUT : X1,X2,Y1,Y2  : Input rectangle position from keyboard   *
  *                                                                   *
  *  RETURNS :  None                                                  *
  *                                                                   *
  *********************************************************************/

void get_coord(X1,X2,Y1,Y2)
int *X1,*X2,*Y1,*Y2;
{
  int ch;

  _clearscreen(_GCLEARSCREEN);
  _settextposition(1,1);
  printf(" ENTER IN THE COORDINATES OF X1,Y1 AND X2,Y2 OF THE        \n");
  printf(" AREA OF INTEREST   Note: The amount of pixel rows         \n");
  printf("                          may not exeed 100                \n");

  do
  {
        _settextposition(11,30);
        printf("X1 = %d",*X1);
        ch = getch();
        if (ch != 13)
        {
              _settextposition(11,35);
              printf("          ");
              _settextposition(11,35);
              *X1 = get_num(ch);  /* get_num tests ch to*/
        }  /*be a valid number input */
        _settextposition(12,30);
        printf("Y1 = %d",*Y1);
        ch = getch();
        if (ch != 13)
        {
              _settextposition(12,35);
              printf("          ");
              _settextposition(12,35);
              *Y1 = get_num(ch);
        }

        _settextposition(14,30);
        printf("X2 = %d",*X2);
        ch = getch();
        if (ch != 13)
        {
              _settextposition(14,35);
```

```
            printf("            ");
            _settextposition(14,35);
            *X2 = get_num(ch);
      }

      _settextposition(15,30);
      printf("Y2 = %d",*Y2);
      ch = getch();
      if (ch != 13)
      {
            _settextposition(15,35);
            printf("            ");
            _settextposition(15,35);
            *Y2 = get_num(ch);
      }
      ch = 'Y';
      printf("\n\n\n\n\nARE THESE VALUE CORRECT : %c",ch);
      ch = getch();
      _settextposition(20,27);
      printf("%c",ch);
  }
  while ((ch == 'n') || (ch == 'N'));
}



/*********************************************************************
  *  PROCEDURE :  move_rect()                                         *
  *                                                                   *
  *  PURPOSE : Moves the area of interest rectngle around by using    *
  *            the display functions of the Framegrabber.  Control    *
  *            keys are :                                             *
  *                     1. Arrow keys  for movement of rectangle      *
  *                     2. L fixes the top left hand corner of rect   *
  *                     3. -/+ coarce or fine movement (1 or 10)      *
  *                     4. B -/+ drawing index of rectangle(0 to 255)*
  *                     5. <esc> to exit routin                       *
  *                                                                   *
  *  INPUT :  X1,X2,Y1,Y2  : Input rectangle position                 *
  *                                                                   *
  *  OUTPUT : X1,X2,Y1,Y2  :  Correct rectangle position              *
  *                                                                   *
  *  RETURNS : None                                                   *
  *                                                                   *
  *********************************************************************/

void move_rect(X1,X2,Y1,Y2)
int *X1,*X2,*Y1,*Y2;
{
  char ch;
  int step,flg;

  step = 1;
  flg = 0;

  _clearscreen(_GCLEARSCREEN);
  _settextposition(1,1);
  printf(" PRESS <ESC> TO EXIT INTIALIZATION                    \n");
  printf("                                                      \n");
```



```
  printf("                                                      \n");
  printf("                moves rectangle with or without corner placed\n");
  printf("        ↓                                             \n");
  printf("                                                      \n");
  printf("        L       places corner X1 Y1                   \n");
  printf("        -       sets fine movement                    \n");
  printf("        +       sets coarse movement                  \n");
  printf("    B  -/+      sets drawing index                    \n");
  _settextposition(4,11);
  putch(26);
  _settextposition(3,9);
  putch(24);
  _settextposition(4,7);
  putch(27);

  _settextposition(14,30);
  printf("STEP SIZE = %d",step);
  _settextposition(15,30);
  if (flg == 1)
       printf("X1,Y1 : LOCKED");
  else
       printf("X1,Y1 : FREE");
  _settextposition(17,30);
  printf("X1 = %d",*X1);
  _settextposition(18,30);
  printf("Y1 = %d",*Y1);
  _settextposition(20,30);
  printf("X2 = %d",*X2);
  _settextposition(21,30);
  printf("Y2 = %d",*Y2);

  do
  {
       if (kbhit())
       {
            ch = (char) getch();
            if ((ch == 'l') || (ch == 'L'))  /* fixes left hand corner of*/
            {  /* area of interest rectangle */
                 flg = flag(flg);
                 _settextposition(15,30);
                 if (flg == 1)
                      printf("X1,Y1 : LOCKED  ");
                 else
                      printf("X1,Y1 : FREE    ");
            }
            if ((ch == 'b') || (ch == 'B'))  /* set draw index of rect */
                 draw_index(*X1,*X2,*Y1,*Y2);
            switch (ch)
            {
                 case 43:  /* grey plus key */
                      step = 20;
                      _settextposition(14,42);
                      printf("%d ",step);
                      break;
                 case 45:  /* grey minus key */
                      step = 1;
                      _settextposition(14,42);
                      printf("%d ",step);
                      break;
                 case 75:  /* left arrow key */
```

```
            if (flg == 0)
            {
                  *X1 -= step;
                  *X2 -= step;
                  _settextposition(17,35);
                  printf("%d ",*X1);
                  _settextposition(20,35);
                  printf("%d ",*X2);
            }
            else
            {
                  *X2 -= step;
                  if (*X2 < *X1)
                        *X2 = *X1;
                  _settextposition(20,35);
                  printf("%d ",*X2);
            }
            break;
      case 77:  /* right arrow key */
            if (flg == 0)
            {
                  *X1 += step;
                  *X2 += step;
                  _settextposition(17,35);
                  printf("%d ",*X1);
                  _settextposition(20,35);
                  printf("%d ",*X2);
            }
            else
            {
                  *X2 += step;
                  _settextposition(20,35);
                  printf("%d ",*X2);
            }
            break;
      case 80:  /* down arrow key */
            if (flg == 0)
            {
                  *Y1 += step;
                  *Y2 += step;
                  _settextposition(18,35);
                  printf("%d ",*Y1);
                  _settextposition(21,35);
                  printf("%d ",*Y2);
            }
            else
            {
                  *Y2 += step;
                  if (*Y2 - *Y1 >= 100)  /*limit rect to*/
                        *Y2 = *Y1 + 100;  /* width of 100*/
                  _settextposition(21,35);
                  printf("%d ",*Y2);
            }
            break;
      case 72:  /* up arrow key */
            if (flg == 0)
            {
                  *Y1 -= step;
                  *Y2 -= step;
                  _settextposition(18,35);
```



```
                  printf("%d ",*Y1);
                  _settextposition(21,35);
                  printf("%d ",*Y2);
            }
            else
            {
                  *Y2 -= step;
                  if (*Y2 < *Y1)
                        *Y2 = *Y1;
                  _settextposition(21,35);
                  printf("%d ",*Y2);
            }
            break;
         }
      }
      fg_snap(1);
      fg_rect(*X1,*Y1,*X2,*Y2);
   }
   while (ch != 27);
}



/*********************************************************************
  *  PROCEDURE : initialize_variables()                               *
  *                                                                   *
  *  PURPOSE :  initializes all the variables local to main()         *
  *                                                                   *
  *  INPUT / OUTPUT :                                                 *
  *   1.  X1,X2,Y1,Y2  : Rectangle position                           *
  *   2.  wsz : Window size for the mean/median filter                *
  *   3.  ft  : Filter type 1:mean, 2:median, 3:median weighted mean  *
  *   4.  sflg: flag determining update of a file                     *
  *             0 : dont update file                                  *
  *             1 : update file                                       *
  *   5.  sps : pointer posititon used in saving edge position data   *
  *   6.  snp : snap flag used in framegrabber                        *
  *             0 : return  immediately                               *
  *             1 : waits until framegrab is complete                 *
  *   7.  th  :  Time constant used in filter 3                       *
  *   8. werrflg : determine wave loss 0 no 1 yes                     *
  *   9. ferrflg  : determine filter loss 0 no 1 yes                  *
  *   10. terrflg : determine total loss 0 no 1 yes                   *
  *   11. slossflg : save the loss values for the 8,9,10 .            *
  *                  0 dont save 1 save                               *
  *  RETURNS :  None                                                  *
  *                                                                   *
  *********************************************************************/

void initialize_variables(X1,X2,Y1,Y2,wsz,ft,sflg,sps,snp,th,werrflg,ferrflg,
                          terrflg,slossflg)
int *X1,*X2,*Y1,*Y2,*ft,*wsz,*sflg,*sps,*snp,*werrflg,*ferrflg,*terrflg,*
    slossflg;
float *th;
{
  *X1 = 80;
  *X2 = 380;
  *Y1 = 315;
  *Y2 = 325;
```

```
  cnt = 250;
  *ft = 3;
  *wsz = 3;
  *sflg = 0;
  *sps = 0;
  *snp = 0;
  *th = (float) 0.89;
  *werrflg = 0;
  *ferrflg = 0;
  *terrflg = 0;
  *slossflg = 0;
}



/*********************************************************************
  *  PROCEDURE : initialize_fgrab()                                   *
  *                                                                   *
  *  PURPOSE :  Initializes the framegraber                           *
  *                                                                   *
  *  INPUT :  None                                                    *
  *                                                                   *
  *  OUTPUT : None                                                    *
  *                                                                   *
  *  RETURNS : None                                                   *
  *                                                                   *
  *********************************************************************/

void initialize_fgrab(void)
{
  fg_inifmt(0x26c,1,0,0,1,0);  /*iniatialize the frame grabber*/
  fg_chan(2);  /* input channel 2 */
  fg_sync(1);  /* external sync    */
  fg_setind(255);  /* drawing index 255 */
  fg_sbuf(1);  /* display output buffer */
  fg_snap(1);  /* snap a frame          */
}



/*********************************************************************
  *  PROCEDURE : initialize_graph()                                   *
  *                                                                   *
  *  PURPOSE : Draws and labels graph axis                            *
  *                                                                   *
  *  INPUT : X1,X2 width of rectangle/graph window                    *
  *                                                                   *
  *  OUTPUT : None                                                    *
  *                                                                   *
  *  RETURNS : None                                                   *
  *                                                                   *
  *********************************************************************/

void initialize_graph(X1,X2)
int X1,X2;
{
  int c,text,x;
```



```
  _setcolor(7);
  _setviewport(0,140,(X2 - X1) + 80,480);
  _rectangle(_GBORDER,0,4,(X2 - X1) + 80,339);
  _setcolor(15);
  _rectangle(_GBORDER,39,39,(X2 - X1) + 41,300);
  _moveto(39,170);
  _lineto((X2 - X1) + 41,170);

  c = 39;
  text = 5;
  x = X1;

  do
  {
        _moveto(c,39);  /*place tics on axis*/
        _lineto(c,36);
        _settextposition(11,text);
        printf("%d",x);
        _moveto(c,300);
        _lineto(c,303);
        _settextposition(29,text);
        printf("%d",x);

        text += 5;
        c += 40;
        x += 40;
  }
  while (c <= (X2 - X1) - 1);

  _moveto((X2 - X1) + 41,39);
  _lineto((X2 - X1) + 41,36);
  _settextposition(11,((X2 - X1) + 41) / 8);
  printf("%d",X2);
  _moveto((X2 - X1) + 41,300);
  _lineto((X2 - X1) + 41,303);
  _settextposition(29,((X2 - X1) + 41) / 8);
  printf("%d",X2);
}



/*********************************************************************
  *  PROCEDURE : initialize_filter()                                  *
  *                                                                   *
  *  PURPOSE :  Initializes the choice of filter, the window size     *
  *                                                                   *
  *  INPUT / OUTPUT :                                                 *
  *   1. wsz : Window size for the mean/median filter                 *
  *   2. ft  : Filter type 1:mean, 2:median, 3:median weighted mean   *
  *   3. th  : Time constant theta for filter 3                       *
  *   4. n   : Number of initializing points in filter 3              *
  *   5. kn  : Running mean for initializing filter 3                 *
  *   6. fflg: Flag for initializing filter 3. Initializing ends      *
  *            when fflg = n                                          *
  *                                                                   *
  *  RETURNS : None                                                   *
  *                                                                   *
  *********************************************************************/
```

```
void initialize_filter(wsz,ft,th,n,kn,fflg)
int *wsz,*ft,*n,*fflg;
float *th,*kn;
{
  int ch;
  char temp[10];

  _settextposition(3,45);
  printf("          1 : MEAN");
  _settextposition(4,45);
  printf("          2 : MEDIAN");
  _settextposition(5,45);
  printf("          3 : WEIGHTED MEAN MEDIAN");
  _settextposition(7,45);
  printf("FILTER  : %d",*ft);

  do
  {
        ch = getch();
        if (ch != 13)
        {
              _settextposition(7,55);
              printf("          ");
              _settextposition(7,55);
              *ft = get_num(ch);
        }
  }
  while ((*ft != 1) && (*ft != 2) && (*ft != 3) && (*ft != 13));

  _settextposition(8,45);
  if (*ft == 1)
        printf("MEAN  WINDOW  SIZE : %d",*wsz);
  else
        printf("MEDIAN WINDOW SIZE : %d",*wsz);
  ch = getch();
  if (ch != 13)
  {
        _settextposition(8,66);
        printf("          ");
        _settextposition(8,66);
        *wsz = get_num(ch);
  }

  if (*ft == 3)
  {
        _settextposition(9,45);
        printf("TIME CONST (0->1) : %5.4f",*th);

        ch = getch();

        if ((ch != 13) && (ch == '0'))
        {
              _settextposition(9,65);
              printf("          ");
              _settextposition(9,65);
              printf("0");
              gets(temp);
              *th = (float) atof(temp);
        }
        *n = (int) ((2 * *th / (1 - *th)) + 0.5);  /* calculates length of */
```



```
        *kn = 0;  /* initializing window size */
        *fflg = 0;  /* fflag incremental flag for initializing*/
  }             /* initializing end when fflag = n*/
  _settextwindow(2,45,9,80);
  _clearscreen(_GWINDOW);
  _settextwindow(1,1,30,80);
}



/*********************************************************************
  *  PROCEDURE :  initialize_circbuff()                               *
  *                                                                   *
  *  PURPOSE : Initializes buffers for edge and filtered edge         *
  *            location                                               *
  *                                                                   *
  *  INPUT :   X1,X2  Width of rectangle/graph window                 *
  *            m      Edge location buffer                            *
  *            f      Filtered edge location buffer                   *
  *            g      buffer that holds the average grey level profile*
  *                   of ore                                          *
  *            r      right average value buffer from diff of average*
  *                   output                                          *
  *                                                                   *
  *  OUTPUT :  m  Edge location buffer  set to middle of AOI window   *
  *            f  Filtered edge location buffer set to mid AOI window*
  *            g  buffer that holds the average grey level profile   *
  *               of ore set to 0                                     *
  *            r  right average value buffer from diff of average .  *
  *               output set to 0                                     *
  *                                                                   *
  *  RETURNS : None                                                   *
  *                                                                   *
  *********************************************************************/

void initialize_circbuff(m,X1,X2,f,g,r)
int X1,X2;
unsigned int m[],f[],g[];
long int r[];
{
  unsigned int c,cst;

  cst = X1 + ((X2 - X1) / 2);  /* middle value of AOI window*/

  for (c = 0;c <= 255;c++)
  {
        m [c] = cst;
        f [c] = cst;
  }

  for (c = 0;c <= 511;c++)
  {
        g [c] = 0;
        r [c] = 0;
  }
}
```

```
/*********************************************************************
  *  PROCEDURE : initialize_loss_buff()                               *
  *                                                                   *
  *  PURPOSE : Initializes the circular buffer of size 30 used to     *
  *            hold wave loss error percentage values                 *
  *                                                                   *
  *  INPUT :  bloss : black loss percentage values                    *
  *           wloss :  white contamination percentage value           *
  *           lossflg : flag pointer to position in buffer            *
  *                                                                   *
  *  OUTPUT : bloss : black loss percentage value set to 0            *
  *           wloss :  white contamination percentage value set to 0  *
  *           lossflg : flag pointer to position in buffer set to 0   *
  *                                                                   *
  *  RETURNS : None                                                   *
  *                                                                   *
  *********************************************************************/

void initialize_loss_buff(bloss,wloss,lossflg)
float *bloss,*wloss;
int *lossflg;
{
  int c;

  for (c = 0;c <= 30;c++)
  {
        bloss [c] = (float) 0.0;
        wloss [c] = (float) 0.0;
  }
  *lossflg = 0;
}



/*********************************************************************
  *  PROCEDURE : initialize_totalloss_buff()                          *
  *                                                                   *
  *  PURPOSE : Initializes the circular buffer of size 30 used to     *
  *            hold total loss error  values                          *
  *                                                                   *
  *  INPUT :  blk : black gained value                                *
  *           wloss :  white contamination value                      *
  *           blkloss: black loss  value                              *
  *           lossflg : flag pointer to position in buffer            *
  *                                                                   *
  *  OUTPUT : blk : black gained value set to 0                       *
  *           wloss :  white contamination value set to 0             *
  *           blkloss: black loss  value set to 0                     *
  *           lossflg : flag pointer to position in buffer set to 0   *
  *                                                                   *
  *  RETURNS : None                                                   *
  *                                                                   *
  *********************************************************************/

void initialize_totalloss_buff(blk,wht,blkloss,lossflg)
int *lossflg;
long int *blk,*wht,*blkloss;
{
  int c;
```



```
  for (c = 0;c <= 30;c++)
  {
        wht [c] = 0;
        blk [c] = 0;
        blkloss [c] = 0;
  }
  *lossflg = 0;
}



/*********************************************************************
  *  PROCEDURE : get_num()                                            *
  *                                                                   *
  *  PURPOSE : Reads in a character from keyboard checks for valid    *
  *            integer number and appends it to the number to be      *
  *            returned                                               *
  *                                                                   *
  *  INPUT : a First digit in the number                              *
  *                                                                   *
  *  OUTPUT : None                                                    *
  *                                                                   *
  *  RETURNS : num The integer number scaned from the keyboard        *
  *                                                                   *
  *********************************************************************/

int get_num(a)
int a;
{
  int num,b;

  num = 0;
  b = a;
  a = 0;

  for (;;)
  {
        if (b == 8)  /* backspace typed ?*/
        {
              printf("\b \b");
              num = num / 10;
        }
        if ((isdigit(b)) || (b == 13))  /* is a valid number 0-9*/
        {
              switch (b)
              {
                    case '0':
                          a = 0;
                          break;
                    case '1':
                          a = 1;
                          break;
                    case '2':
                          a = 2;
                          break;
                    case '3':
                          a = 3;
```

```
                        break;
                case '4':
                        a = 4;
                        break;
                case '5':
                        a = 5;
                        break;
                case '6':
                        a = 6;
                        break;
                case '7':
                        a = 7;
                        break;
                case '8':
                        a = 8;
                        break;
                case '9':
                        a = 9;
                        break;
                case 13:
                        return (num);
                default:
                        a = 0;
            }
            if (num <= 32000)  /*restrict number size*/
            {
                printf("%d",a);
                num = num * 10 + a;
            }
        }
        b = getch();
    }
}



/*********************************************************************
  *  PROCEDURE : flag()                                               *
  *                                                                   *
  *  PURPOSE : Toggles the value of a flag                            *
  *                                                                   *
  *  INPUT : a Flag value 0 or 1                                      *
  *                                                                   *
  *  OUTPUT : None                                                    *
  *                                                                   *
  *  RETURNS : Flag value 0 or 1                                      *
  *                                                                   *
  *********************************************************************/

int flag(a)
int a;
{
  switch (a)
  {
        case 0:
                a = 1;
                break;
        case 1:
```



```
                a = 0;
                break;
  }
  return (a);
}



/*********************************************************************
  *  PROCEDURE : draw_index()                                         *
  *                                                                   *
  *  PURPOSE : Sets drawing index for the framegraber                 *
  *                                                                   *
  *  INPUT : X1,X2,Y1,Y2  Rectangle position                          *
  *                                                                   *
  *  OUTPUT :  None                                                   *
  *                                                                   *
  *  RETURNS :  None                                                  *
  *                                                                   *
  *********************************************************************/

void draw_index(X1,X2,Y1,Y2)
int X1,X2,Y1,Y2;
{
  char ch;

  _settextposition(1,60);
  printf("                ");
  _settextposition(1,60);
  printf("DRAW INDEX = %d  ",cnt);

  do
  {
        if (kbhit())
        {
                ch = (char) getch();
                switch (ch)
                {
                        case 43:  /*grey + */
                                cnt += 5;
                                if (cnt > 255)
                                        cnt = 255;
                                break;
                        case 45:  /*grey - */
                                cnt -= 5;
                                if (cnt < 0)
                                        cnt = 0;
                                break;
                }
        }
        fg_snap(1);
        fg_setind(cnt);
        fg_rect(X1,Y1,X2,Y2);
        _settextposition(1,60);
        printf("DRAW INDEX = %d  ",cnt);
  }
  while ((ch != 'b') && (ch != 'B'));
  _settextposition(1,60);
```

```
  printf("                ");
}



/*************************************************************************
  *  PROCEDURE :  erralloc()                                              *
  *                                                                       *
  *  PURPOSE : Exit to DOS when unable to allocate space for arrays       *
  *                                                                       *
  *  INPUT : None                                                         *
  *                                                                       *
  *  OUTPUT : None                                                        *
  *                                                                       *
  *  RETURNS : None                                                       *
  *                                                                       *
  *************************************************************************/

void erralloc(void)
{
  printf("\nUnable to allocate space for array\n");
  printf("Memory Availabe %u\n",_memavl());
  exit(1);
}



/*************************************************************************
  *  PROCEDURE : draw_graph()                                             *
  *                                                                       *
  *  PURPOSE : Sets viewports and draws graph of edge location as         *
  *             well as filtered edge location                            *
  *                                                                       *
  *  INPUT :   m      Edge location buffer                                *
  *            fbuff  Filtered edge location buffer                       *
  *            X1,X2  Width of rectangle                                  *
  *                                                                       *
  *  OUTPUT :  None                                                       *
  *                                                                       *
  *  RETURNS : None                                                       *
  *                                                                       *
  *************************************************************************/

void draw_graph(m,fbuff,X1,X2)
unsigned int m[],fbuff[];
int X1,X2;
{
  int indx,xvaldel,xvalin;
  unsigned char delindx,cont;
  unsigned int *pt1,*pt2;

  pt1 = m;  /* pointer to edge buffer*/
  pt2 = fbuff;  /* pointer to filter buffer*/

  _setviewport(40,180,(X2 - X1) + 40,310);
  delindx = ptindx;
  delindx++;  /* pointer to previous values drawn so can be errased*/
```



```
  xvaldel = *(pt1 + delindx) - X1;
  _setcolor(0);
  _setpixel(xvaldel,0);
  xvalin = *(pt1 + ptindx) - X1;
  _setcolor(4);
  _setpixel(xvalin,0);

  cont = ptindx;

  for (indx = 0;indx < 128;indx++)
  {
        _moveto(xvaldel,indx);
        delindx++;
        ptindx++;
        xvaldel = *(pt1 + delindx) - X1;
        _setcolor(0);
        _lineto(xvaldel,indx);
        _moveto(xvalin,indx);
        xvalin = *(pt1 + ptindx) - X1;
        _setcolor(4);
        _lineto(xvalin,indx);
  }
  ptindx = cont;


  _setviewport(40,311,(X2 - X1) + 40,440);
  delindx = ptindx;
  delindx++;
  xvaldel = *(pt2 + delindx) - X1;
  _setcolor(0);
  _setpixel(xvaldel,0);
  xvalin = *(pt2 + ptindx) - X1;
  _setcolor(2);
  _setpixel(xvalin,0);

  cont = ptindx;

  for (indx = 0;indx < 128;indx++)
  {
        _moveto(xvaldel,indx);
        delindx++;
        ptindx++;
        xvaldel = *(pt2 + delindx) - X1;
        _setcolor(0);
        _lineto(xvaldel,indx);
        _moveto(xvalin,indx);
        xvalin = *(pt2 + ptindx) - X1;
        _setcolor(2);
        _lineto(xvalin,indx);
  }
  ptindx = cont;
}



/*************************************************************************
  *  PROCEDURE : find_edge()                                              *
  *                                                                       *
  *  PURPOSE : Detects the position of the edge by the use of             *
```

```
 *           diference of averages                                      *
 *                                                                      *
 *  INPUT :  X1,X2,Y1,Y2  Rectangle position                            *
 *           matrix       Base address of 1-D matrix of data           *
 *                        transfered from framegraber                  *
 *           grlevel buffer of average grey level of ore                *
 *           rav     buffer of right average value of diff of avgerage*
 *                    output                                            *
 *                                                                      *
 *  OUTPUT : grlevel average grey level of ore                          *
 *           rav     right average value of diff of avgerage           *
 *                   output                                             *
 *                                                                      *
 *  RETURNS : ed Integer edge position                                  *
 *                                                                      *
 ************************************************************************/

int find_edge(X1,X2,Y1,Y2,matrix,grlevel,rav)
int X1,X2,Y1,Y2;
unsigned char far *matrix;
unsigned int *grlevel;
long int *rav;
{
  int a,i,j,ed,numx,numy;
  unsigned long int total,coltot,numtot,ltot,rtot,x;
  long int temp,Lave,maxval;

  total = 0;
  maxval = -100000;  /* initialize value used in Diff Of Ave comparison */
  ed = 0;  /* edge value */
  numx = X2 - X1;  /* number of pixels in x direction*/
  numy = Y2 - Y1;  /* number of pixels in y direction */

  if (numx == 0)
       numx = 1;
  if (numy == 0)
       numy = 1;
  numtot = numx * numy;  /* toatl amount of pixels in AOI window*/


  for (i = 0;i < numy;i++)  /* total grey level value in AOI window */
       for (j = X1;j < X2;j++)
            total = total + matrix [LOC(i,j,512)];  /* LOC = (i*cols + j) */

  ltot = 0;  /*total in left diff of ave window*/
  rtot = total;  /*total in right diff of ave window*/
  a = 0;

  for (j = X1;j < X2 - 1;j++)
  {
       coltot = 0;  /* column total*/
       a += 1;

       for (i = 0;i < numy;i++)
            coltot = coltot + matrix [LOC(i,j,512)];
       grlevel [j] = (int) (coltot / (unsigned long int) numy);
       ltot = ltot + coltot;
       rtot = rtot - coltot;

       x = numtot - a * numy;  /* number of elements in right window*/
```



```
       if (x == 0)
            x = 1;
       rav [j] = (rtot * 1000) / x;  /*multiply by 1000 to get 3 decimal*/
       Lave = (ltot * 1000) / (a * numy);  /* places */
       temp = rav [j] - Lave;
       if (temp >= maxval)
            maxval = temp,ed = j;  /*test to see if output is maximum*/
  }
  if (X1 == 0)
       ed += 1;
  return (ed);
}




/************************************************************************
  *  PROCEDURE :  filter()                                              *
  *                                                                     *
  *  PURPOSE :Implements the filters Filter type 1:mean, 2:median,      *
  *           3:median weighted mean. Mean filter is unweighted         *
  *           averaging over a window, Median is the median value in    *
  *           a window, Median weighted mean is the data is first       *
  *           median filterd then averaged acording to a exponential    *
  *           window.  This filter uses a runing average therfore it    *
  *           first has to be initialized                               *
  *                                                                     *
  *  INPUT :   m      Edge location buffer                              *
  *            findx  Pointer to mean/median window buffer              *
  *            wsz    Mean/median window size                           *
  *            ft     Filter type 1:mean, 2:median, 3:median weighted   *
  *                    mean                                             *
  *            th     Filter 3 time constant                            *
  *            n      Number of initializing points in filter 3         *
  *            kn     Running mean for initializing filter 3            *
  *            fflg   Flag for initializing filter 3.                   *
  *                   Initializing ends when fflg = n                   *
  *            med    Last value of filtered edgelocation               *
  *                                                                     *
  *  OUTPUT : None                                                      *
  *                                                                     *
  *  RETURNS : Integer filtered edge location                           *
  *                                                                     *
  ************************************************************************/

int filter(m,findx,wsz,ft,th,n,kn,fflg,med)
unsigned int m[],findx[];
int wsz,ft,n,*fflg,med;
float th,*kn;
{
  int c,median,avev,filtval;
  unsigned char a;
  float totl;

  a = ptindx; /* pointer to circular filter buffer */

  for (c = 1;c <= wsz;c++)
  {
       *(findx + c) = *(m + a);  /* transfer filter values to temporary */
```

```
        a++;  /* matrix starting at base find */
    }

    if ((ft == 2) || (ft == 3))
    {
        if (wsz >= 2)
            median = heap_sort(findx,wsz);
        else
            median = *(m + ptindx);  /* if window size is less than 2*/
        if (ft == 2)  /* no heapsort returns input edge value*/
            return (median);
        else
        {
            if (*fflg <= n)
            {
                _settextposition(4,45);
                printf(" INITIALIZING FILTER ");
                _settextposition(5,45);
                printf(" FILTER  WINDOW LENGTH : %d %d",n,*fflg);

                if (*fflg == n)
                {
                    _settextposition(4,45);
                    printf("                          ");
                    _settextposition(5,45);
                    printf("                                        ");
                }  /* initializing*/
                *kn = (*kn * *fflg + median) / (*fflg + 1);  /*runing
                                                               averge*/
                *fflg += 1;  /*filter flag*/
                return (med);
            }
            else
            {
                *kn = th * *kn + (1 - th) * median;  /*filter output
                                                       based*/
                filtval = (int) *kn;  /* on exp weighted window*/
                return (filtval);
            }
        }
    }
    else
    {
        totl = (float) 0.0;  /* filter 1 calculates the mean value */
        for (c = 1;c <= wsz;c++)
            totl += *(findx + c);
        avev = (int) ((totl / (float) wsz) + (float) 0.5);
        return (avev);
    }
}



/*********************************************************************
 *  PROCEDURE : heap_sort()                                          *
 *                                                                   *
 *  PURPOSE : Edge values sorted from biggest to smallest using      *
```



```
 *            the heap sort algorithm                              *
 *                                                                 *
 *  INPUT : f Pointer to median window buffer                      *
 *          w Number of elements in buffer                         *
 *                                                                 *
 *  OUTPUT : f Pointer to sorted median window buffer              *
 *                                                                 *
 *  RETURNS : None                                                 *
 *                                                                 *
 *******************************************************************/

int heap_sort(f,w)
unsigned int f[];
int w;
{
    int l,j,ir,i,median,medval;
    unsigned int rra;

    l = (w >> 1) + 1;
    ir = w;

    for (;;)
    {
        if (l > 1)
            rra = f [--l];
        else
        {
            rra = f [ir];
            f [ir] = f [1];
            if (--ir == 1)
            {
                f [1] = rra;
                break;
            }
        }
        i = l;
        j = l << 1;
        while (j <= ir)
        {
            if (j < ir && f [j] < f [j + 1])
                ++j;
            if (rra < f [j])
            {
                f [i] = f [j];
                j += (i = j);
            }
            else
                j = ir + 1;
        }
        f [i] = rra;
    }

    medval = (int) w / 2;
    median = f [medval + 1];
    return (median);
}
```

```
/*********************************************************************
  *  PROCEDURE : save_data()                                          *
  *                                                                   *
  *  PURPOSE : Saves the buffers of the edge and filtered edge        *
  *            locations                                              *
  *                                                                   *
  *  INPUT : m      Edge location buffer                              *
  *          fbuff  Filtered edge location buffer                     *
  *          sflg:  flag determining update of a file                 *
  *                    0 : dont update file                           *
  *                    1 : update file                                *
  *          sps    pointer posititon of circular buffers             *
  *                 used in saving edge positiondata                  *
  *          spnt   number of points stored to date                   *
  *          slossflg update loss 0 no 1 yes                          *
  *                                                                   *
  *  OUTPUT : sflg:  flag determining update of a file                *
  *                    0 : dont update file                           *
  *                    1 : update file                                *
  *           sps    pointer posititon of circular buffer             *
  *                  used in saving edge position data                *
  *           spnt   number of points stored to date                  *
  *          slossflg update loss 0 no 1 yes                          *
  *                                                                   *
  *  RETURNS : None                                                   *
  *                                                                   *
  *********************************************************************/

void save_data(m,fbuff,sflg,sps,spnt,slossflg)
unsigned int m[],fbuff[];
int *sflg,*sps,*spnt,*slossflg;
{

  char ch;
  unsigned int *pt1,*pt2,c;

  ch = 'N';

  if (*slossflg == 0)
  {
        _settextposition(7,40);
        printf("DO YOU WISH TO STORE LOSS DATA :  %c",ch);
        ch = (char) getch();
        _settextposition(7,74);
        printf("%c",_toupper((int) ch));
        if ((ch == 'y') || (ch == 'Y'))
        {
              *slossflg = 1;

              _settextposition(8,40);
              printf("ENTER IN FILENAME  :  ");
              scanf("%s",filenamel);
              if ((fpl = fopen(filenamel,"w")) == NULL)
              {
                    _clearscreen(_GCLEARSCREEN);
                    _setcolor(19);
                    printf("\nCANNOT OPEN FILE\n");
                    exit(1);
              }
        }
```



```
        _settextposition(7,40);
        printf("                                    ");
        _settextposition(8,40);
        printf("                                     ");

        ch = 'N';
  }

  if (*sflg == 0)
  {
        _settextposition(7,40);
        printf("DO YOU WISH TO STORE EDGE DATA :  %c",ch);
        ch = (char) getch();
        _settextposition(7,74);
        printf("%c",_toupper((int)ch));
        if ((ch == 'y') || (ch == 'Y'))
        {
              pt1 = m;  /* pointer to edge buffer*/
              pt2 = fbuff;  /* pointer to filter buffer*/
              *sflg = 1;  /* saveflag true*/
              *sps = ptindx;  /* set postion in circular buffer from to save*/
              *spnt = 256;  /* amount of points stored */

              _settextposition(8,40);
              printf("ENTER IN FILENAME  :  ");
              scanf("%s",filename);

              if ((fp = fopen(filename,"w")) == NULL)
              {
                    _clearscreen(_GCLEARSCREEN);
                    _setcolor(19);
                    printf("\nCANNOT OPEN FILE\n");
                    exit(1);
              }
              for (c = 0;c <= 255;c++)  /*store edge and filter outputs*/
              {
                    ptindx--;
                    fprintf(fp,"%d %d\n",*(pt1 + ptindx),*(pt2 + ptindx));
              }
              _settextposition(2,60);
              printf("POINTS STORED :  %d ",*spnt);
        }
        _settextposition(7,40);
        printf("                                    ");
        _settextposition(8,40);
        printf("                                     ");
  }
}




/*********************************************************************
  *  PROCEDURE : update_data()                                        *
  *                                                                   *
  *  PURPOSE :  Updates file already opened with edge and filtered    *
  *             edge locations                                        *
  *                                                                   *
  *  INPUT :  m      Edge location buffer                             *
```

```
  *           fbuff  Filtered edge location buffer                  *
  *           spnt   number of points stored to date                *
  *                                                                 *
  *  OUTPUT : spnt   number of points stored to date                *
  *                                                                 *
  *  RETURNS : None                                                 *
  *                                                                 *
  ******************************************************************/

void update_data(m,fbuff,spnt)
unsigned int m[],fbuff[];
int *spnt;
{
  unsigned int *pt1,*pt2,c;

  pt1 = m;
  pt2 = fbuff;
  *spnt += 256;

  _settextposition(7,40);
  printf("UPDATING STORED EDGE DATA ");
  _settextposition(8,40);
  printf("EDGE DATA FILENAME  : %s",filename);

  for (c = 0;c <= 255;c++)
  {
        ptindx--;
        fprintf(fp,"%d %d\n",*(pt1 + ptindx),*(pt2 + ptindx));
  }
  _settextposition(7,40);
  printf("                                    ");
  _settextposition(8,40);
  printf("                                     ");
  _settextposition(2,60);
  printf("POINTS STORED : %d ",*spnt);
}



/******************************************************************
  *  PROCEDURE : update_graph()                                    *
  *                                                                *
  *  PURPOSE : Toggles snap flag to draw graph or not              *
  *            NOTE  if snap = 0 framegrab returns imediatly       *
  *            therefore a delay is  needed after the draw_graph   *
  *            routine in order to ensure fgrab is complete (0.06 s)*
  *                                                                *
  *  INPUT : snp    Snap flag                                      *
  *                   0 : return  immediately                      *
  *                   1 : waits until framegrab is complete        *
  *          X1,X2  Width of rectangle                             *
  *                                                                *
  *  OUTPUT :  snp    Snap flag                                    *
  *                                                                *
  *  RETURNS : None                                                *
  *                                                                *
  ******************************************************************/
```



```
void update_graph(snp,X1,X2)
int *snp,X1,X2;
{
  if (*snp == 0)
  {
        _setviewport(0,140,(X2 - X1) + 80,480);
        _clearscreen(_GVIEWPORT);
        *snp = 1;
  }
  else
  {
        initialize_graph(X1,X2);
        *snp = 0;
  }
}



/******************************************************************
  *  PROCEDURE : option_window()                                   *
  *                                                                *
  *  PURPOSE :  Displays the function keys and there purpose       *
  *             to the screen                                      *
  *                                                                *
  *  INPUT :   None                                                *
  *                                                                *
  *  OUTPUT : None                                                 *
  *                                                                *
  *  RETURNS :  ch the character key pressed                       *
  *                                                                *
  ******************************************************************/

char option_window(void)
{
  char ch;

  _settextwindow(1,1,9,45);
  _clearscreen(_GWINDOW);
  _settextposition(1,1);

  printf(" PRESS  ANY KEY TO RETURN      \n");
  printf("         B   TO TOGGLE DRAWING INDEX    \n");
  printf("         F   TO RESET FILTER PARAMETERS  \n");
  printf("         G   TO TOGGLE GRAPH ON/OFF  \n");
  printf("         I   TO RESTART INITIALIZATION  \n");
  printf("         S   TO SAVE GRAPH/LOSS DATA      \n");
  printf("         W   FOR SPRAYING LOSS      \n");
  printf("         L   FOR FILTER LOSS      \n");
  printf("         T   FOR TOTAL LOSS      \n");

  ch = (char) getch();

  _clearscreen(_GWINDOW);
  printf(" PRESS <ESC> TO EXIT PROGRAM     \n");
  printf("         O   FOR OPTION WINDOW    \n");

  _settextposition(8,2);
  printf("POSTION OF THE BLADE   :    ");
  _settextposition(9,2);
```

```
  printf("TIME FOR UPDATE        :  ");
  _settextwindow(1,1,30,80);
  return (ch);
}



/*************************************************************************
  *  PROCEDURE : wave_error()                                            *
  *                                                                      *
  *  PURPOSE : Finds the percentage of black material lost and           *
  *            white material gained due to spraying in the spiral       *
  *                                                                      *
  *  INPUT :  X1,X2  Rectangle position                                  *
  *           ed          edge location                                  *
  *           grlevel  grey level of ore                                 *
  *           rav      right ave value of diff of ave output             *
  *           bloss    percentage of black material lossed               *
  *           wloss    percentage of white material gained               *
  *           lossflg  pointer to circular buffer                        *
  *           slossflg save flag for losses                              *
  *                                                                      *
  *  OUTPUT : bloss    percentage of black material lossed               *
  *           wloss    percentage of white material gained               *
  *           lossflg  pointer to circular buffer                        *
  *                                                                      *
  *  RETURNS : none                                                      *
  *                                                                      *
  ************************************************************************/

void wave_error(X1,X2,ed,grlevel,rav,bloss,wloss,lossflg,slossflg)
int X1,X2,ed,*lossflg,slossflg;
unsigned int *grlevel;
long int *rav;
float *bloss,*wloss;
{
  int j,start,end,tempold,tempnew,toplevel,lowlevel,bubthresh,startlevel;
  long int Lareaw,Lareab,Rarea,diffnew;
  float tempb,tempw;

  startlevel = 7;  /*offset to start of black material*/
  bubthresh = 25;  /* thershold for detection of buubles*/

  for (j = ed;j < X2 - 1;j++)
  {
       diffnew = rav [j] - rav [j - 1];
       if (diffnew <= 0)  /*find maximum of right average value*/
            break;
  }
  end = j;
  toplevel = (int) (rav [j] / 1000);  /*top level grey value*/

  tempold = 10000;
  for (j = X1;j < ed;j++)
  {
       tempnew = grlevel [j];
       if (tempnew <= tempold)  /* find lowest value for start location*/
```



```
            tempold = tempnew,start = j;
  }
  start += startlevel;
  lowlevel = grlevel [start];  /*lower grey level*/

  for (j = start + 1;j < end;j++)
  {
       diffnew = abs((grlevel [j] - grlevel [j - 1]));
       if (diffnew >= bubthresh)  /* check for bubble from start to end*/
            return;
  }

  Lareaw = 0;
  Lareab = 0;
  for (j = start;j < ed;j++)
  {
       tempnew = ((int) grlevel [j] - lowlevel);
       if (tempnew >= 0)
            Lareaw = Lareaw + tempnew;  /* calculate white contamination*/
       tempnew = (toplevel - (int) grlevel [j]);
       if (tempnew >= 0)
            Lareab = Lareab + tempnew;  /* calculate black gain*/
  }

  Rarea = 0;
  for (j = ed;j <= end;j++)
  {
       tempnew = (toplevel - (int) grlevel [j]);
       if (tempnew >= 0)
            Rarea = Rarea + tempnew;  /*calculate black loss*/
  }

  if (Lareaw <= 0)  /*divide by zero will cause crash*/
  {
       _settextposition(6,2);
       printf("      SPRAYING LOSS");
       _settextposition(7,2);
       printf("BLACK LOSS             : 0 %%             ");
       _settextposition(8,2);
       printf("WHITE GAINED           : 0 %%             ");
       return;
  }

  if (Lareab <= 0)
  {
       _settextposition(6,2);
       printf("      SPRAYING LOSS");
       _settextposition(7,2);
       printf("BLACK LOSS             : 0 %%             ");
       _settextposition(8,2);
       printf("WHITE GAINED           : 0 %%             ");
       return;
  }

  if (Rarea <= 0)
  {
       _settextposition(6,2);
       printf("      SPRAYING LOSS");
       _settextposition(7,2);
       printf("BLACK LOSS             : 0 %%             ");
```

```
        _settextposition(8,2);
        printf("WHITE GAINED          : 0 %%              ");
        return;
  }

  bloss [*lossflg] = ((float) Rarea / ((float) Rarea + (float) Lareab)) * 100;
  wloss [*lossflg] = ((float) Lareaw / ((float) Lareaw + (float) Lareab)) * 100
  ;  /* percentage of black loss and white contamination*/

  if (*lossflg < 29)
        *lossflg = *lossflg + 1;  /*lossflag to circular buffer of size 30*/
  else
        *lossflg = 0;

  tempb = (float) 0.0;
  tempw = (float) 0.0;
  for (j = 0;j < 30;j++)
  {
        tempb += bloss [j];
        tempw += wloss [j];
  }

  tempb = tempb / 30.0;  /* average percentage of black loss*/
  tempw = tempw / 30.0;  /* average percentage of white contamination*/

  _settextposition(6,2);
  printf("      SPRAYING LOSS");
  _settextposition(7,2);
  printf("BLACK LOSS            : %-4.2f %%           ",tempb);
  _settextposition(8,2);
  printf("WHITE GAINED          : %-4.2f %%           ",tempw);

  if (slossflg == 1)
        fprintf(fpl,"%4.2f %4.2f\n",tempb,tempw);
}



/*********************************************************************
  *  PROCEDURE : filter_error()                                       *
  *                                                                   *
  *  PURPOSE : Finds the percentage of black material lost and        *
  *            white material gained due to filter lag in the spiral *
  *                                                                   *
  *  INPUT :  m  edge position buffer                                 *
  *           fbuff filter position buffer                            *
  *           X1  Rectangle start position                            *
  *           ed  edge position                                       *
  *           grlevel  grey  level profile of ore                     *
  *           flossflg filter loss flag must be 40 new filter outputs*
  *                    before losses are calculated                   *
  *           slossflg save flag                                      *
  *                                                                   *
  *  OUTPUT :  none                                                   *
  *                                                                   *
  *  RETURNS : none                                                   *
  *                                                                   *
  *********************************************************************/
```



```
void filter_error(m,fbuff,X1,ed,grlevel,flossflg,slossflg)
int X1,ed,flossflg,slossflg;
unsigned int *grlevel;
unsigned int m[],fbuff[];
{

  int j,start,tempold,tempnew,startlevel;
  long int btotal,wtotal,bloss;
  float tempb,tempw;
  unsigned char a;

  if (flossflg <= 40)
  {
        _settextposition(6,2);
        printf("FILTER LOSS INITIALIZATION");
        _settextposition(7,2);
        printf("BLACK LOSS            : 0 %%              ");
        _settextposition(8,2);
        printf("WHITE GAINED          : 0 %%              ");
        return;
  }

  startlevel = 7;

  tempold = 10000;
  for (j = X1;j < ed;j++)
  {
        tempnew = grlevel [j];
        if (tempnew <= tempold)
              tempold = tempnew,start = j;
  }
  start += startlevel;

  btotal = 0;
  wtotal = 0;
  bloss = 0;
  a = ptindx;

  for (j = 0;j < 40;j++)
  {
        btotal = btotal + (*(fbuff + a) - start);  /* total material gained*/
        tempnew = *(m + a) - *(fbuff + a);
        if (tempnew > 0)
              bloss = bloss + tempnew;  /* black lossed*/
        if (tempnew < 0)
              wtotal = wtotal + abs(tempnew);  /*white contamination*/
        a++;
  }

  if (btotal <= 0)
  {
        _settextposition(6,2);
        printf("      FILTER LOSS              ");
        _settextposition(7,2);
        printf("BLACK LOSS            : 0 %%              ");
        _settextposition(8,2);
        printf("WHITE GAINED          : 0 %%              ");
        return;
  }
```

```
  if (wtotal <= 0)
  {
        _settextposition(6,2);
        printf("      FILTER LOSS            ");
        _settextposition(7,2);
        printf("BLACK LOSS             : 0 %%             ");
        _settextposition(8,2);
        printf("WHITE GAINED           : 0 %%             ");
        return;
  }

  if (bloss <= 0)
  {
        _settextposition(6,2);
        printf("      FILTER LOSS                ");
        _settextposition(7,2);
        printf("BLACK LOSS             : 0 %%             ");
        _settextposition(8,2);
        printf("WHITE GAINED           : 0 %%             ");
        return;
  }

  tempb = ((float) bloss / ((float) (btotal - wtotal) + (float) bloss)) * 100;
  tempw = ((float) wtotal / (float) btotal) * 100;

  _settextposition(6,2);
  printf("      FILTER LOSS                   ");
  _settextposition(7,2);
  printf("BLACK LOSS             : %-4.2f %%             ",tempb);
  _settextposition(8,2);
  printf("WHITE GAINED           : %-4.2f %%             ",tempw);

  if (slossflg == 1)
        fprintf(fpl,"%4.2f %4.2f\n",tempb,tempw);
}




/*********************************************************************
  *  PROCEDURE : total_error()                                        *
  *                                                                   *
  *  PURPOSE : Finds the percentage of black material lost and        *
  *            white material gained due to spraying in the spiral    *
  *                                                                   *
  *  INPUT :  X1,X2 Rectangle position                                *
  *           ed    edge location                                     *
  *           grlevel ore grey level profile                          *
  *           rav  rght average value od DOA                          *
  *           fltv  filter output at present sampling instant         *
  *           blk  buffer of black material gained                    *
  *           wht  buffer of white material contamination             *
  *           blkloss buffer of black material lossed                 *
  *           lossflg pointer to these buffers                        *
  *           slossflg save flag                                      *
  *                                                                   *
  *  OUTPUT : blk  buffer input of black material gained              *
```



```
  *           wht  buffer input of white material contamination       *
  *           blkloss buffer input of black material lossed           *
  *           lossflg incremented pointer to these buffers            *
  *                                                                   *
  *  RETURNS : none                                                   *
  *                                                                   *
  *********************************************************************/

void total_error(X1,X2,ed,grlevel,rav,fltv,blk,wht,blkloss,lossflg,slossflg)
int *lossflg,slossflg;
long int *blk,*wht,*blkloss;
int X1,X2,ed,fltv;
unsigned int *grlevel;
long int *rav;
{

  int j,start,end,tempold,tempnew,toplevel,lowlevel,bubthresh,startlevel;
  long int Lareaw,Lareab,Rarea,diffnew;
  long int tempb,tempw,tempbl;
  float tmpb,tmpw;

  startlevel = 7;
  bubthresh = 25;

  for (j = ed;j < X2 - 1;j++)
  {
        diffnew = rav [j] - rav [j - 1];
        if (diffnew <= 0)
              break;
  }
  end = j;
  toplevel = (int) (rav [j] / 1000);

  tempold = 10000;
  for (j = X1;j < ed;j++)
  {
        tempnew = grlevel [j];
        if (tempnew <= tempold)
              tempold = tempnew,start = j;
  }
  start += startlevel;
  lowlevel = grlevel [start];

  for (j = start + 1;j < end;j++)
  {
        diffnew = abs((grlevel [j] - grlevel [j - 1]));
        if (diffnew >= bubthresh)
              return;
  }

  Lareaw = 0;
  Lareab = 0;
  Rarea = 0;

  for (j = start;j < fltv;j++)
  {
        tempnew = ((int) grlevel [j] - lowlevel);
        if (tempnew >= 0)
              Lareaw = Lareaw + tempnew;
        tempnew = (toplevel - (int) grlevel [j]);
```

```
      if (tempnew >= 0)
            Lareab = Lareab + tempnew;
}
while (fltv <= end)
{
      tempnew = (toplevel - (int) grlevel [fltv]);
      if (tempnew >= 0)
            Rarea = Rarea + tempnew;
      fltv++;
}

blk [*lossflg] = Lareab;
wht [*lossflg] = Lareaw;
blkloss [*lossflg] = Rarea;

if (*lossflg < 29)
      *lossflg = *lossflg + 1;
else
      *lossflg = 0;

tempb = 0;
tempw = 0;
tempbl = 0;

for (j = 0;j < 30;j++)
{
      tempb += blk [j];
      tempw += wht [j];
      tempbl += blkloss [j];
}

if (tempw <= 0)
{
      _settextposition(6,2);
      printf("      TOTAL LOSS            ");
      _settextposition(7,2);
      printf("BLACK LOSS            : 0 %%              ");
      _settextposition(8,2);
      printf("WHITE GAINED          : 0 %%              ");
      return;
}

if (tempb <= 0)
{
      _settextposition(6,2);
      printf("      TOTAL LOSS                ");
      _settextposition(7,2);
      printf("BLACK LOSS            : 0 %%              ");
      _settextposition(8,2);
      printf("WHITE GAINED          : 0 %%              ");
      return;
}

tmpb = ((float) tempbl / ((float) tempbl + (float) tempb)) * 100;
tmpw = ((float) tempw / ((float) tempw + (float) tempb)) * 100;

_settextposition(6,2);
printf("      TOTAL LOSS");
_settextposition(7,2);
printf("BLACK LOSS            : %-4.2f %%           ",tmpb);
```



```
  _settextposition(8,2);
  printf("WHITE GAINED          : %-4.2f %%           ",tmpw);

  if (slossflg == 1)
        fprintf(fpl,"%4.2f %4.2f\n",tmpb,tmpw);
}
```

APPENDIX C

Users Guide to

Program : AOFIC4C.C

Description

This program was used as an aid for the development of EDGE9GC.C. The program shows the grey level profile of the ore in the AOI window, as well as the determined edge position. Also shown is the determined UPPER and LOWER levels and the percentage values for the black loss and the white contamination.

Hardware Required

IBM computer, preferably an AT or higher with a VGA monitor. A MATROX PIP 1024B or 512B Video Digitizer Board.

Operation

To start program type AOFIC4C [Enter].

First an image must be loaded into the frame buffer of the frame grabber. The program will first ask DO YOU WISH TO LOAD FILE FROM DISK, if Y then ENTER IN FILENAME. This will load an image from a 262144 byte array to the frame buffer of the frame grabber. If N is specified, an image can be obtained from a video input source. By pressing any key, the program will invoke a snapshot of the input video source. Once an image has been obtained the next stage is the initialization of the AOI window.

Initialization

To start the user must enter in the coordinates of the AOI window. This is done by entering in the coordinates of the top left hand corner, denoted by X1, Y1 and the bottom right hand corner, denoted by X2, Y2. Due to the transfer capability of the frame grabber, the AOI window cannot have a height greater than 100 pixel rows. The program will prompt the user to enter in the values for X1, Y1, X2 and Y2 respectively. The default values for X1, Y1, X2 and Y2 are 80, 315, 380 and 325 respectively. These values can be enter by simply pressing [Enter] at the appropriate prompt. Once this has been done it will ask if these values are correct, the default is Y. If Y is specified, the program will proceed to the next stage of initialization else

if N is specified the program will return to the beginning of the initialization so that the coordinate values can be retyped.

The second stage of initialization is the fine movement of the AOI window. The AOI can now be seen on the frame grabber output monitor. The following options are available to the user :

Arrow Keys

By pressing one of the arrow keys, the AOI rectangle will move in the direction specified by which arrow key was pressed.

L

L will lock the top left hand corner of the AOI window, and if pressed again it will unlock. The status of the point X1, Y1 can be seen in the middle of the screen to be either FREE or LOCKED.

Grey -

Grey - will select fine movement of the AOI window, which will move 1 pixel at a time in the direction of the selected arrow key. The movement status can be seen in the middle of the screen to be STEP SIZE = 1.

Grey +

Grey + will select coarse movement of the AOI window, which will move 20 pixels at a time in the direction of the selected arrow key. The movement status can be seen in the middle of the screen to be STEP SIZE = 20.

B

B will enable the drawing index (grey level) of the AOI rectangle to be changed. Once selected the current drawing index can be seen at the top right hand corner of the screen, DRAW INDEX = 255. By pressing the grey - or grey +, the drawing index can be decreased or increased between the values of 0 and 255. Pressing B again will exit the drawing index initialization.

Once the AOI window is in the correct position, the initialization routine can be exited by pressing the [Esc] key.

The program has now completed the initialization and the following output screen can be seen.

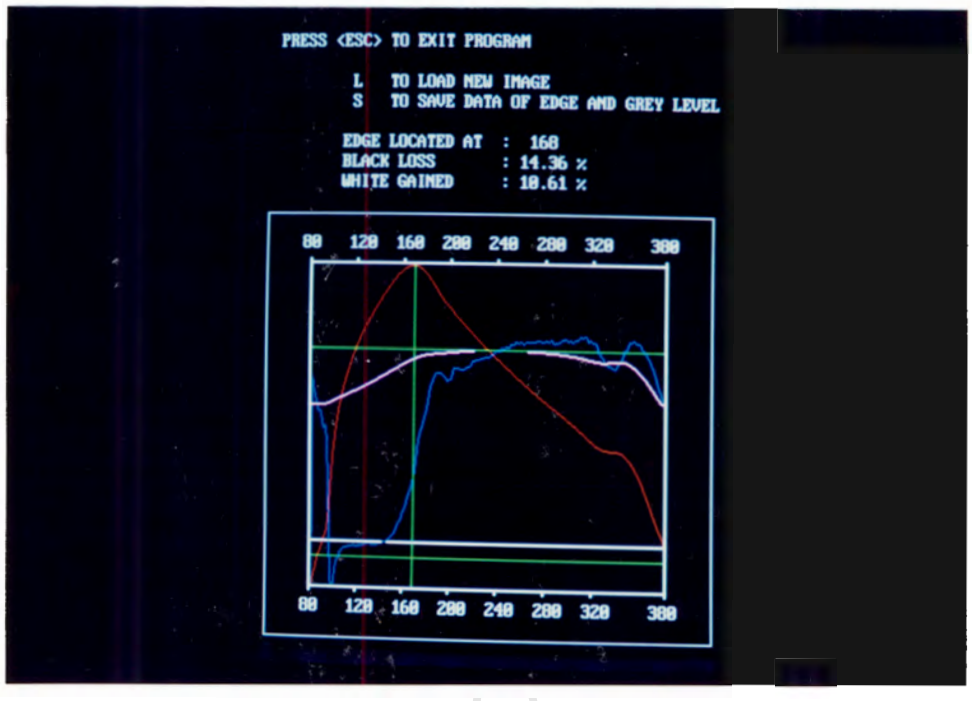


FIGURE C.1 : Main output screen of program AOFIC4C.C

A graph box will be seen who's height ranges from 0 to 255 and width ranges from X1 to X2. The following traces are shown :

1. The blue trace is the grey level profile of the ore in the AOI window.
2. The red trace is the DOA edge detector output that has been scaled to be within the range of 0 to 255.
3. The horizontal white line is the zero line of the DOA edge detector output. The DOA output is negative below this line and positive above this line.
3. The vertical green line is the determined blade position which is at the peak of the DOA output.
4. The purple line is the right average value (RAV) of the DOA edge detector and is used to determine the UPPER level.
5. The two horizontal green lines are the determined UPPER and LOWER levels that are used together with the edge position to calculate the percentage losses.

Shown at the top of the graph is the edge location, the black loss percentage and the white contamination percentage. The percentage losses are calculated from the equations (7.4) and (7.5). The frame grabber output monitor will show the AOI rectangle with the determined edge position. By pressing any key, the program will return to the second stage of initialization thus allowing the AOI rectangle to be repositioned.

The following options are available to the user :

L

L will load a new image. The program will ask DO YOU WISH TO LOAD FILE FROM DISK, if Y then ENTER IN FILENAME. This will load an image from a 262144 byte array to the frame buffer of the frame grabber. If N is specified, an image can be obtained from a video source. By pressing any key the program will take a snapshot of the input video source. Once a new image has been obtained the program will recommence at the second stage of the initialization.

S

S will save the data shown by the graph. In the middle right hand screen the program will ask DO YOU WISH TO STORE EDGE DATA, if Y it will ask for a filename, ENTER IN FILENAME. The specified file will be a text file with 4 columns. Column 1 will be the right average value of the DOA edge detector. Column 2 will be the left average value of the DOA edge detector. Column 3 will be the grey level profile of the ore in the AOI window and column 4 will contain the DOA edge detector output which is the difference between column 1 and column 2. All four columns are stored between the values of X1 and X2. The file is closed after saving is complete.

To quit the program press [Esc] from the main output screen of figure C.1.

```
/*****************************************************************************/
/*  PROGRAM AOFIC4C.C TO CALCULATE BLADE POSITION IN A FILE OR VIDEO INPUT */
/*  ========================================================================= */
/*                                                                           */
/*       AUTHOR : DAVID GOLD                                                 */
/*       DATE   : 14 FEB 1991                                                */
/*****************************************************************************/


/*************************************************************
  *                  Header Files                             *
  *************************************************************/

#include <stdio.h>
#include <ctype.h>
#include <math.h>
#include <conio.h>
#include <float.h>
#include <graph.h>
#include <stdlib.h>
#include <time.h>

#define LOC(i,j,cols) (i * cols + j)


/*************************************************************
  *                  Function Declarations                    *
  *************************************************************/
void initialize_fgrab(void);
void initialize_variables(int *X1,int *X2,int *Y1,int *Y2);
void initialize_graph(int X1,int X2);
void initialize_edgebuff(void);
void get_coord(int *X1,int *X2,int *Y1,int *Y2);
void move_rect(int *X1,int *X2,int *Y1,int *Y2);
void draw_index(int X1,int X2,int Y1,int Y2);
void draw_graph();
void load_file(void);
void save_file(int X1,int X2,int Y1,int Y2);
void wave_error(int X1,int X2,int Y1,int Y2,int ed);
void erralloc(void);
int find_edge(int X1,int X2,int Y1,int Y2,unsigned char matrix[],float *maxv,
              float *minv);
int get_num(int a);
int flag(int a);


/*************************************************************
  *                  Global Variables                         *
  *************************************************************/
unsigned char *indx,ptindx;
int cnt,grval[512];
float edgebuff[512],avel[512],aver[512];
char filename[25];


/*************************************************************
  *                  Start of Main Program                    *
  *************************************************************/
main()
{
```



```
  int x1,x2,y1,y2,edge;
  char ch;
  float maxval,minval;

  indx = (unsigned char *) calloc(20 * 512,sizeof(unsigned char));
  if (indx == NULL)
        erralloc();

  initialize_fgrab();
  load_file();
  initialize_variables(&x1,&x2,&y1,&y2);
  get_coord(&x1,&x2,&y1,&y2);
  fg_btrans(0,y1 - 1,((y2 - y1) + 2) * 512,0,1,0,indx,-1);

  do
  {
        fg_btrans(0,y1 - 1,((y2 - y1) + 2) * 512,1,1,0,indx,-1);
        move_rect(&x1,&x2,&y1,&y2);
        initialize_graph(x1,x2);
        initialize_edgebuff();
        fg_btrans(0,y1 - 1,((y2 - y1) + 2) * 512,0,1,0,indx,-1);
        edge = (find_edge(x1,x2,y1,y2,indx,&maxval,&minval));
        draw_graph(indx,x1,x2,y1,y2,maxval,minval,edge);
        fg_rect(x1,y1,x2,y2);
        fg_rect(x1,y1,edge,y2);
        wave_error(x1,x2,y1,y2,edge);
        _settextposition(8,2);

        for (;;)
        {
              if (kbhit())
                    break;
        }
        ch = (char) getch();
        if ((ch == 'l') || (ch == 'L'))
        {
              fg_btrans(0,y1 - 1,((y2 - y1) + 2) * 512,1,1,0,indx,-1);
              load_file();
              fg_btrans(0,y1 - 1,((y2 - y1) + 2) * 512,0,1,0,indx,-1);
        }
        if ((ch == 's') || (ch == 'S'))
              save_file(x1,x2,y1,y2);
  }
  while (ch != 27);
  free(indx);
  _setvideomode(_DEFAULTMODE);
}



/*****************************************************************************
  *  PROCEDURE : draw_graph()                                                 *
  *                                                                           *
  *  PURPOSE : Sets viewport and draws graph of edge location,                *
  *             verticlaly averaged grey value, and edge detector             *
  *              output                                                       *
  *                                                                           *
  *  INPUT :   matrix Edge location buffer                                    *
  *            X1,X2,Y1,Y2      AOI rectangle                               *
```

```
  *            maxv Maximum value of edge detector                  *
  *            minv Minimum value of edge detector                  *
  *            edge   Edge location                                 *
  *                                                                 *
  *  OUTPUT : None                                                  *
  *                                                                 *
  *  RETURNS : None                                                 *
  *                                                                 *
  ******************************************************************/

void draw_graph(matrix,X1,X2,Y1,Y2,maxv,minv,edge)
float maxv,minv;
int X1,X2,Y1,Y2,edge;
unsigned char matrix[];
{
  int indx,xval,yval,numy,col,row;
  float sum,x;
  float scale,sc;

  xval = 0;

  if (minv > 0)
  {
        sc = maxv;
        minv = (float) 0.0;
  }
  else
        sc = maxv - minv;  /* sc is scaling factor */

  if (sc == 0)
        sc = 1;
  scale = (float) 25.5 / sc;
  numy = Y2 - Y1;
  if (numy == 0)
        numy = 1;

  yval = 255 - (int) (fabs(minv) * scale * 10.0);
  _setcolor(15);
  _moveto(0,yval);
  _lineto(X2 - X1,yval);
  _setcolor(4);
  _moveto(0,255);

  for (indx = X1;indx <= X2 - 1;indx++)
  {
        xval++;
        yval = 255 - (int) ((edgebuff [indx] - minv) * scale * 10);
        _lineto(xval,yval);
  }

  _setcolor(1);
  _moveto(0,255);
  xval = 0;

  for (col = X1;col < X2;col++)
  {
        sum = 0;
        x = 0;
        xval++;
```



```
        for (row = 0;row < numy;row++)
        {
              x++;
              sum += matrix [LOC(row,col,512)];  /* LOC = (i*cols + j) */
        }
        grval [col] = (int) (sum / x);  /* average grey level of ore profile*/
        yval = 255 - grval [col];
        _lineto(xval,yval);
  }


  yval = 255 - (int) aver [X1];
  _setcolor(13);
  _moveto(0,yval);
  xval = 0;
  for (col = X1;col < X2 - 1;col++)
  {
        xval++;
        yval = 255 - (int) aver [col];
        _lineto(xval,yval);
  }


  _setcolor(2);
  _moveto(edge - X1,0);
  _lineto(edge - X1,255);
  _settextposition(6,9);
  printf("EDGE LOCATED AT  :  %d",edge);
}




/*******************************************************************
  *  PROCEDURE : find_edge()                                        *
  *                                                                 *
  *  PURPOSE : Detects the position of the edge by the use of       *
  *            diference of averages                                *
  *                                                                 *
  *  INPUT :  X1,X2,Y1,Y2  Rectangle position                       *
  *           matrix       Base address of 1-D matrix of data       *
  *                        transfered from framegraber              *
  *                                                                 *
  *  OUTPUT :  maxv Maximum value of edge detector                  *
  *            minv Minimum value of edge detector                  *
  *                                                                 *
  *  RETURNS : ed Integer edge position                             *
  *                                                                 *
  *******************************************************************/

int find_edge(X1,X2,Y1,Y2,matrix,maxv,minv)
int X1,X2,Y1,Y2;
unsigned char matrix[];
float *maxv,*minv;
{
  int a,i,j,ed,ny;
  unsigned long int total,coltot;
  float Rave,Lave;
```

```
  long double ltot,rtot,numx,numy,numtot,x;

  total = 0;
  *maxv = (float) -100000.0;
  *minv = (float) 100000.0;
  ed = 0;
  numx = X2 - X1;
  numy = Y2 - Y1;

  if (numx == 0)
        numx = 1;
  if (numy == 0)
        numy = 1;
  numtot = numx * numy;
  ny = Y2 - Y1;
  if (ny == 0)
        ny = 1;


  for (i = 0;i < ny;i++)
        for (j = X1;j < X2;j++)
              total += matrix [LOC(i,j,512)];  /* LOC = (i*cols + j) */

  ltot = 0;
  rtot = total;
  a = 0;

  for (j = X1;j < X2 - 1;j++)
  {
        coltot = 0;
        a = a + 1;

        for (i = 0;i < ny;i++)
              coltot = coltot + matrix [LOC(i,j,512)];
        ltot = ltot + coltot;
        rtot = rtot - coltot;


        x = numtot - a * numy;
        if (x == 0)
              x = 1;
        Rave = rtot / x;
        Lave = ltot / (a * numy);
        avel [j] = Lave;
        aver [j] = Rave;
        edgebuff [j] = Rave - Lave;
        if (edgebuff [j] <= *minv)
              *minv = edgebuff [j];
        if (edgebuff [j] >= *maxv)
              *maxv = edgebuff [j],ed = j;
  }
  if (*minv == 0)
        *minv = 1;
  if (*maxv == 0)
        *maxv = 1;
  ed = ed + 1;
  return (ed);
}
```



```
/*********************************************************************
  *  PROCEDURE : wave_error()                                         *
  *                                                                   *
  *  PURPOSE : Finds the percentage of black material lost and        *
  *            white material gained due to spraying in the spiral    *
  *                                                                   *
  *  INPUT :  X1,X2,Y1,Y2  Rectangle position                         *
  *           ed           edge location                             *
  *                                                                   *
  *                                                                   *
  *  OUTPUT :  none                                                   *
  *                                                                   *
  *  RETURNS : none                                                   *
  *                                                                   *
  *********************************************************************/

void wave_error(X1,X2,Y1,Y2,ed)
int X1,X2,Y1,Y2,ed;
{
  int j,start,end,tempold,tempnew,toplevel,lowlevel;
  long int Lareaw,Lareab,Rarea;
  float diffold,diffnew,blacklost,whitegained;

  diffold = 10000.0;
  for (j = ed;j < X2;j++)
  {
        diffnew = aver [j] - aver [j - 1];
        if (diffnew <= 0.0)
              break;
        diffold = diffnew;
  }
  end = j;
  toplevel = (int) aver [j];


  tempold = 10000;
  for (j = X1;j < ed;j++)
  {
        tempnew = grval [j];
        if (tempnew <= tempold)
              tempold = tempnew,start = j;
  }
  start += 5;
  lowlevel = grval [start];

  Lareaw = 0;
  Lareab = 0;
  for (j = start;j < ed;j++)
  {
        Lareaw += grval [j] - lowlevel;
        Lareab += toplevel - grval [j];
  }


  Rarea = 0;
  for (j = ed;j <= end;j++)
        Rarea += toplevel - grval [j];
```

```
  blacklost = ((float) Rarea / (float) (Rarea + (float) Lareab)) * 100;
  whitegained = ((float) Lareaw / ((float) Lareaw + (float) Lareab)) * 100;

  printf("\n        BLACK LOSS       : %4.2f %% ",blacklost);
  printf("\n        WHITE GAINED     : %4.2f %% ",whitegained);

  _setcolor(2);
  _moveto(0,255 - toplevel);
  _lineto(X2 - X1,255 - toplevel);
  _moveto(0,255 - lowlevel);
  _lineto(X2 - X1,255 - lowlevel);
}



/*********************************************************************
  *  PROCEDURE :  get_coord()                                         *
  *                                                                   *
  *  PURPOSE :    Reads in from the keyboard the values for the       *
  *               area of interest rectangle (X1,X2,Y1,Y2)            *
  *                                                                   *
  *  INPUT : X1,X2,Y1,Y2  : Initialized rectangle position            *
  *                                                                   *
  *  OUTPUT : X1,X2,Y1,Y2  : Input rectangle position                 *
  *                                                                   *
  *  RETURNS :  None                                                  *
  *                                                                   *
  *********************************************************************/

void get_coord(X1,X2,Y1,Y2)
int *X1,*X2,*Y1,*Y2;
{
  int ch;

  _clearscreen(_GCLEARSCREEN);
  _settextposition(1,1);
  printf(" ENTER IN THE COORDINATES OF X1,Y1 AND X2,Y2 OF THE          \n");
  printf(" AREA OF INTREST   Note: The amount of pixel rows        \n");
  printf("                         may not exeed 88              \n");

  do
  {
        _settextposition(11,30);
        printf("X1 = %d",*X1);
        ch = getch();
        if (ch != 13)
        {
              _settextposition(11,35);
              printf("    ");
              _settextposition(11,35);
              *X1 = get_num(ch);
        }

        _settextposition(12,30);
        printf("Y1 = %d",*Y1);
        ch = getch();
        if (ch != 13)
        {
              _settextposition(12,35);
```



```
              printf("    ");
              _settextposition(12,35);
              *Y1 = get_num(ch);
        }

        _settextposition(14,30);
        printf("X2 = %d",*X2);
        ch = getch();
        if (ch != 13)
        {
              _settextposition(14,35);
              printf("    ");
              _settextposition(14,35);
              *X2 = get_num(ch);
        }

        _settextposition(15,30);
        printf("Y2 = %d",*Y2);
        ch = getch();
        if (ch != 13)
        {
              _settextposition(15,35);
              printf("    ");
              _settextposition(15,35);
              *Y2 = get_num(ch);
        }
        ch = 'Y';
        printf("\n\n\n\n\nARE THESE VALUE CORRECT : %c",ch);
        ch = getch();
        _settextposition(20,27);
        printf("%c",_toupper(ch));
  }
  while ((ch == 'n') || (ch == 'N'));
}



/*********************************************************************
  *  PROCEDURE :  move_rect()                                         *
  *                                                                   *
  *  PURPOSE : Moves the area of interest rectngle around by using    *
  *            the display functions of the Framegrabber.  Control    *
  *            keys are :                                             *
  *                     1. Arrow keys  for movement of rectangle      *
  *                     2. L places the top left hand corner of rect  *
  *                     3. -/+ coarce or fine movement                *
  *                     4. B -/+ drawing index up or down             *
  *                     5. <esc> to exit routine                      *
  *                                                                   *
  *  INPUT :  X1,X2,Y1,Y2  : Inputed rectangle position               *
  *                                                                   *
  *  OUTPUT : X1,X2,Y1,Y2  : Input rectangle position                 *
  *                                                                   *
  *  RETURNS : None                                                   *
  *                                                                   *
  *********************************************************************/

void move_rect(X1,X2,Y1,Y2)
int *X1,*X2,*Y1,*Y2;
```

```
{
  char ch;
  int step,flg;

  step = 1;
  flg = 0;

  _clearscreen(_GCLEARSCREEN);
  _settextposition(1,1);
  printf(" PRESS <ESC> TO EXIT INTIALIZATION                          \n");
  printf("                                                            \n");
  printf("                                                           \n");
  printf("                  moves rectangle with or without corner placed\n");
  printf("        ↓                                                   \n");
  printf("                                                            \n");
  printf("        L         places corner X1 Y1                       \n");
  printf("        -         sets fine movement                       \n");
  printf("        +         sets coarse movement                     \n");
  printf("    B  -/+        sets drawing index                       \n");
  _settextposition(4,11);
  putch(26);
  _settextposition(3,9);
  putch(24);
  _settextposition(4,7);
  putch(27);

  _settextposition(14,30);
  printf("STEP SIZE = %d",step);
  _settextposition(15,30);
  if (flg == 1)
        printf("X1,Y1 : LOCKED");
  else
        printf("X1,Y1 : FREE");

  _settextposition(17,30);
  printf("X1 = %d",*X1);
  _settextposition(18,30);
  printf("Y1 = %d",*Y1);
  _settextposition(20,30);
  printf("X2 = %d",*X2);
  _settextposition(21,30);
  printf("Y2 = %d",*Y2);

  fg_btrans(0,*Y1 - 1,((*Y2 - *Y1) + 2) * 512,0,1,0,indx,-1);

  do
  {
        if (kbhit())
        {
              ch = (char) getch();
              if ((ch == 'l') || (ch == 'L'))
              {
                    flg = flag(flg);
                    _settextposition(15,30);
                    if (flg == 1)
                          printf("X1,Y1 : LOCKED  ");
                    else
                          printf("X1,Y1 : FREE    ");
              }
              if ((ch == 'b') || (ch == 'B'))
```



```
                    draw_index(*X1,*X2,*Y1,*Y2);
              fg_btrans(0,*Y1 - 1,((*Y2 - *Y1) + 2) * 512,1,1,0,indx,-1);
              switch (ch)
              {
                    case 43:
                          step = 20;
                          _settextposition(14,42);
                          printf("%d ",step);
                          break;
                    case 45:
                          step = 1;
                          _settextposition(14,42);
                          printf("%d ",step);
                          break;
                    case 75:
                          if (flg == 0)
                          {
                                *X1 -= step;
                                *X2 -= step;
                                _settextposition(17,35);
                                printf("%d ",*X1);
                                _settextposition(20,35);
                                printf("%d ",*X2);
                          }
                          else
                          {
                                *X2 -= step;
                                if (*X2 < *X1)
                                      *X2 = *X1;
                                _settextposition(20,35);
                                printf("%d ",*X2);
                          }
                          break;
                    case 77:
                          if (flg == 0)
                          {
                                *X1 += step;
                                *X2 += step;
                                _settextposition(17,35);
                                printf("%d ",*X1);
                                _settextposition(20,35);
                                printf("%d ",*X2);
                          }
                          else
                          {
                                *X2 += step;
                                _settextposition(20,35);
                                printf("%d ",*X2);
                          }
                          break;
                    case 80:
                          if (flg == 0)
                          {
                                *Y1 += step;
                                *Y2 += step;
                                _settextposition(18,35);
                                printf("%d ",*Y1);
                                _settextposition(21,35);
                                printf("%d ",*Y2);
                          }
```

```
                        else
                        {
                              *Y2 += step;
                              if (*Y2 - *Y1 >= 88)
                                    *Y2 = *Y1 + 88;
                              _settextposition(21,35);
                              printf("%d ",*Y2);
                        }
                        break;
                  case 72:
                        if (flg == 0)
                        {
                              *Y1 -= step;
                              *Y2 -= step;
                              _settextposition(18,35);
                              printf("%d ",*Y1);
                              _settextposition(21,35);
                              printf("%d ",*Y2);
                        }
                        else
                        {
                              *Y2 -= step;
                              if (*Y2 < *Y1)
                                    *Y2 = *Y1;
                              _settextposition(21,35);
                              printf("%d ",*Y2);
                        }
                        break;
            }
            fg_btrans(0,*Y1 - 1,((*Y2 - *Y1) + 2) * 512,0,1,0,indx,-1);
            fg_rect(*X1,*Y1,*X2,*Y2);
      }
  }
  while (ch != 27);
  fg_btrans(0,*Y1 - 1,((*Y2 - *Y1) + 2) * 512,1,1,0,indx,-1);
}



/*********************************************************************
  *  PROCEDURE : initialize_variables()                               *
  *                                                                   *
  *  PURPOSE :  initializes all the variables local to main()         *
  *                                                                   *
  *  INPUT / OUTPUT :                                                 *
  *       X1,X2,Y1,Y2  : Rectangle position                           *
  *       cnt : drawing index                                         *
  *        ptindx  : pointer to circular buffer                       *
  *                                                                   *
  *  RETURNS :  None                                                  *
  *                                                                   *
  *********************************************************************/

void initialize_variables(X1,X2,Y1,Y2)
int *X1,*X2,*Y1,*Y2;
{
  *X1 = 80;
  *X2 = 380;
```



```
  *Y1 = 316;
  *Y2 = 326;

  cnt = 250;
  ptindx = 0;
}



/*********************************************************************
  *  PROCEDURE : initialize_fgrab()                                   *
  *                                                                   *
  *  PURPOSE :  Initializes the framegraber                           *
  *                                                                   *
  *  INPUT :  None                                                    *
  *                                                                   *
  *  OUTPUT : None                                                    *
  *                                                                   *
  *  RETURNS : None                                                   *
  *                                                                   *
  *********************************************************************/

void initialize_fgrab(void)
{
  fg_inifmt(0x26c,1,0,0,1,0);  /*iniatialize the frame grabber*/
  fg_chan(2);
  fg_sync(1);
  fg_setind(255);
  fg_winmode(0);
  fg_quadm(1);
  fg_dquad(0);
  fg_sbuf(0);
}



/*********************************************************************
  *  PROCEDURE : initialize_graph()                                   *
  *                                                                   *
  *  PURPOSE : Draws and labels graph axis                            *
  *                                                                   *
  *  INPUT : X1,X2 width of rectangle/graph window                    *
  *                                                                   *
  *  OUTPUT : None                                                    *
  *                                                                   *
  *  RETURNS : None                                                   *
  *                                                                   *
  *********************************************************************/

void initialize_graph(X1,X2)
int X1,X2;
{
  int c,text,x;

  _setvideomode(_VRES16COLOR);
```

```
  _setcolor(7);
  _setviewport(0,144,(X2 - X1) + 80,479);
  _rectangle(_GBORDER,0,0,(X2 - X1) + 80,335);
  _setcolor(15);
  _rectangle(_GBORDER,39,39,(X2 - X1) + 41,296);

  c = 39;
  text = 5;
  x = X1;

  do
  {
        _moveto(c,39);
        _lineto(c,36);
        _settextposition(11,text);
        printf("%d",x);
        _moveto(c,296);
        _lineto(c,299);
        _settextposition(29,text);
        printf("%d",x);

        text += 5;
        c += 40;
        x += 40;
  }
  while (c <= (X2 - X1) - 1);

  _moveto((X2 - X1) + 41,39);
  _lineto((X2 - X1) + 41,36);
  _settextposition(11,((X2 - X1) + 41) / 8);
  printf("%d",X2);
  _moveto((X2 - X1) + 41,296);
  _lineto((X2 - X1) + 41,299);
  _settextposition(29,((X2 - X1) + 41) / 8);
  printf("%d",X2);

  _setviewport(40,184,(X2 - X1) + 40,440);

  _settextposition(1,1);
  printf(" PRESS <ESC> TO EXIT PROGRAM     \n\n");
  printf("         L   TO LOAD NEW IMAGE  \n");
  printf("         S   TO SAVE DATA OF EDGE AND GREY LEVEL    \n");
}



/*********************************************************************
  *  PROCEDURE :  initialize_edgebuff()                               *
  *                                                                   *
  *  PURPOSE : Initializes buffers for edge, verticle averaged grey   *
  *            value and difference of averages output                *
  *                                                                   *
  *  INPUT :   None                                                   *
  *                                                                   *
  *  OUTPUT :  None                                                   *
  *                                                                   *
  *  RETURNS : None                                                   *
  *                                                                   *
```



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

void initialize_edgebuff(void)
{
  int c;

  for (c = 0;c < 512;c++)
  {
        edgebuff [c] = 0.0;
        grval [c] = 0.0;
        avel [c] = 0.0;
        aver [c] = 0.0;
  }
}




/*********************************************************************
  *  PROCEDURE : get_num()                                            *
  *                                                                   *
  *  PURPOSE : Reads in a character from keyboard checks for valid    *
  *            integer and appends it to the number to be returned    *
  *                                                                   *
  *  INPUT : a First digit in the number                              *
  *                                                                   *
  *  OUTPUT : None                                                    *
  *                                                                   *
  *  RETURNS : num The integer number scaned from the keyboard        *
  *                                                                   *
  *********************************************************************/

int get_num(a)
int a;
{
  int num,b;

  num = 0;
  b = a;
  a = 0;

  for (;;)
  {
        if (b == 8)
        {
              printf("\b \b");
              num = num / 10;
        }
        if ((isdigit(b)) || (b == 13))
        {
              switch (b)
              {
                    case '0':
                          a = 0;
                          break;
                    case '1':
                          a = 1;
                          break;
```

```
                case '2':
                        a = 2;
                        break;
                case '3':
                        a = 3;
                        break;
                case '4':
                        a = 4;
                        break;
                case '5':
                        a = 5;
                        break;
                case '6':
                        a = 6;
                        break;
                case '7':
                        a = 7;
                        break;
                case '8':
                        a = 8;
                        break;
                case '9':
                        a = 9;
                        break;
                case 13:
                        return (num);
                default:
                        a = 0;
            }
            if (num <= 3200)
            {
                printf("%d",a);
                num = num * 10 + a;
            }
        }
        b = getch();
    }
}



/*********************************************************************
  *  PROCEDURE : flag()                                               *
  *                                                                   *
  *  PURPOSE : Toggles the value of a flag                            *
  *                                                                   *
  *  INPUT : a Flag value 0 or 1                                      *
  *                                                                   *
  *  OUTPUT : None                                                    *
  *                                                                   *
  *  RETURNS : Flag value 0 or 1                                      *
  *                                                                   *
  *********************************************************************/

int flag(a)
int a;
{
  switch (a)
```



```
  {
        case 0:
                a = 1;
                break;
        case 1:
                a = 0;
                break;
  }
  return (a);
}



/*********************************************************************
  *  PROCEDURE : draw_index()                                         *
  *                                                                   *
  *  PURPOSE : Sets drawing index for the framegraber                 *
  *                                                                   *
  *  INPUT : X1,X2,Y1,Y2  Rectangle position                          *
  *                                                                   *
  *  OUTPUT :  None                                                   *
  *                                                                   *
  *  RETURNS :  None                                                  *
  *                                                                   *
  *********************************************************************/

void draw_index(X1,X2,Y1,Y2)
int X1,X2,Y1,Y2;
{
  char ch;

  _settextposition(1,60);
  printf("DRAW INDEX = %d  ",cnt);

  do
  {
        if (kbhit())
        {
                ch = (char) getch();
                fg_btrans(0,Y1 - 1,((Y2 - Y1) + 2) * 512,1,1,0,indx,-1);
                switch (ch)
                {
                        case 43:
                                cnt += 5;
                                if (cnt > 255)
                                        cnt = 255;
                                break;
                        case 45:
                                cnt -= 5;
                                if (cnt < 0)
                                        cnt = 0;
                                break;
                }
                fg_btrans(0,Y1 - 1,((Y2 - Y1) + 2) * 512,0,1,0,indx,-1);
        }
        fg_setind(cnt);
        fg_rect(X1,Y1,X2,Y2);
```

```
        _settextposition(1,60);
        printf("DRAW INDEX = %d  ",cnt);
    }
    while ((ch != 'b') && (ch != 'B'));
    _settextposition(1,60);
    printf("                ");
    fg_btrans(0,Y1 - 1,((Y2 - Y1) + 2) * 512,1,1,0,indx,-1);
}
```

```
/*********************************************************************
  *  PROCEDURE :  erralloc()                                          *
  *                                                                   *
  *  PURPOSE : Exit to DOS on error                                   *
  *                                                                   *
  *  INPUT : None                                                     *
  *                                                                   *
  *  OUTPUT : None                                                    *
  *                                                                   *
  *  RETURNS : None                                                   *
  *                                                                   *
  *********************************************************************/

void erralloc(void)
{
  _setvideomode(_DEFAULTMODE);
  _settextposition(12,20);
  _settextcolor(31);
  _outtext("Unable to allocate space for array");
  exit(1);
}
```

```
/*********************************************************************
  *  PROCEDURE :  load_file()                                         *
  *                                                                   *
  *  PURPOSE : Loads a stored image into the framegrabers frame       *
  *            buffer or an image from a live video input on          *
  *            channel 2                                              *
  *                                                                   *
  *  INPUT : None                                                     *
  *                                                                   *
  *  OUTPUT : None                                                    *
  *                                                                   *
  *  RETURNS : None                                                   *
  *                                                                   *
  *********************************************************************/

void load_file(void)
{
  char ch;

  ch = 'N';
```



```
  _clearscreen(_GCLEARSCREEN);
  _settextposition(1,1);
  printf("DO YOU WISH TO LOAD FILE FROM DISK :  %c",ch);
  ch = getch();
  _settextposition(1,35);
  printf("%c",_toupper(ch));
  if ((ch == 'y') || (ch == 'Y'))
  {
        _clearscreen(_GCLEARSCREEN);
        _settextposition(1,1);
        printf("\n ENTER FILE NAME : ");
        scanf("%s",filename);
        fg_sbuf(1);
        fg_sync(0);
        fg_frdisk(1000,0,filename,0x5200,-1);
  }
  else
  {
        _clearscreen(_GCLEARSCREEN);
        _settextposition(1,1);
        printf("\n HIT ANY KEY TO FREEZE FRAME ");
        fg_sync(1);
        fg_sbuf(0);
        getch();
        fg_snap(1);
        fg_sbuf(1);
        fg_sync(0);
  }
}
```

```
/*********************************************************************
  *  PROCEDURE :  save_file()                                         *
  *                                                                   *
  *  PURPOSE : Saves the verticly averaged edge profile,edge detector*
  *            output, right and left average values of DOA output    *
  *                                                                   *
  *  INPUT : X1,X2,Y1,Y2  Rectangle position                          *
  *                                                                   *
  *  OUTPUT : None                                                    *
  *                                                                   *
  *  RETURNS : None                                                   *
  *                                                                   *
  *********************************************************************/

void save_file(X1,X2,Y1,Y2)
int X1,X2,Y1,Y2;
{
  int c;
  char ch;
  FILE *fp;

  ch = 'N';

  _clearscreen(_GCLEARSCREEN);
  _settextposition(1,1);
  printf("DO YOU WISH TO STORE EDGE DATA :  %c",ch);
  ch = (char) getch();
```

```
  _settextposition(1,35);
  printf("%c",_toupper(ch));
  if ((ch == 'y') || (ch == 'Y'))
  {
        printf("\n\nENTER IN FILENAME  :  ");
        scanf("%s",filename);

        if ((fp = fopen(filename,"w")) == NULL)
        {
              _clearscreen(_GCLEARSCREEN);
              _setcolor(19);
              printf("\nCANNOT OPEN FILE\n");
              exit(1);
        }

        for (c = X1;c < X2;c++)
              fprintf(fp,"%15.10f %15.10f %d %15.10f\n",aver [c],avel [c],grval
                    [c],edgebuff [c]);
        fclose(fp);
  }
}
```

APPENDIX D

Users Guide to

Program : EDGETRUE.C

Description

This program traces out the true edge position in 1 video frame. This is done within a specified window of set height, with the sampling interval in this window being adjustable. The process can be repeated for successive frames thus building up a plot of the true edge position curve from pixel point to pixel point. This curve can be saved to a text file.

Hardware Required

IBM computer, preferably an AT or higher with a VGA monitor. A MATROX PIP 1024B or 512B Video Digitizer Board.

Operation

To start program type EDGETRUE [Enter]. The first stage of the program is the initialization of the Area Of Interest (AOI) window.

Initialization

To start the user must enter in the coordinates of the AOI window. This is done by entering in the coordinates of the top left hand corner, denoted by X1, Y1 and the bottom right hand corner, denoted by X2, Y2. Due to the transfer capability of the frame grabber, the AOI window cannot have a height greater than 100 pixel rows. The program will prompt the user to enter in the values for X1, Y1, X2 and Y2 respectively. The default values for X1, Y1, X2 and Y2 are 80, 315, 380 and 325 respectively. These values can be enter by simply pressing [Enter] at the appropriate prompt. Once this has been done it will ask if these values are correct, the default is Y. If Y is specified, the program will proceed to the next stage of initialization else if N is specified the program will return to the beginning of the initialization so that the coordinate values can be retyped.

The second stage of initialization is the fine movement of the AOI window. The AOI can now be seen on the frame grabber output monitor. The following options are available to the user :

Arrow Keys

By pressing one of the arrow keys, the AOI rectangle will move in the direction specified by which arrow key was pressed.

L

L will lock the top left hand corner of the AOI window, and if pressed again it will unlock. The status of the point X1, Y1 can be seen in the middle of the screen to be either FREE or LOCKED.

Grey -

Grey - will select fine movement of the AOI window, which will move 1 pixel at a time in the direction of the selected arrow key. The movement status can be seen in the middle of the screen to be STEP SIZE = 1.

Grey +

Grey + will select coarse movement of the AOI window, which will move 20 pixels at a time in the direction of the selected arrow key. The movement status can be seen in the middle of the screen to be STEP SIZE = 20.

B

B will enable the drawing index (grey level) of the AOI rectangle to be changed. Once selected the current drawing index can be seen at the top right hand corner of the screen, DRAW INDEX = 255. By pressing the grey - or grey +, the drawing index can be decreased or increased between the values of 0 and 255. Pressing B again will exit the drawing index initialization.

Once the AOI window is in the correct position, the initialization routine can be exited by pressing the [Esc] key.

The third stage of the initialization is the selection of the window height through which the DOA edge detector is to move through. The program will ask ENTER IN THE HEIGHT OF EDGE TO SCAN, the default value is 56 and is the pixel distance from the top to the bottom of the window which the DOA edge detector is to move through frame by frame. Next the sampling interval or point must be selected, ENTER IN SAMPLE POINT, the default value is 2. This specifies that the edge position is to be determined every second point or row as the DOA edge detector moves down through the 56 point window.

The program has now completed the initialization and the following output screen should be seen.

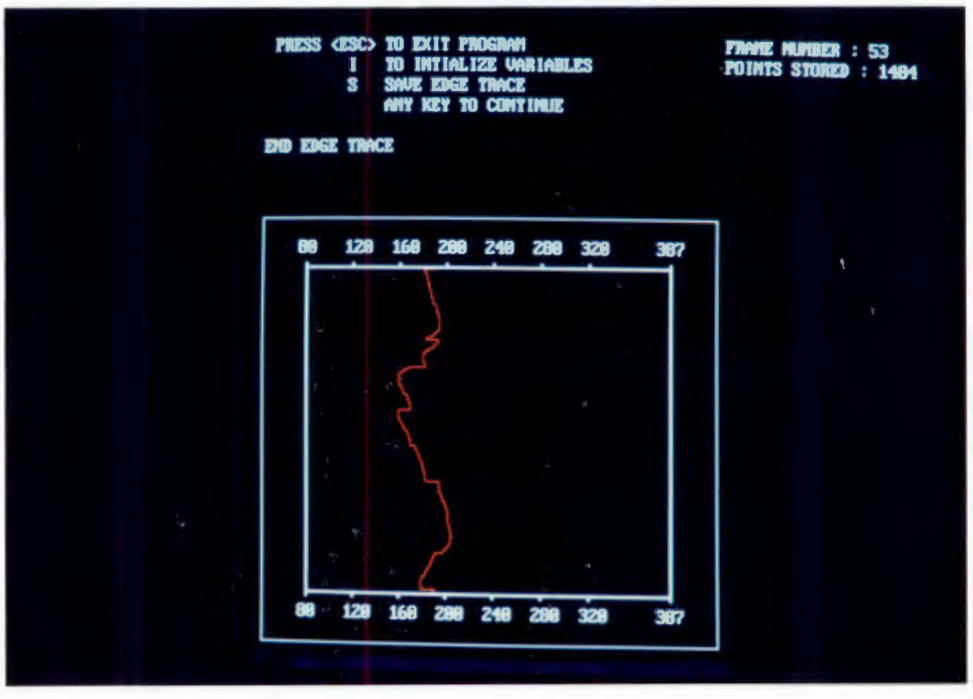


FIGURE D.1 : Main output screen of program EDGETRUE.C

A graph box will be seen who's height ranges from 0 to 255 and width ranges from X1 to X2. The trace of the true edge will be seen by the red curve. To build up a trace of the true edge curve, video frames must be stepped through with the pause facility of the video recorder. Once a new frame is shown on the output monitor of the frame grabber any key can be pressed for the program to trace out the true edge position within the specified window of that frame. The number of points that the red curve is updated by, is (height / sample point). The status of the edge trace is shown at the top of the graph by START EDGE TRACE at the beginning of the trace and END EDGE TRACE at the end of the edge trace. The determined true edge position is also shown on the output monitor of the frame grabber. Once the edge trace has ended the video recorder is stepped to the next frame. Any key is pressed for the program to begin a new trace, thus building up a plot of the true edge position curve.

The following options are available to the user :

I

I will return the program to stage two of the initialization.

S

S will save the true edge trace. After S has been typed the program will ask DO YOU WISH TO STORE EDGE DATA. If Y is selected, the filename needs to entered, ENTER IN FILENAME. Once this has been done the true edge will be saved to a text file. This file will only have one column which will be the true edge curve. At the top right hand corner of the display, it will show what the current frame number is and underneath it will show how many points have been stored. These numbers are updated from frame to frame. The file is closed if the I is pressed.

To quit the program press [Esc] from the main output screen of figure D.1.

```
/*****************************************************************/
/*  PROGRAM EDGETRUE.C TO CALCULATE TRUE EDGE POSITION LOCATION  */
/*  ==========================================================   */
/*                                                               */
/*       AUTHOR : DAVID GOLD                                     */
/*       DATE   : 1 March 1991                                   */
/*****************************************************************/


/*********************************************************
  *                  Header Files                         *
  *********************************************************/

#include <stdio.h>
#include <ctype.h>
#include <math.h>
#include <conio.h>
#include <float.h>
#include <graph.h>
#include <stdlib.h>
#include <time.h>
#include <malloc.h>
#include <ctype.h>

#define LOC(i,j,cols) (i * cols + j)


/*********************************************************
  *                Function Declarations                  *
  *********************************************************/

void initialize_fgrab(void);
void initialize_variables(int *X1,int *X2,int *Y1,int *Y2,
                          int *hght,int *sflg,int *smple);
void get_coord(int *X1,int *X2,int *Y1,int *Y2);
void move_rect(int *X1,int *X2,int *Y1,int *Y2);
void draw_index(int X1,int X2,int Y1,int Y2);
void initialize_circbuff(unsigned int m[],int X1,int X2);
void initialize_graph(int X1,int X2);
void find_edge(int X1,int X2,int Y1,int Y2,unsigned char far *matrix,
               int hght,unsigned int m[],int sflg,int *frm,int *pnts,int smple)
;
void draw_graph(unsigned int m[],int X1,int X2);
void save_data(int *sflg,int *frm,int *pnts);
void erralloc(void);
int get_height(int hght);
int get_sample(int smple);
int get_num(int a);
int flag(int a);

/*********************************************************
  *                Global Variables                       *
  *********************************************************/

int cnt;
unsigned char ptindx;
FILE *fp;
char filename[25];
```



```
/*********************************************************
  *                Start of Main Program                  *
  *********************************************************/
main()
{
  int x1,x2,y1,y2,height,sflag,frame,points,sample;
  unsigned int sz,circbuff[256];
  unsigned char far *indx;
  char ch;

  initialize_fgrab();
  initialize_variables(&x1,&x2,&y1,&y2,&height,&sflag,&sample);
  get_coord(&x1,&x2,&y1,&y2);
  _clearscreen(_GCLEARSCREEN);

a:
  ch = 0;
  move_rect(&x1,&x2,&y1,&y2);
  height = (get_height(height));
  sample = (get_sample(sample));
  initialize_circbuff(circbuff,x1,x2);
  initialize_graph(x1,x2);

  sz = (height + (y2 - y1)) * 512;
  if (sz == 0)
       sz = 512;
  indx = (unsigned char far *) _fmalloc(sz * sizeof(unsigned char));
  if (indx == NULL)
       erralloc();


  ch = 0;
  ptindx = 0;


  _settextposition(1,1);
  printf(" PRESS <ESC> TO EXIT PROGRAM     \n");
  printf("          I   TO INTIALIZE VARIABLES \n");
  printf("          S   SAVE EDGE TRACE \n");
  printf("              ANY KEY TO CONTINUE");

  for (;;)
  {
       fg_snap(1);
       fg_btrans(0,y1,sz,0,1,1,indx,-1);
       _settextposition(6,1);
       printf("START EDGE TRACE");
       find_edge(x1,x2,y1,y2,indx,height,circbuff,sflag,&frame,&points,sample)
       ;
       draw_graph(circbuff,x1,x2);
       _settextposition(6,1);
       printf("END EDGE TRACE    ");
       fg_sbuf(0);
       ch = (char) getch();
       fg_sbuf(1);
       if ((ch == 27) || (ch == 'i') || (ch == 'I'))
            break;
       if ((ch == 's') || (ch == 'S'))
            save_data(&sflag,&frame,&points);
  }
```

```
  _ffree(indx);
  _clearscreen(_GCLEARSCREEN);
  _setvideomode(_DEFAULTMODE);

  if (sflag)
  {
        sflag = 0;
        fclose(fp);
  }
  if ((ch == 'i') || (ch == 'I'))
        goto a;
}



/*********************************************************************
  *  PROCEDURE :  get_coord()                                         *
  *                                                                   *
  *  PURPOSE :   Reads in from the keyboard the values for the        *
  *              area of interest rectangle (X1,X2,Y1,Y2)             *
  *                                                                   *
  *  INPUT : X1,X2,Y1,Y2  : Initialized rectangle position            *
  *                                                                   *
  *  OUTPUT : X1,X2,Y1,Y2  : Inputed rectangle position               *
  *                                                                   *
  *  RETURNS :  None                                                  *
  *                                                                   *
  *********************************************************************/

void get_coord(X1,X2,Y1,Y2)
int *X1,*X2,*Y1,*Y2;
{
  int ch;

  _clearscreen(_GCLEARSCREEN);
  _settextposition(1,1);
  printf(" ENTER IN THE COORDINATES OF X1,Y1 AND X2,Y2 OF THE        \n");
  printf(" AREA OF INTEREST   Note: The amount of pixel rows          \n");
  printf("                          may not exeed 100                \n");

  do
  {
        _settextposition(11,30);
        printf("X1 = %d",*X1);
        ch = getch();
        if (ch != 13)
        {
              _settextposition(11,35);
              printf("             ");
              _settextposition(11,35);
              *X1 = get_num(ch);
        }
        _settextposition(12,30);
        printf("Y1 = %d",*Y1);
        ch = getch();
        if (ch != 13)
        {
              _settextposition(12,35);
              printf("             ");
              _settextposition(12,35);
              *Y1 = get_num(ch);
        }

        _settextposition(14,30);
        printf("X2 = %d",*X2);
        ch = getch();
        if (ch != 13)
        {
              _settextposition(14,35);
              printf("             ");
              _settextposition(14,35);
              *X2 = get_num(ch);
        }

        _settextposition(15,30);
        printf("Y2 = %d",*Y2);
        ch = getch();
        if (ch != 13)
        {
              _settextposition(15,35);
              printf("             ");
              _settextposition(15,35);
              *Y2 = get_num(ch);
        }
        ch = 'Y';
        printf("\n\n\n\n\nARE THESE VALUE CORRECT : %c",ch);
        ch = getch();
        _settextposition(20,27);
        printf("%c",ch);
  }
  while ((ch == 'n') || (ch == 'N'));
}



/*********************************************************************
  *  PROCEDURE :  move_rect()                                         *
  *                                                                   *
  *  PURPOSE : Moves the area of interest rectngle around by using    *
  *            the display functions of the Framegrabber.  Control    *
  *            keys are :                                             *
  *                     1. Arrow keys  for movement of rectangle      *
  *                     2. L places the top left hand corner of rect  *
  *                     3. -/+ coarce or fine movement                *
  *                     4. B -/+ drawing index up or down             *
  *                     5. <esc> to exit routine                      *
  *                                                                   *
  *  INPUT :  X1,X2,Y1,Y2  : Inputed rectangle position               *
  *                                                                   *
  *  OUTPUT : X1,X2,Y1,Y2  : Inputed rectangle position               *
  *                                                                   *
  *  RETURNS : None                                                   *
  *                                                                   *
  *********************************************************************/

void move_rect(X1,X2,Y1,Y2)
int *X1,*X2,*Y1,*Y2;
```

```
{
  char ch;
  int step,flg;

  step = 1;
  flg = 0;

  _clearscreen(_GCLEARSCREEN);
  _settextposition(1,1);
  printf(" PRESS <ESC> TO EXIT INTIALIZATION                              \n");
  printf("                                                                \n");
  printf("                                                               \n");
  printf("                  moves rectangle with or without corner placed\n");
  printf("        ↓                                                       \n");
  printf("                                                                \n");
  printf("        L         places corner X1 Y1                           \n");
  printf("        -         sets fine movement                           \n");
  printf("        +         sets coarse movement                         \n");
  printf("    B  -/+        sets drawing index                           \n");
  _settextposition(4,11);
  putch(26);
  _settextposition(3,9);
  putch(24);
  _settextposition(4,7);
  putch(27);


  _settextposition(14,30);
  printf("STEP SIZE = %d",step);
  _settextposition(15,30);
  if (flg == 1)
        printf("X1,Y1 : LOCKED");
  else
        printf("X1,Y1 : FREE");
  _settextposition(17,30);
  printf("X1 = %d",*X1);
  _settextposition(18,30);
  printf("Y1 = %d",*Y1);
  _settextposition(20,30);
  printf("X2 = %d",*X2);
  _settextposition(21,30);
  printf("Y2 = %d",*Y2);


  do
  {
        if (kbhit())
        {
              ch = (char) getch();
              if ((ch == 'l') || (ch == 'L'))
              {
                    flg = flag(flg);
                    _settextposition(15,30);
                    if (flg == 1)
                          printf("X1,Y1 : LOCKED  ");
                    else
                          printf("X1,Y1 : FREE    ");
              }
              if ((ch == 'b') || (ch == 'B'))
                    draw_index(*X1,*X2,*Y1,*Y2);
```



```
              switch (ch)
              {
                    case 43:
                          step = 20;
                          _settextposition(14,42);
                          printf("%d ",step);
                          break;
                    case 45:
                          step = 1;
                          _settextposition(14,42);
                          printf("%d ",step);
                          break;
                    case 75:
                          if (flg == 0)
                          {
                                *X1 -= step;
                                *X2 -= step;
                                _settextposition(17,35);
                                printf("%d ",*X1);
                                _settextposition(20,35);
                                printf("%d ",*X2);
                          }
                          else
                          {
                                *X2 -= step;
                                if (*X2 < *X1)
                                      *X2 = *X1;
                                _settextposition(20,35);
                                printf("%d ",*X2);
                          }
                          break;
                    case 77:
                          if (flg == 0)
                          {
                                *X1 += step;
                                *X2 += step;
                                _settextposition(17,35);
                                printf("%d ",*X1);
                                _settextposition(20,35);
                                printf("%d ",*X2);
                          }
                          else
                          {
                                *X2 += step;
                                _settextposition(20,35);
                                printf("%d ",*X2);
                          }
                          break;
                    case 80:
                          if (flg == 0)
                          {
                                *Y1 += step;
                                *Y2 += step;
                                _settextposition(18,35);
                                printf("%d ",*Y1);
                                _settextposition(21,35);
                                printf("%d ",*Y2);
                          }
                          else
                          {
```

```
                        *Y2 += step;
                        if (*Y2 - *Y1 >= 100)
                              *Y2 = *Y1 + 100;
                        _settextposition(21,35);
                        printf("%d ",*Y2);
                  }
                  break;
            case 72:
                  if (flg == 0)
                  {
                        *Y1 -= step;
                        *Y2 -= step;
                        _settextposition(18,35);
                        printf("%d ",*Y1);
                        _settextposition(21,35);
                        printf("%d ",*Y2);
                  }
                  else
                  {
                        *Y2 -= step;
                        if (*Y2 < *Y1)
                              *Y2 = *Y1;
                        _settextposition(21,35);
                        printf("%d ",*Y2);
                  }
                  break;
            }
      }
      fg_snap(1);
      fg_rect(*X1,*Y1,*X2,*Y2);
  }
  while (ch != 27);
}


/*********************************************************************
  *  PROCEDURE : initialize_variables()                               *
  *                                                                   *
  *  PURPOSE :  Initializes variables used by main()                  *
  *                                                                   *
  *  INPUT : X1,X2,Y1,Y2 AOI rectangle position                       *
  *          hght height of window to move through                    *
  *          sflg save edge position flag 0 no 1 yes                  *
  *          smple sample point number                                *
  *                                                                   *
  *  OUTPUT : X1,X2,Y1,Y2 AOI rectangle position                      *
  *           hght height of window to move through                   *
  *           sflg save edge position flag 0 no 1 yes                 *
  *           smple sample point number                               *
  *                                                                   *
  *  RETURNS : None                                                   *
  *                                                                   *
  *********************************************************************/

void initialize_variables(X1,X2,Y1,Y2,hght,sflg,smple)
int *X1,*X2,*Y1,*Y2,*hght,*sflg,*smple;
{
  *X1 = 80;
  *X2 = 387;
```



```
  *Y1 = 284;
  *Y2 = 294;

  *hght = 56;
  *sflg = 0;
  *smple = 2;
}


/*********************************************************************
  *  PROCEDURE : initialize_fgrab()                                   *
  *                                                                   *
  *  PURPOSE :  Initializes the framegraber                           *
  *                                                                   *
  *  INPUT :  None                                                    *
  *                                                                   *
  *  OUTPUT : None                                                    *
  *                                                                   *
  *  RETURNS : None                                                   *
  *                                                                   *
  *********************************************************************/

void initialize_fgrab(void)
{
  fg_inifmt(0x26c,1,0,0,1,0);  /*iniatialize the frame grabber*/
  fg_chan(2);
  fg_sync(1);
  fg_setind(255);
  fg_sbuf(1);
  fg_snap(1);
}


/*********************************************************************
  *  PROCEDURE : get_num()                                            *
  *                                                                   *
  *  PURPOSE : Reads in a character from keyboard checks for valid    *
  *            integer and appends it to the number to be returned    *
  *                                                                   *
  *  INPUT : a First digit in the number                              *
  *                                                                   *
  *  OUTPUT : None                                                    *
  *                                                                   *
  *  RETURNS : num The integer number scaned from the keyboard        *
  *                                                                   *
  *********************************************************************/

int get_num(a)
int a;
{
  int num,b;

  num = 0;
  b = a;
  a = 0;

  for (;;)
  {
```

```
        if (b == 8)
        {
              printf("\b \b");
              num = num / 10;
        }
        if ((isdigit(b)) || (b == 13))
        {
              switch (b)
              {
                    case '0':
                          a = 0;
                          break;
                    case '1':
                          a = 1;
                          break;
                    case '2':
                          a = 2;
                          break;
                    case '3':
                          a = 3;
                          break;
                    case '4':
                          a = 4;
                          break;
                    case '5':
                          a = 5;
                          break;
                    case '6':
                          a = 6;
                          break;
                    case '7':
                          a = 7;
                          break;
                    case '8':
                          a = 8;
                          break;
                    case '9':
                          a = 9;
                          break;
                    case 13:
                          return (num);
                    default:
                          a = 0;
              }
              if (num <= 32000)
              {
                    printf("%d",a);
                    num = num * 10 + a;
              }
        }
        b = getch();
  }
}



/*************************************************************************
  *  PROCEDURE : flag()                                                   *
  *                                                                       *
```



```
  *  PURPOSE : Toggles the value of a flag                                *
  *                                                                       *
  *  INPUT : a Flag value 0 or 1                                          *
  *                                                                       *
  *  OUTPUT : None                                                        *
  *                                                                       *
  *  RETURNS : Flag value 0 or 1                                          *
  *                                                                       *
  *************************************************************************/

int flag(a)
int a;
{
  switch (a)
  {
        case 0:
              a = 1;
              break;
        case 1:
              a = 0;
              break;
  }
  return (a);
}



/*************************************************************************
  *  PROCEDURE : draw_index()                                             *
  *                                                                       *
  *  PURPOSE : Sets drawing index for the framegraber                     *
  *                                                                       *
  *  INPUT : X1,X2,Y1,Y2  Rectangle position                              *
  *                                                                       *
  *  OUTPUT :  None                                                       *
  *                                                                       *
  *  RETURNS :  None                                                      *
  *                                                                       *
  *************************************************************************/

void draw_index(X1,X2,Y1,Y2)
int X1,X2,Y1,Y2;
{
  char ch;

  _settextposition(1,60);
  printf("                ");
  _settextposition(1,60);
  printf("DRAW INDEX = %d  ",cnt);

  do
  {
        if (kbhit())
        {
              ch = (char) getch();
              switch (ch)
              {
                    case 43:
                          cnt += 5;
```

```
                        if (cnt > 255)
                              cnt = 255;
                        break;
                  case 45:
                        cnt -= 5;
                        if (cnt < 0)
                              cnt = 0;
                        break;
                }
        }
        fg_snap(1);
        fg_setind(cnt);
        fg_rect(X1,Y1,X2,Y2);
        _settextposition(1,60);
        printf("DRAW INDEX = %d  ",cnt);
  }
  while ((ch != 'b') && (ch != 'B'));
  _settextposition(1,60);
  printf("                 ");
}



/*************************************************************************
  *  PROCEDURE :  erralloc()                                              *
  *                                                                       *
  *  PURPOSE : Exit to DOS on error                                       *
  *                                                                       *
  *  INPUT : None                                                         *
  *                                                                       *
  *  OUTPUT : None                                                        *
  *                                                                       *
  *  RETURNS : None                                                       *
  *                                                                       *
  *************************************************************************/

void erralloc(void)
{
  printf("\nUnable to allocate space for array\n");
  printf("Memory Availabe %u\n",_memavl());
  exit(1);
}



/*************************************************************************
  *  PROCEDURE :  initialize_circbuff()                                   *
  *                                                                       *
  *  PURPOSE : Initializes buffers for edge location                      *
  *                                                                       *
  *  INPUT :   X1,X2  Width of rectangle/graph window                     *
  *                                                                       *
  *  OUTPUT :  m  Edge location buffer                                    *
  *                                                                       *
  *  RETURNS : None                                                       *
  *                                                                       *
  *************************************************************************/

void initialize_circbuff(m,X1,X2)
```



```
int X1,X2;
unsigned int m[];
{
  unsigned int c,cst;

  cst = X1 + ((X2 - X1) / 2);

  for (c = 0;c <= 255;c++)
        m [c] = cst;
}



/*************************************************************************
  *  PROCEDURE : initialize_graph()                                       *
  *                                                                       *
  *  PURPOSE : Draws and labels graph axis                                *
  *                                                                       *
  *  INPUT : X1,X2 width of rectangle/graph window                        *
  *                                                                       *
  *  OUTPUT : None                                                        *
  *                                                                       *
  *  RETURNS : None                                                       *
  *                                                                       *
  *************************************************************************/
void initialize_graph(X1,X2)
int X1,X2;
{
  int c,text,x;

  _setvideomode(_VRES16COLOR);
  _setcolor(7);
  _setviewport(0,144,(X2 - X1) + 80,479);
  _rectangle(_GBORDER,0,0,(X2 - X1) + 80,335);
  _setcolor(15);
  _rectangle(_GBORDER,39,39,(X2 - X1) + 41,296);

  c = 39;
  text = 5;
  x = X1;

  do
  {
        _moveto(c,39);
        _lineto(c,36);
        _settextposition(11,text);
        printf("%d",x);
        _moveto(c,296);
        _lineto(c,299);
        _settextposition(29,text);
        printf("%d",x);

        text += 5;
        c += 40;
        x += 40;
  }
  while (c <= (X2 - X1) - 1);

  _moveto((X2 - X1) + 41,39);
  _lineto((X2 - X1) + 41,36);
```

```
  _settextposition(11,((X2 - X1) + 41) / 8);
  printf("%d",X2);
  _moveto((X2 - X1) + 41,296);
  _lineto((X2 - X1) + 41,299);
  _settextposition(29,((X2 - X1) + 41) / 8);
  printf("%d",X2);

  _setviewport(40,184,(X2 - X1) + 40,439);
}


/*********************************************************************
  *  PROCEDURE : get_height()                                         *
  *                                                                   *
  *  PURPOSE : gets the height of the widow through which the AOI     *
  *            rectangle moves through to calculate the edge position*
  *            at every smple point                                   *
  *                                                                   *
  *  INPUT :  hght height of window                                   *
  *                                                                   *
  *  OUTPUT : None                                                    *
  *                                                                   *
  *  RETURNS : hght height of window                                  *
  *                                                                   *
  *********************************************************************/

int get_height(hght)
int hght;
{
  char ch;

  _clearscreen(_GCLEARSCREEN);
  _settextposition(1,1);
  printf("ENTER IN THE HEIGHT OF EDGE TO SCAN  :  %d",hght);
  ch = (char) getch();
  if (ch != 13)
  {
        _settextposition(1,40);
        printf("            ");
        _settextposition(1,41);
        hght = get_num(ch);
  }

  return (hght);
}



/*********************************************************************
  *  PROCEDURE : get_sample()                                         *
  *                                                                   *
  *  PURPOSE : get the value for the distance between sample points   *
  *            in determining the true edge location                  *
  *                                                                   *
  *  INPUT :  smple sample point number                               *
  *                                                                   *
  *  OUTPUT : None                                                    *
  *                                                                   *
  *  RETURNS :  smple sample point number                             *
```



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

int get_sample(smple)
int smple;
{
  char ch;

  _settextposition(3,1);
  printf("ENTER IN SAMPLE POINT  :  %d",smple);
  ch = (char) getch();
  if (ch != 13)
  {
        _settextposition(3,25);
        printf("            ");
        _settextposition(3,26);
        smple = get_num(ch);
  }

  return (smple);
}



/*********************************************************************
  *  PROCEDURE : draw_graph()                                         *
  *                                                                   *
  *  PURPOSE : Sets viewport and draws graph of true edge location    *
  *                                                                   *
  *  INPUT :   m Edge location buffer                                 *
  *            X1,X2,Y1,Y2      AOI rectangle                         *
  *                                                                   *
  *  OUTPUT : None                                                    *
  *                                                                   *
  *  RETURNS : None                                                   *
  *                                                                   *
  *********************************************************************/

void draw_graph(m,X1,X2)
unsigned int m[];
int X1,X2;
{
  int indx,xvalin;
  unsigned int *pt;

  pt = m;

  _clearscreen(_GVIEWPORT);

  xvalin = *(pt + ptindx) - X1;
  _setcolor(4);
  _setpixel(xvalin,0);

  for (indx = 0;indx <= 255;indx++)
  {
        ptindx++;
        _moveto(xvalin,indx);
        xvalin = *(pt + ptindx) - X1;
        _lineto(xvalin,indx);
```

```
  }
}



/*********************************************************************
  *  PROCEDURE : find_edge()                                          *
  *                                                                   *
  *  PURPOSE : Detects the position of the edge by the use of         *
  *            diference of averages.  Edge is found by runing the    *
  *            AOI window down through the height of hgt and finding  *
  *            the edge position every smple point                    *
  *                                                                   *
  *  INPUT :  X1,X2,Y1,Y2  Rectangle position                         *
  *           matrix       Base address of 1-D matrix of data         *
  *                        transfered from framegraber                *
  *           hght height of window through which edge is detected    *
  *           m  buffer storing edge position through the height hght *
  *           sflg  save flag 0 no 1 yes                              *
  *           frm   frame number stored                               *
  *           pnts  point number stored                               *
  *           smple  sampling point interval                          *
  *                                                                   *
  *  OUTPUT : frm   frame number stored                               *
  *           pnts  point number stored                               *
  *                                                                   *
  *  RETURNS : None                                                   *
  *                                                                   *
  *********************************************************************/

void find_edge(X1,X2,Y1,Y2,matrix,hght,m,sflg,frm,pnts,smple)
int X1,X2,Y1,Y2,hght,smple,sflg,*frm,*pnts;
unsigned char far *matrix;
unsigned int m[];
{
  int a,i,j,ed,numx,numy,c,starty,endy,sm;
  unsigned long int total,coltot,numtot,ltot,rtot,x;
  long int temp,Lave,Rave,maxval;
  unsigned char tempindx;

  sm = hght / smple;
  starty = 0;
  endy = Y2 - Y1;
  tempindx = ptindx;
  ptindx = ptindx - sm + 1;


  for (c = 0;c < sm;c++)
  {
        total = 0;
        maxval = -100000;
        ed = 0;
        numx = X2 - X1;
        numy = Y2 - Y1;

        if (numx == 0)
              numx = 1;
        if (numy == 0)
              numy = 1;
```



```
        numtot = numx * numy;


        for (i = starty;i < endy;i++)
              for (j = X1;j < X2;j++)
                    total = total + matrix [LOC(i,j,512)];  /* LOC = (i*cols +
                                                               j) */

        ltot = 0;
        rtot = total;
        a = 0;

        for (j = X1;j < X2 - 1;j++)
        {
              coltot = 0;
              a += 1;

              for (i = starty;i < endy;i++)
                    coltot = coltot + matrix [LOC(i,j,512)];


              ltot = ltot + coltot;
              rtot = rtot - coltot;


              x = numtot - a * numy;
              if (x == 0)
                    x = 1;
              Rave = (rtot * 1000) / x;
              Lave = (ltot * 1000) / (a * numy);
              temp = Rave - Lave;
              if (temp >= maxval)
                    maxval = temp,ed = j;
        }
        if (X1 == 0)
              ed += 1;

        fg_pixw(ed,(starty + Y1) + numy / 2,255);
        starty = starty + smple;
        endy = endy + smple;
        *(m + ptindx) = ed;
        ptindx++;
  }

  ptindx = tempindx - sm;
  if (sflg)
  {
        for (c = 0;c < sm;c++)
        {
              fprintf(fp,"%d \n",*(m + tempindx));
              tempindx--;
        }
        *pnts = *pnts + sm;
        *frm = *frm + 1;
        _settextposition(1,50);
        printf("FRAME NUMBER : %d  ",*frm);
        _settextposition(2,50);
        printf("POINTS STORED : %d   ",*pnts);
  }
}
```

```
/*********************************************************************
  *  PROCEDURE :  save_data()                                         *
  *                                                                   *
  *  PURPOSE : Saves the true edge location to a file                 *
  *                                                                   *
  *  INPUT :   sflg save flag 0 not to save 1 to save                 *
  *            frm  number of frames stored                           *
  *            pnts  number of points  stored                         *
  *                                                                   *
  *  OUTPUT :  sflg save flag 0 not to save 1 to save                 *
  *            frm  number of frames stored                           *
  *            pnts  number of points  stored                         *
  *                                                                   *
  *  RETURNS : None                                                   *
  *                                                                   *
  *********************************************************************/

void save_data(sflg,frm,pnts)
int *sflg,*frm,*pnts;
{

  char ch;

  ch = 'N';

  _settextposition(1,40);
  printf("DO YOU WISH TO STORE EDGE DATA :  %c",ch);
  ch = (char) getch();
  _settextposition(1,74);
  printf("%c",_toupper(ch));
  if ((ch == 'y') || (ch == 'Y'))
  {
        *sflg = 1;
        *frm = 0;
        *pnts = 0;

        _settextposition(2,40);
        printf("ENTER IN FILENAME  :  ");
        scanf("%s",filename);

        if ((fp = fopen(filename,"w")) == NULL)
        {
              _clearscreen(_GCLEARSCREEN);
              _setcolor(19);
              printf("\nCANNOT OPEN FILE\n");
              exit(1);
        }
  }
  _settextposition(1,40);
  printf("                                  ");
  _settextposition(2,40);
  printf("                                   ");
}
```