(2)

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

ANALYSIS OF AN IMAGE PROCESSING
ALGORITHM FOR ITS IMPLEMENTATION IN REAL TIME

by

Roberto M. Ventura

March 1988

Thesis Advisor: C.W. Therrien

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b OFFICE SYMBOL (If applicable) 62	7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000	7b ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
8a NAME OF FUNDING / SPONSORING ORGANIZATION 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO.

11 TITLE (Include Security Classification)
ANALYSIS OF AN IMAGE PROCESSING ALGORITHM FOR ITS IMPLEMENTATION IN REAL TIME

12 PERSONAL AUTHOR(S)
VENTURA, Roberto M.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	March 1988	50

16 SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not relfect the official policy or position of the Department of Defense of the U.S. Government

17 COSATI CODES		
FIELD	GROUP	SUB-GROUP

18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Image enhancement, Contrast Enhancement, Underwater Imaging, Underwater Recovery Operations

19 ABSTRACT (Continue on reverse if necessary and identify by block number) This thesis deals with the implementation and analysis of an image processing algorithm in order to determine the feasibility of its operation in a real time environment at standard video frame rates.

A modification of the Peli and Lim algorithm has been to put work in processing images taken from a submerged camera operating in turbid water environments with uneven light distribution. The algorithm's performance was observed in detail to determine if it met the required specifications. Through analysis, the algorithm's intense computational requirements and large memory storage demands were resolved. It was determined that for the algorithm to operate in real time, a system with characteristics of a supercomputer would most likely be needed.

The algorithm was transported to an IBM AT equipped for image processing. Possible optimization techniques were discussed briefly, and other solutions for processing images with this type of equipment were suggested.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL C.W. Therrien	22b TELEPHONE (Include Area Code) (408) 646-3347 22c OFFICE SYMBOL 62Ti

☆ U.S. Government Printing Office: 1986—606-24

Approved for public release; distribution is unlimited

Analysis of an Image Processing Algorithm for its Implementation in Real Time

by

Roberto M. Ventura
Lieutenant, Colombian Navy
B.S., Escuela Naval "Almirante Padilla", 1983

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March, 1988

Author: _____
Roberto M. Ventura

Approved by: _____
Charles W. Therrien, Thesis Advisor

_____
Roberto Cristi, Second Reader

_____
John P. Powers, Chairman of the Department of
Electrical and Computer Engineering

_____
Gordon E. Schacher
Dean of Science and Engineering

ABSTRACT

This thesis deals with the implementation and analysis of an image processing algorithm in order to determine the feasibility of its operation in a real time environment at standard video frame rates.

A modification of the Peli and Lim algorithm has been put to work in processing images taken from a submerged camera operating in turbid water environments with uneven light distribution. The algorithm's performance was observed in detail to determine if it met the required specifications. Through analysis, the algorithm's intense computational requirements and large memory storage demands were resolved. It was determined that for the algorithm to operate in real time, a system with characteristics of a supercomputer would most likely be needed.

The algorithm was transported to an IBM AT equipped for image processing. Possible optimization techniques were discussed briefly, and other solutions for processing images with this type of equipment were suggested.

TABLE OF CONTENTS

I. INTRODUCTION

The Naval Undersea Warfare Engineering Station (NUWES), in Keyport, Washington, frequently conducts torpedo recovery operations on its testing range. These operations are carried on undersea with the aid of special digging equipment. In order to maneuver this equipment, its activity is monitored with underwater video cameras and the operation is observed on video monitors at the surface onboard the recovery vessel.

The underwater cameras are equipped with strong artificial lights, which shine irregularly on the objective, and occasionally cause intense specular reflections of light in the direction of the camera.

From the described environmental conditions it can be anticipated that the true dynamic range of the image will not be faithfully represented on the video system. To make things worse, the recovery equipment, in its attempt to dig out the torpedo, stirs up sediments which obscure the objective, and produce additional reflections that further degrade the image. In many cases the degradation is such that the whole digging operation must be stopped in order to wait for the sediments to settle, and allow sufficient visibility to continue.

There is clearly a need to perform some processing of the image in order to enhance it before displaying it on the monitor. To date, no adequate technique capable of rapidly and efficiently enhancing the image before display, has been found.

The purpose of this thesis was to implement and analyze an algorithm which has been previously used to perform enhancement on single frames of these images. Major modifications to the algorithm have been introduced for two specific reasons: to avoid the accentuation of the background noise, and to make the algorithm less image dependent, thus requiring less operator intervention to perform.

A further purpose of the thesis was to determine what could be done to process images with this algorithm on an IBM-PC AT compatible computer, that would in some way aid the torpedo recovery operation at NUWES. A small machine this size could be easily installed on board the recovery vessel and incorporated with other equipment.

II. THE PELI AND LIM ALGORITHM

A. DESCRIPTION AND APPLICABILITY

When an image with a large dynamic range is recorded on a medium with a smaller dynamic range, the details of the image, principally in areas with very high and very low luminance values, are not well represented. This is the case for the images that will be dealt with throughout this thesis. The Peli and Lim Algorithm [Ref. 1] performs contrast enhancement adaptively and mitigates the reduction in dynamic range of an image by modifying the values of the local contrast and local luminance mean separately.

Figure 1 shows a block diagram of the algorithm. The vector $\underline{n}$ represents a pair of spatial coordinates and $f(\underline{n})$ denotes the unprocessed digital image; for our purposes a 512 by 512 8-bit pixel array of values between 0 and 255. The function $f_L(\underline{n})$ represents the local luminance mean of $f(\underline{n})$, which is obtained by low-pass filtering. The function $f_H(\underline{n})$ denotes the local contrast, obtained by subtracting the local luminance mean values from the original image values.

The local contrast is modified by multiplying its value by a factor $k(f_L)$, which varies according to the value of the local luminance mean of the given pixel. The modified contrast is designated $f'_H(\underline{n})$. The local luminance mean is modified by an intensity mapping which may be nonlinear; the result is

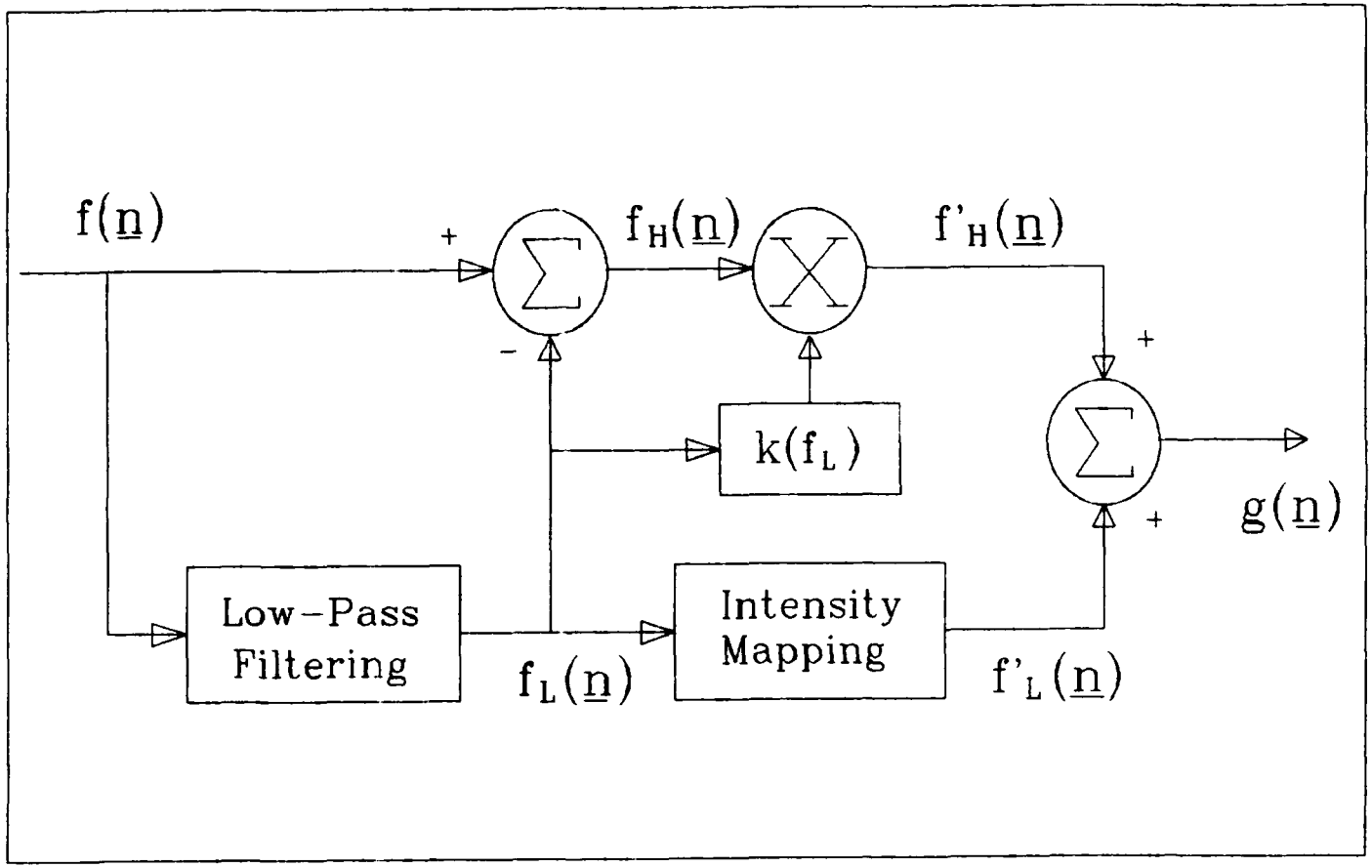

Figure 1 – Peli and Lim Algorithm – Block Diagram

$f'_L(\underline{n})$. The specific mapping chosen depends on the desired effect. Typically the mapping is selected so that the overall dynamic range of the resulting image is approximately the same as that of the medium on which it is to be recorded. Finally, the two modified components are combined to obtain the processed image $g(\underline{n})$.

Considering the algorithm's features, highlighted above, and the given conditions of the environment where the imaging takes place, we could expect that its processing should be favorable to our needs. The local contrast can be enhanced in vicinities of low luminance, where the artificial lights do not have the required intensity, and also in high luminance areas, where reflections from the objective are too bright.

Figure 2 shows examples of the functions of the factor $k(f_L)$ used in a former implementation of the Peli and Lim Algorithm [Ref. 2] that was used to process single frames of these same images. The results obtained in terms of contrast enhancement were highly satisfactory. However, undesired background noise, already present in the image, was accentuated greatly. Figures 3 and 4 show the original and the processed images, respectively.

B. MODIFICATION INTRODUCED

Since the algorithm had the unwanted effect of intensifying the background noise of this particular type of images, Professor Jae S. Lim of M.I.T., one of the co-authors of the algorithm, suggested [Ref. 3] that the factor **k**, by which the

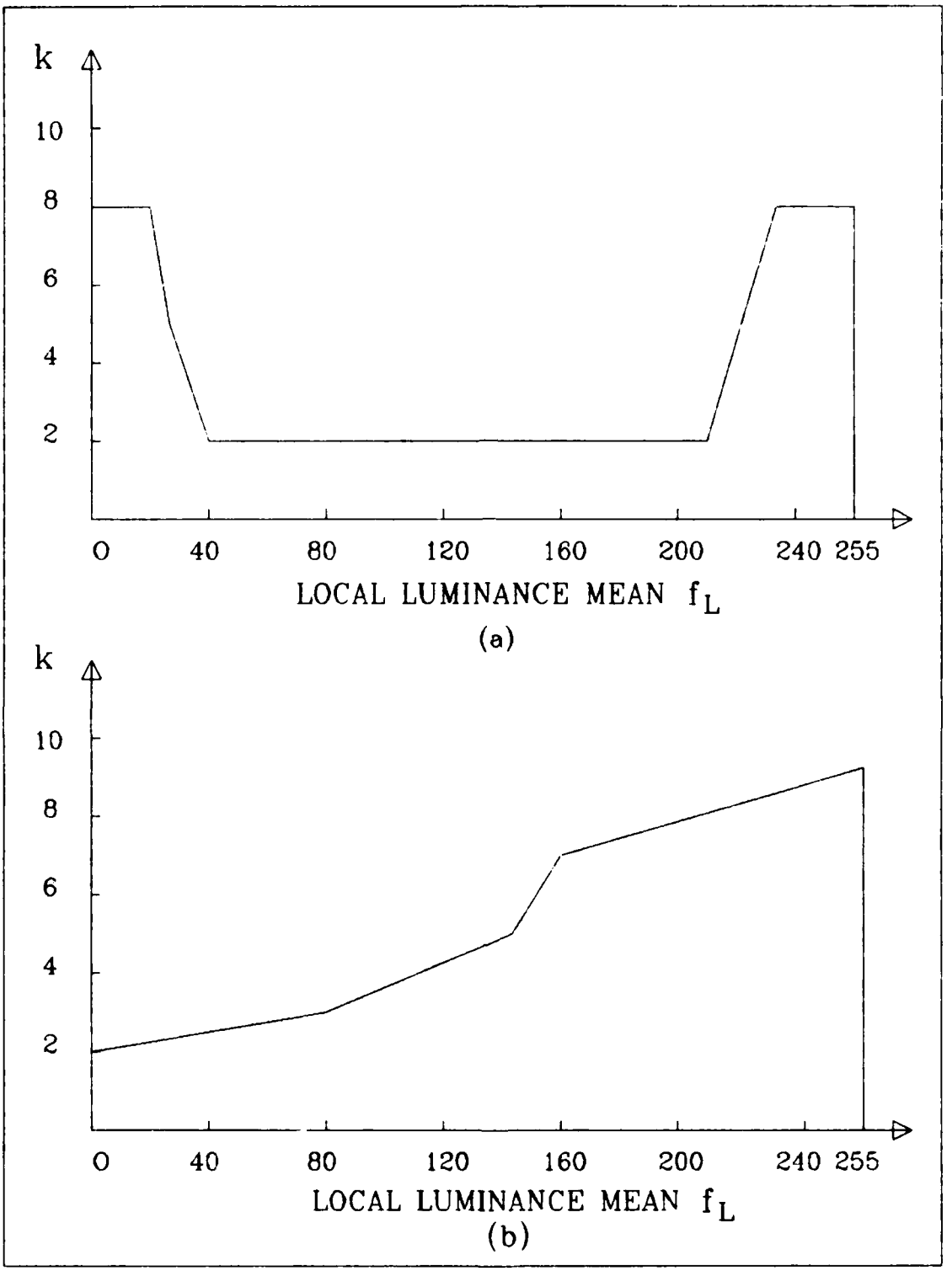

Figure 2 – $k(f_L)$ Functions used in [Ref. 2]

Figure 3. Original Degraded Image.

Figure 4. Image Processed by Peli & Lim Algorithm.

local contrast is multiplied, should be made to be a function not only of the local luminance mean, but also of the local variance.

The algorithm was modified to compute the local variance which is a measure of the energy of the high frequency component of the image. This quantity was computed as the sum of the squares of the values of $f_H(\underline{n})$ over a neighborhood of 25 pixels.

By setting variance threshold points, areas composed exclusively of smooth backgrounds can be isolated from the rest of the image. Increasing the local contrast in these areas, would result in the intensification of noise. Therefore, the parameters of the algorithm are set so that only the local luminance mean values are modified. In the more irregular regions of the image where the presence of edges or boundaries of different objects is evident, the values of the variance will be higher. In these regions, both the local luminance mean and the local contrast values are modified. If the variance values are higher than a third threshold, then the local contrast of the pixels they represent is high enough, and no changes in contrast are required.

Figure 5 shows an example of the degraded image after it has been processed with the modified Peli and Lim Algorithm. It can be seen that it is sharper than Figure 3, the original. Although contrast enhancement could be more evident in the image of Figure 4, there is also noise accentuation in the

smooth regions of the image. This is not the case of the image of Figure 5.

Figure 5. Image Processed by Modified Peli & Lim.

Figure 6 shows the block diagram of the modified Peli and Lim Algorithm. The new functional block introduced is the variance calculation. The output of this operation is one of the inputs for the next functional block. The calculation of the factor k, involves the multiplication of its two components k_1 and k_2 as shown in the following equation:

$$k(f_L,\sigma^2) = k_1(f_L) \times k_2(\sigma^2)$$

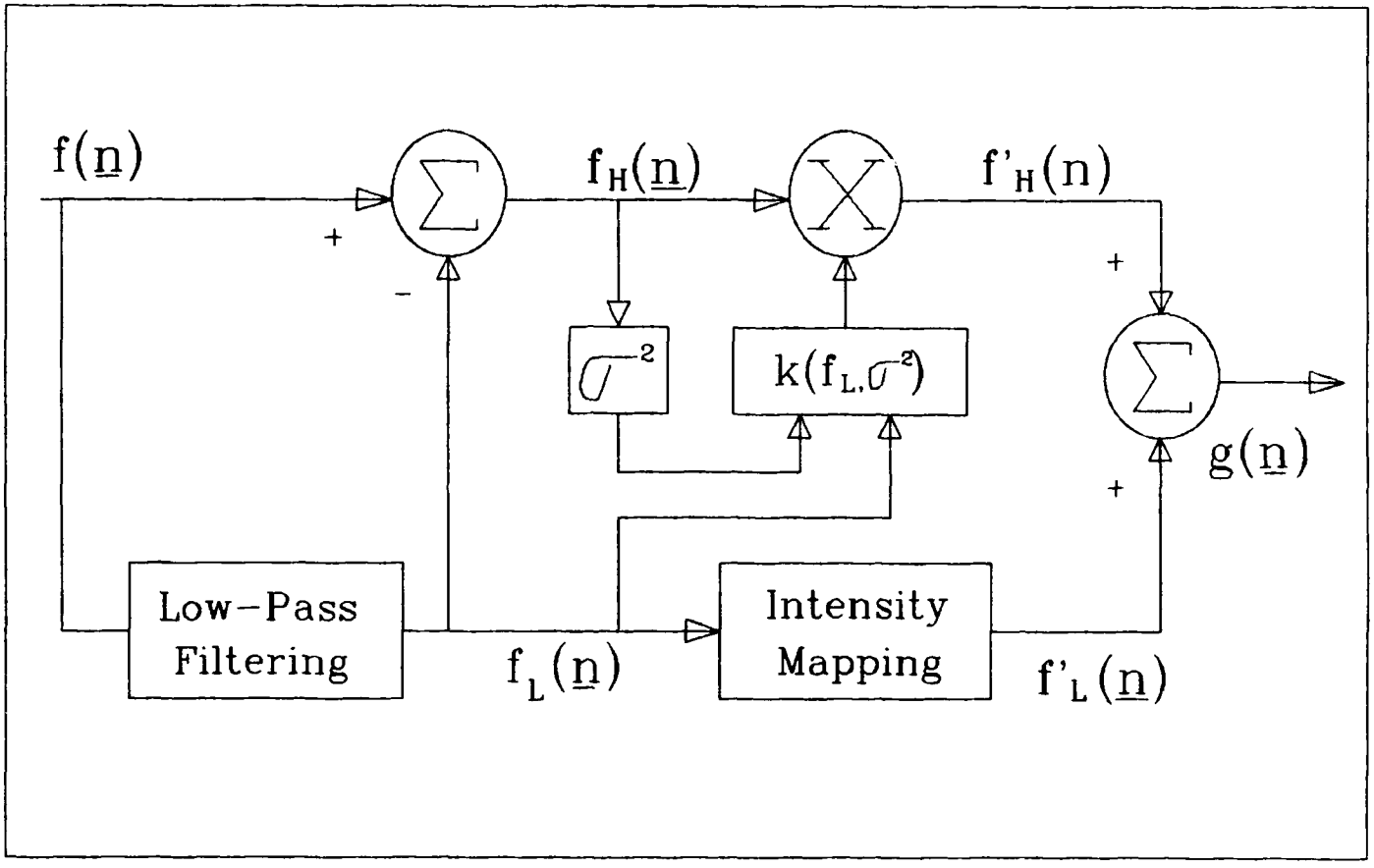

Figure 6. – Peli and Lim Algorithm – Modified,

An example of the dependence of the k_2 operator on the local variance is shown in Figure 7 (a). An example of the function of the factor k_1 is given by the equation:

$$k_1(f_L) = 0.75 \times f_L(\underline{n}) + 35$$

Figure 7 (b) shows an example of a typical intensity mapping used to operate on the local luminance mean. This particular mapping is chosen to reduce the dynamic range of the local luminance, which in fact, is the more dominant of the two components. In the figure, the range of input local luminance mean values is from 0 to 255, while the range of output local luminance mean is 20 to 235.

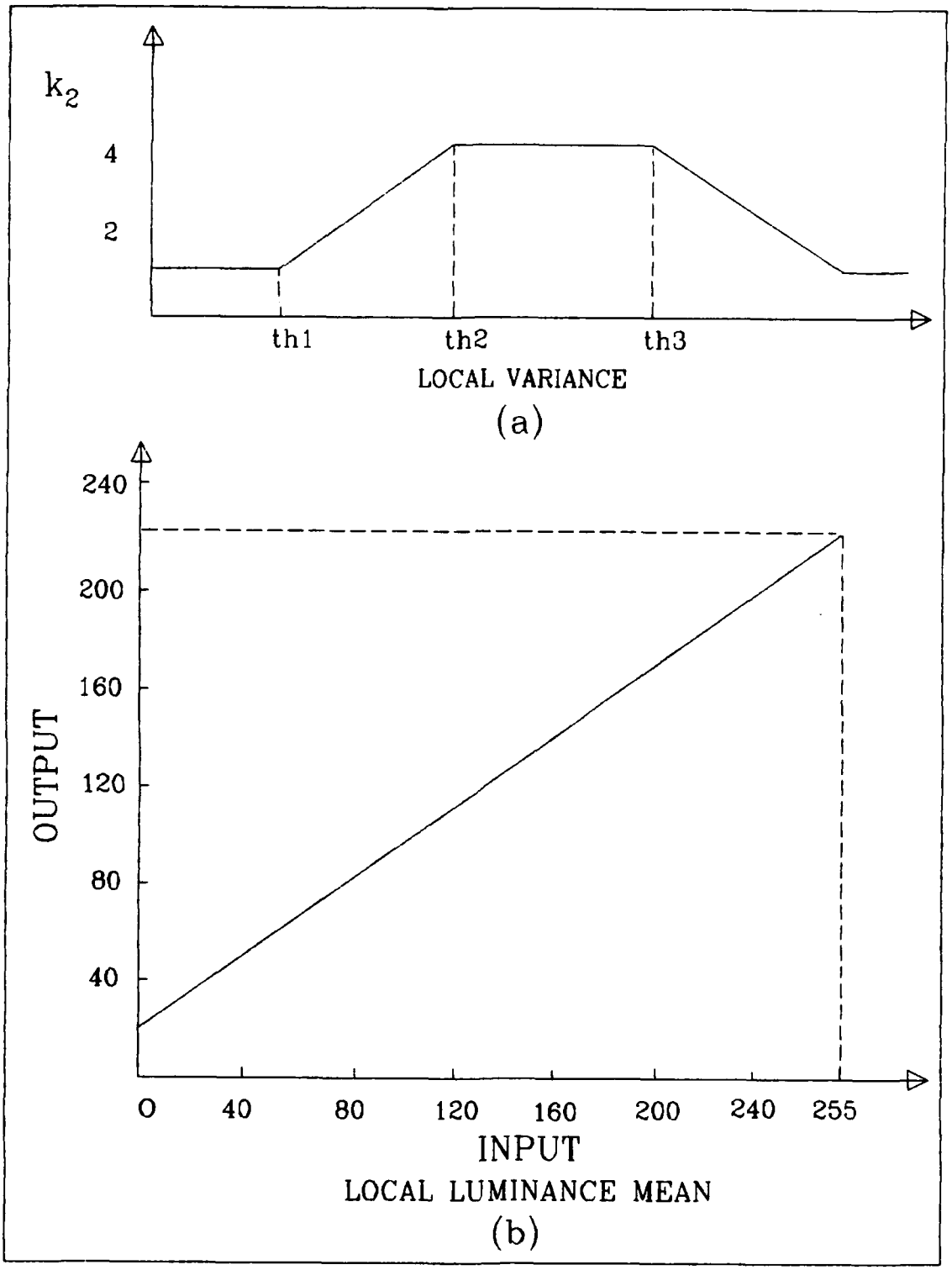

Figure 7. k Coefficient and Intensity Mapping Functions.

III. ANALYSIS

For a process to run in real time, it must finish processing a set of input data before the next subsequent set of input data is available. If the Modified Peli and Lim Algorithm is to run in real time at standard video frame rates[1], it should be capable of processing thirty image frames in one second, or one frame every thirty three milliseconds.

Computations take a finite amount of time. When the input data set is large, performing even a simple computation can take a long time.

Data moves, or load/store operations also require a finite amount of time. There must be an address calculation performed in order to determine the physical address of an operand for it to be moved to a CPU register, where the ALU can deal with it.

In this analysis the number of operations that must be completed in order to process an image 512 x 512 pixels in size with the Peli and Lim Modified Algorithm will be determined. The storage capacity, the computation speed, and the data transfer rates that a computer system must have in order to process the image in a specified time interval will also be discussed.

---

[1]RS-170 Video Standard.

A. MEMORY STORAGE REQUIREMENTS

To determine the size of an array, the size of the data type used to represent each of its elements must be known. Different implementations of the algorithm will therefore require different storage capacities depending on the type of computer system being used as a host. For instance, an array of 262,144 integers on a large mainframe, which typically uses 64 bit integers, would require 2 megabytes of storage. The same array would occupy half the storage on a minicomputer, if the integer were to be represented with 32 bits. On a small micro or a PC, an unsigned short integer data type is only 16 bits long, and can represent integer values up to 65,536. This quantity is suitable for the application, and the storage required would be only half a megabyte.

As stated before, the image can be stored in a 512 by 512 8-bit pixel array. Some implementations of programming languages like the Microsoft C Compiler, allow the use of the unsigned character data type. This data type is perfect for representing a digital image, and allows the compressing of the array to 256 KBytes of memory storage.

During the execution of the program, there is a second one of these arrays generated from the low-pass filtering operation, which in turn is subtracted from the original image pixel values to obtain the local contrast. At all times the program must maintain these two arrays in memory since their values are constantly being referenced.

The only other array that results at run-time is the local variance. The highest possible value resulting from this computation can be scaled down so that it can be represented with a 16 bit unsigned integer. This will minimize the amount of storage needed, contributing only with an extra 512 KBytes to the previous requirement.

In summary, the implementation of the algorithm must have at least one megabyte of memory allocated to the process at all times during execution. On the surface, this does not look so bad. However, considering a common virtual memory paging system like the VAX-11/780 as an example, the page size used is only 512 bytes. The peak working set size of pages for a normal user is in the vicinity of 300, therefore the maximum memory allocated to the process at one time would be 153.6 KBytes. Quite a few page faults could be expected during the execution of the process.

For every page fault on a multi-user time sharing system, an I/O interrupt is generated. The process is then blocked and enters a ready queue until the disk I/O operation is completed [Ref. 4]. While a memory reference requires hundreds of nanoseconds, a disk I/O will need tens and sometimes hundreds of milliseconds to be carried out. If real time operation a goal, an overhead of this size can not be tolerated.

If the algorithm were to be implemented to run on a single user system which does not use virtual memory and does

not run other processes in the background at the same time, the process is guaranteed to run without I/O interrupts, provided a minimum of one megabyte of real memory is allocated to it at all times during execution.

B. COMPUTATIONAL REQUIREMENTS

Arithmetic computations are performed by the Arithmetic-Logic Unit of the CPU. Additions and subtractions can be accomplished ten times faster than multiplications and divisions. Also integer operations require less time than floating point operations since for the latter extra manipulations like normalization must also be performed on the result.

When the specifications of a certain machine cite the time that it takes to execute a floating-point multiply, it is assumed that both operands are located in the CPU registers. However several memory cycles must elapse for these operands to be brought to the registers.

A memory cycle includes the calculation of the address of the memory location of the operand and the actual memory reference or moving of the operand through the system bus into the CPU register. All of these operations must be considered since each one takes a finite amount of time.

In the implementation of the algorithm, for every operation there are two data moves: one to fetch the operand and a second to store the result. Each data move, in turn, involves one address calculation and one memory reference. The size of the values moved when reading and writing the image

is 8 bits (one byte). Other operations, like variance computations involve larger data types like integers, the size of which can be 16 or 32 bits, depending on the machine the program is running on. Nevertheless, in most cases the system bus should be wide enough to move each value in a single memory cycle.

To read the image into memory, or to write the image array to a secondary storage device, there are 512 x 512 = 262,144 data moves each time.

If no particular attempt is made to reduce the number of computations, then in the low-pass filtering or averaging operation with a filter mask of five by five pixels, there are 262,144 x 25 = 6,553,600 operand fetches and the same number of additions. There are also 262,144 divisions and the same number of operand stores.

The variance calculation requires 6,553,600 operand fetches and the same number of multiplications and additions. It also requires 262,144 divisions and operand stores.

During the contrast enhancement operation, there are 262,144 x 2 = 524,288 operand fetches, 262,144 compares and additions, which require the same length of time, and 262,144 multiplies and operand stores.

The intensity-mapping requires a total of 262,144 operand fetches, multiplications, additions, and operand stores. The operation of combining the two functions $f'_L(\underline{n})$ and $f'_H(\underline{n})$ requires a further 262,144 operand fetches and additions.

Summing up all the operations so far performed on the image in this "brute force" implementation, there would be 262,144 x 60 = 15,728,640 load/store operations, 262, 144 x 54 = 14,155,776 additions, and 262,144 x 29 = 7,602,176 multiply/-divide operations, as summarized in TABLE 1. If one floating

TABLE 1. OPERATIONS PERFORMED IN THE ALGORITHM

Program Block	Load/Store	Add/Comp	Mult/Div
Reading Image	262,144		
Low-Pass Filt.	6,815,744	6,553,600	262,144
Variance Calc.	6,815,744	6,553,600	6,815,744
Contrast Enh.	786,432	524,288	262,144
Intensity Map	786,432	524,288	262,144
Saving Image	262,144		
Totals:	15,728,640	14,155,776	7,602,176

point multiplication requires same time as 10 floating point additions, then the number of operations of the whole process would be equivalent to nine million multiplies and sixteen million load/stores.

A system with average data transfer rates of 10 MWords per second (which is not far fetched), and a CPU capable of handling 20 MFLOPS[2] would guarantee the completion of the

[2]Millions of Floating Point Operations per second.

processing of one image frame in two seconds. However, for real time operation this time must be brought down to thirty milliseconds, which would require data rates of 300 MWords per second and CPU's of 600 MFLOPS. These rates would only be achievable with highly specialized architectures involving vector pipeline processors high bandwidth memories and bus systems, and reduced clock periods. Examples of these architectures can be seen on supercomputers like the Cray-1, the Cyber-205, the Cray-XMP, and others [Ref. 5] of this style, which for obvious reasons are entirely out of reach.

Reduction of the number of operations of the algorithm's implementation through optimization or through more clever ways of doing things have not yet been discussed. Consider for instance, a more efficient way of using the available hardware like the look-up tables of the digitizer card. The whole intensity mapping operation could be performed faster and 524,288 load/store and 262,144 add/compare operations could be deducted from the present total. This can alleviate the process of 1.8 percent of its add/compares, and 3.2 percent of its load/stores.

Other techniques help greatly in eliminating redundant operations from program segments in which neighborhood operations are performed. An example is "memorizing" the sum of the pixels values bounded by a 5 by 5 filter mask and modifying it as the mask is shifted. This way, five new values are added, and another five are subtracted with every shift.

As a result 15 add and 15 load/store operations are saved each time, and this amounts to a reduction of 3,957,760 operations for the filtering of the whole image. The resulting totals shown in TABLE 2 demonstrate the significant reduction

TABLE 2. OPERATIONS PERFORMED AFTER OPTIMIZATION

Program Block	Load/Store	Add/Comp	Mult/Div
Reading Image	262,144		
Low-Pass Filt.	2,857,984	2,595,840	262,144
Variance Calc.	2,857,984	2,595,840	2,857,984
Contrast Enh.	786,432	524,288	262,144
Intensity Map	786,432	524,288	262,144
Saving Image	262,144		
Totals:	7,813,120	6,240,256	3,644,416

of the total number of operations by approximately fifty percent. This implementation will process an image on the same system described earlier in one second instead of two. For real time, however, 150 MWords/sec data transfer rates and 300 MFLOPS of processing power are required. This speed is still only within the range of capabilities of a supercomputer.

IV. IMPLEMENTATION

While it is not feasible to consider the operation of the Peli and Lim Algorithm in real time without expensive and very specialized hardware, it is nevertheless reasonable to ask what can be done to process these underwater images with simple, off-the-shelf, relatively inexpensive equipment such as an IBM-PC AT provided with image processing capabilities.

After its implementation and testing on the VAX-11/780 in Pascal, the Modified Peli and Lim Algorithm was transported to a system with characteristics similar to the one in the research activity at NUWES. Since a system such as this occupies little space, it can be easily installed on board the torpedo recovery vessel, where it could be used to aid the digging operations.

A. SYSTEM DESCRIPTION

The image processing peripherals added to the host were a PC-Vision Plus Frame Grabber[3], a combination image digitizer and video monitor controller, a Sony VP-5020 U-matic tape player, and a Sony PVM 1271-Q high resolution monitor. The tape player serves as a video source, and can be replaced by a camera when needed. An example of the system configuration is shown in Figure 8.

---

[3]Trademark of Imaging Technology Inc.

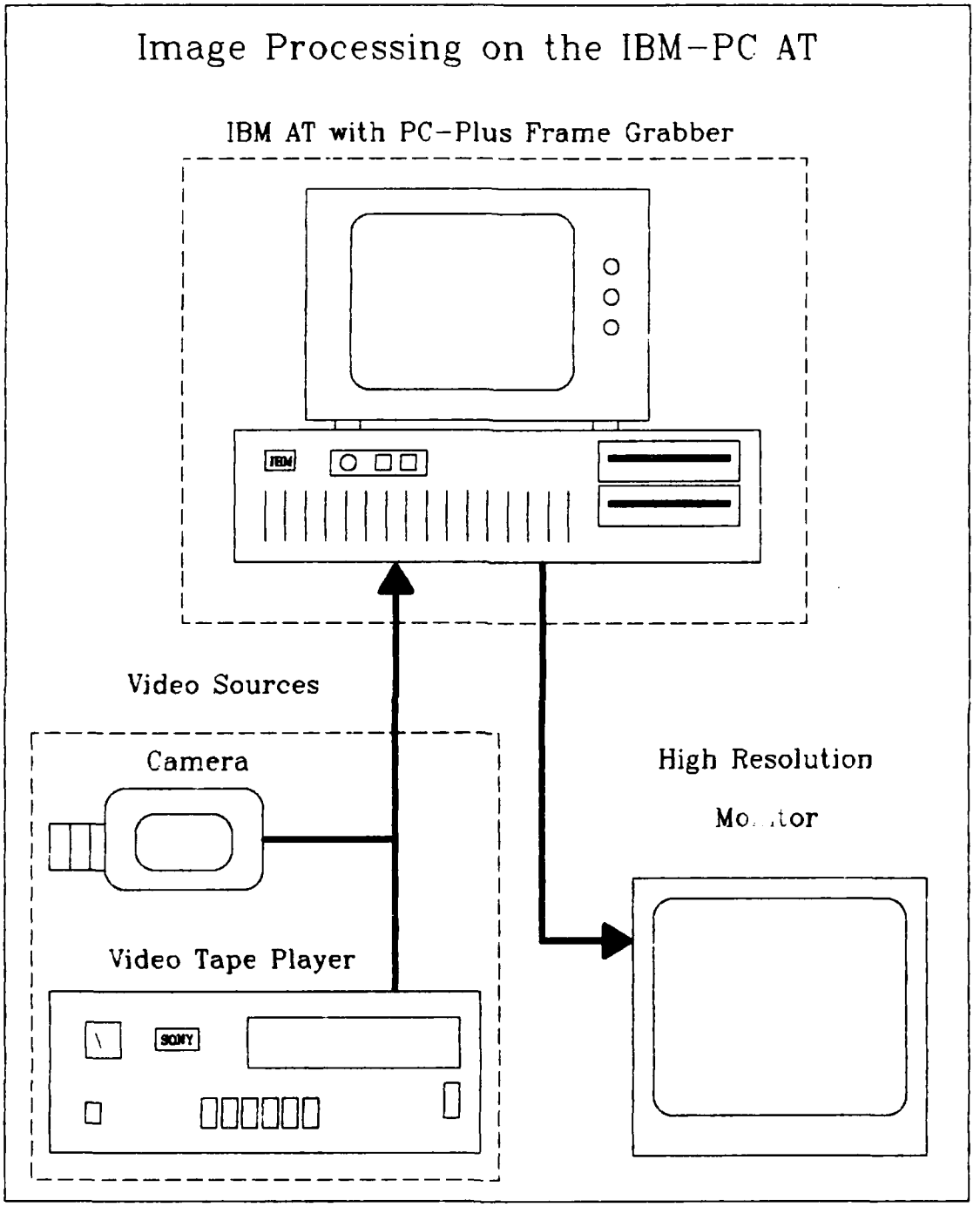

Figure 8. Image Processing System on a PC.

The digitizer is provided with enough frame memory to store two digitized 512 by 512 images at a time. This allows it to display one image on the monitor while the other one is being digitized or processed. Using one single frame buffer, the frame grabber is capable of digitizing images from the video source and continuously displaying them at a rate of 30 frames per second. The digitizer can be controlled by the PC in real time, while its CPU is performing other functions. A block of 64 KBytes of frame memory of the digitizer is mapped to the host's memory address space, and image data can be transferred to and from the host through the system bus. Therefore images can be digitized and saved to a secondary storage device or transferred to the PC's memory for processing at any given time.

The installation of the digitizer on the AT introduced a severe limitation to the system. The maximum amount of random access memory that an 80286 microprocessor can address under the current operating system[4] is 640 KBytes (addresses 0000H to FFFFFH). Of these, 512 KBytes are physically located on the PC's system board (addresses 00000H to 80000H), and the other 128 KBytes must be located on an add-in board. The default settings of the Frame Grabber map its block of frame memory to the PC's address space beginning at A00000H, which is also the starting address of the video memory of the graphics adapter. Due to this conflict, the starting

[4]IBM PC DOS Version 3.3.

address of frame grabber's memory had to be changed to a lower location like 90000H[5].

Since the memory add-in board does not allow the mapping of single 64 KByte blocks, but rather of 128 KByte blocks, 64 KBytes of the PC's address space (addresses 80000H to 90000H) had to be sacrificed in order to be able to map the 64 KBytes block of the digitizer's memory between addresses 900000H to A00000H. In essence, twenty percent of the memory, a total of 128 KBytes, could not be used, bringing the total available RAM down to 512 KBytes.

As mentioned in section III. a. of this thesis, the minimum available memory storage required for the algorithm to process the entire image is a little over 1 MByte. With only half of that amount, a great restriction was imposed on its implementation, and the only solution to the problem was to process the image partially.

B. PERFORMANCE

A reduction of the image to one quarter of its original size made sense, since the array to represent it would only have 65536 elements. Two of these arrays would need 128 K-Bytes, and if a third one composed of the same number of unsigned short integers (16 bits) were included, the total storage requirement would be of one quarter megabyte. However, when the whole program was compiled and linked, the executive

---

[5]589,824 decimal.

file turned out to be 524 KBytes long. This was too large for the computer to handle. Still more reduction was needed.

In order to save more memory space, an array of only 156 by 156 pixels in size was actually processed. This brought the storage required for the arrays down to the vicinity of 100 K-Bytes, and the whole compiled and linked program was near 400 KBytes in size. Timers were used to determine the amount of time each functional block took to execute. The source code of the implementation written in **C** can be found in Appendix A. TABLE 3 shows the execution times of the various program segments in seconds.

TABLE 3. EXECUTION TIME IN SECONDS

Functional Block	Time
Reading the Image	27
Reducing the Image	6
Low-pass filtering	117
Variance computation	152
Contrast enhancement	17
Intensity mapping	6
Expanding the image	3
Saving the image	30
Total execution time	358

If the optimization technique that eliminates the redundant operations from the program segments where filtering is

performed were implemented, the execution times of the two functional blocks of longer duration would be reduced by approximately 60 percent. Since the execution time of these two blocks is 75 percent of the execution time of the whole process, their reduction in time would in turn reduce the overall execution time by approximately 45 percent.

C. OTHER SOLUTIONS

Considering that it is not possible for the Peli and Lim implementation to operate in near real time on a computer system such as the IBM-PC AT, there are ways in which the torpedo recovery operation at NUWES can be aided with this type of equipment. In the following sections, two of the most important ones will be discussed.

1. Use of the Look-up Tables

Perhaps one of the most useful features that the digitizer has, is the capability of mapping the intensity values of the pixels of an image in real time, through its look-up tables.

The look-up tables consist of a set of memory locations on the digitizer, whose address is determined by the intensity of a pixel in question. The content of a specific memory location is a value of the mapping function programmed in the look-up table in advance. As the image is displayed, the values of its elements are substituted by the values in the look-up table, and an intensity modified image will be seen on

the monitor. As stated before, this reference is very fast, and it does not affect the real time operation of the digitizer. In addition there are two sets of look-up tables. One is at the input of the card, and will change the value of the pixel as it is being digitized; the other one is at the output, and will change the value of the pixel only as it is being displayed.

The look-up tables can be programmed in many ways, to achieve different effects in the displayed image. Linear functions can be used as well as nonlinear ones. Whole arrays of values can be accommodated in the look-up tables for any desired mapping. This function can be an approximation of a histogram equalization transformation, and so can be very helpful in the case of enhancing the contrast of the type of underwater images this thesis deals with. Figure 9 illustrates the use of a simple linear function programmed in the look-up tables. This particular function maps the input pixel values below 80 to zero, the values above 200 to 255, and stretches the contrast of the image pixel values between 80 and 200. This simple mapping yielded an image that was judged to be somewhat more intelligible than the original image.

2. Exploiting the Double Buffer Feature

As stated before, the Frame Grabber has enough memory to store two 512 by 512 digitized images at the same time. It is also capable of digitizing and displaying the digitized images from the video source continuously, and in real

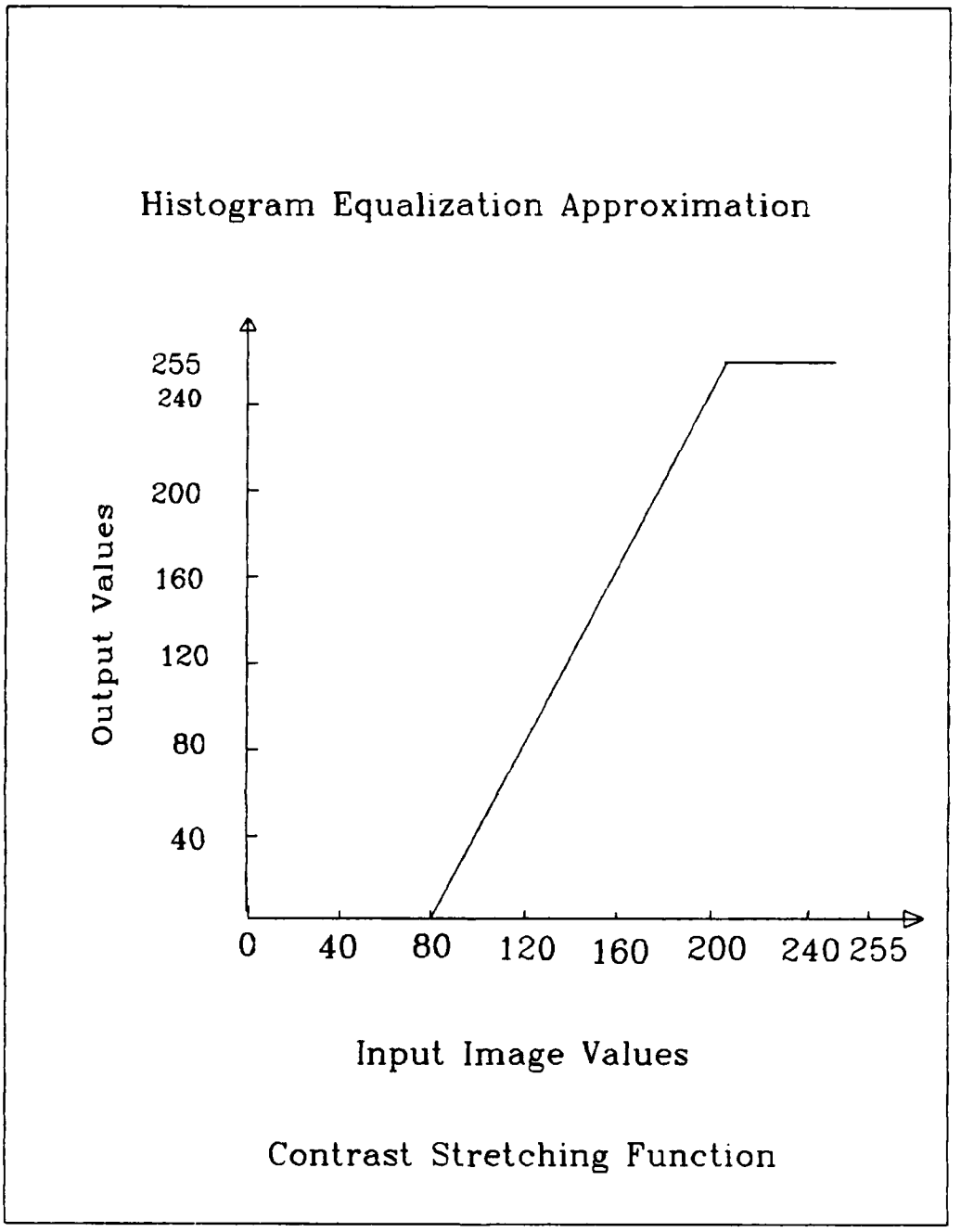

Figure 9. Look-up Table Function.

time, even if the look-up tables have to be referenced at display time.

Considering the appreciable difference in speed between the operation of the digitizer and the processing of an image with the host computer, the torpedo recovery operations on NUWES could be aided by processing one image as needed, and storing it in one of the frame buffers. At the same time, the updated images coming from the video source could be digitized and displayed on the monitor, with a suitable contrast enhancement transformation, stored in the input look-up tables. Since the submerged camera is not constantly moving, but is rather used mainly in a stationary position, the processed image could be displayed by the operator, every now and then, with the real time one, to aid in the identification of specific objects.

V. CONCLUSIONS

The Peli and Lim Algorithm was implemented on the VMS VAX-11/780. Good results were obtained with respect to the local contrast enhancement, but accentuation of background noise was evident. A modification to the Peli and Lim Algorithm involving the use of the local variance as a threshold to operate on the local contrast was also implemented and tested on the VAX. The results obtained were satisfactory, however the computational requirements of the algorithm were increased substantially.

An analysis of the algorithm was performed to determine its storage and computational requirements, and it was concluded that for the algorithm to operate in real time, special and very costly computer architectures would most likely be needed.

The Modified Peli and Lim Algorithm was transported to the IBM-PC AT environment. Adjustments were made to the algorithm in order for it to perform correctly within the limitations of the PC. In this environment, only a portion of the image could be processed due to memory constraints.

Other ways of aiding the torpedo recovery operations at NUWES with the available system were suggested. In particular, some combination of using the look-up tables in the frame grabber to provide global contrast equalization in real

time, and the use of the Peli and Lim Algorithm (which takes a few minutes to execute) to provide improved enhancement of single frames seems feasible.

Optimization of the Peli and Lim computational techniques were briefly analyzed, and the results suggest a natural continuation for work in this area.

APPENDIX A

IMPLEMENTATION ON THE PC USING C

```
/* LIM & PELI PARTIAL  */

/* This program processes images using the Adaptive Filtering
   Algorithm (Mod). It must be compiled with Microsoft C 5.0
   using the HUGE Memory model.
   It processes the image by reducing the original matrix to
 1/4 its size. */


# include <stdio.h>
# include <stdlib.h>
# include <time.h>
# define LEN 512
# define HLN 256
# define PAR 156

int ct = 0, i, j, k, l, x, y, one, two, three, varnce,
           lowlim, uplim;

float m, intcp, factor, average, k1, k2, deltax, slope, value;


double variance;

unsigned char head[81], huge orig[LEN][LEN], hi[PAR][PAR],
                        lo[PAR][PAR];

unsigned short var[PAR][PAR];

FILE huge *fptr, *fp;

main()
   {
   time_t begin, start, finish;
   time (&begin);

   printf ("\n   Reading Parameters ...   \n");

   fp = fopen ("par.dat", "r");
   fscanf (fp, "%f", &factor);
   fscanf (fp, "%d", &one);
   fscanf (fp, "%d", &two);
   fscanf (fp, "%d", &three);
   fscanf (fp, "%d", &lowlim);
```

```
        fscanf (fp, "%d", &uplim);
        fclose (fp);

        printf ("\n Parameters: \n\n");
        printf ("factor: %f \n", factor);
        printf ("one %d \n", one);
        printf ("two %d \n", two);
        printf ("three %d \n", three);
        printf ("lowlim %d \n", lowlim);
        printf ("uplim %d \n", uplim);

        printf ("\n   Reading Image ...   \n");

        fptr = fopen ("inp.dat", "r+b");
        for (i = 0; i < 81; i++)
           head[i] = fgetc (fptr);
        for (i = 0; i < LEN; i++) {
           for (j = 0; j < LEN; j++)
              orig[i][j] = fgetc (fptr);
        }
        fclose (fptr);
        time (&finish);
        printf ("\n %f seconds to read the image.\n\n",
                difftime(finish,begin));

        printf ("\n   Compressing Image ...\n");

        time (&start);
        for (i = 50; i < 206; i++) {
           for (j = 50; j < 206; j++)
              hi[i-50][j-50] = ( (int) ((orig[2*i][2*j] + orig
                          [2*i+1][2*j] +  orig[2*i][2*j+1]
                          + orig[2*i+1][2*j+1]) / 4));
     }
        time (&finish);
        printf ("\n %f seconds to compress the image.\n\n",
                    difftime(finish,start));

        printf ("\n   Low-Pass Filtering ...\n");

        time (&start);
        for (i = 0; i < PAR; i++) {
           for (j = 0; j < PAR; j++) {
              average = 0.0;
                 for (k = 0; k < 5; k++) {
                    for (l = 0; l < 5; l++) {
                       x = i + k - 2;
                       if (x < 0) x = 0;
                       if (x >= PAR) x = PAR - 1;
                       y = j + l - 2;
                       if (y < 0) y = 0;
                       if (y >= PAR) y = PAR - 1;
```

```
                average = average + hi[x][y];
            }
         }
      lo[i][j] = ((int) (average / 25.0));
   }
}
for (i = 0; i < PAR; i++) {
   for (j = 0; j < PAR; j++) {
      hi[i][j] = hi[i][j] - lo[i][j];
   }
}
time (&finish);
printf ("\n %f seconds to filter the image.\n\n",

        difftime(finish,start));

printf ("\n   Computing the Variance ...\n");

time (&start);
for (i = 0; i < PAR; i++) {
   for (j = 0; j < PAR; j++) {
      variance = 0.0;
      for (k = 0; k < 5; k++) {
         for (l = 0; l < 5; l++) {
            x = i + k - 2;
            if (x < 0) x = 0;
            if (x >= PAR) x = PAR - 1;
            y = j + l - 2;
            if (y < 0) y = 0;
            if (y >= PAR) y = PAR - 1;
            value = hi[x][y];
            variance = variance + (value * value);

         }
      }
      var[i][j] = ((int) (variance / 25.0));
   }
}
time (&finish);
printf ("\n %f seconds to compute the variance.\n\n",
            difftime(finish,start));

printf ("\n   Operating on Local Contrast ...\n");

time (&start);
deltax = two - one;
slope = (factor - 1) / deltax;
for (i = 0; i < PAR; i++) {
   for (j = 0; j < PAR; j++) {
      kl = lo[i][j] * 0.002 + 1;
      value = hi[i][j];
      varnce = var[i][j];
```

```
        if (varnce >= one && varnce < two)
           value = value * k1 * ((varnce - one) * slope +
                 1);
        if (varnce >= two && varnce < three)
           value = value * k1 * factor;
        if (varnce > three) {
           k2 = (three - varnce) * k1 * slope + factor;
           if (k2 < 1) k2 = 1;
           else value = value * k2;
           }
        if (value > 255) ct++;
        hi[i][j] = (int) value;
     }
  }
  printf("\n %d Overshoot\n", ct);
  time (&finish);
  printf ("\n %f seconds to enhance the contrast.\n\n",
              difftime(finish,start));

  printf ("\n   Operating on Local Luminance  ...\n");

  time (&start);
  ct = 0;
  for (i = 0; i < PAR; i++) {
     for (j = 0; j < PAR; j++)
        lo[i][j] = (int) (lo[i][j] * .85 + 15) + hi[i][j];
  }
  printf ("\n %d Overshoot\n", ct);
  time (&finish);
  printf ("\n %f seconds to enhance the local luminance.
           \n\n", difftime(finish,start));

  printf ("\n   Expanding the image  ...\n");

  time (&start);
  for (i = 100; i < 412; i += 2) {
      for (j = 100; j < 412; j +=2)
         orig[i][j] = orig[i+1][j] = orig[i][j+1] = orig
                   [i+1][j+1] = lo[(i-100)/2][(j-100)/2];   }

  time (&finish);
  printf ("\n %f seconds to expand the image.\n\n",
           difftime(finish,start));

  printf ("\n Stretching the Luminance Between %d and %d
           ...\n", lowlim, uplim);

  time (&start);
  m = 255 / (uplim - lowlim);
  intcp = m - lowlim;
   for (i = 0; i < LEN; i++) {
      for (j = 0; j < LEN; j++) {
```

```
            value = orig[i][j];
            if (value <= lowlim) value = 0;
            else if (value > lowlim && value <= uplim)
                  value = value * m - intcp;
            else value = 255;
            orig[i][j] = (int) value;
         }
      }
      time (&finish);
      printf ("\n %f seconds to stretch the luminance values.
               \n\n", difftime(finish,start));

      printf ("\n   Saving the image  ...\n");
      time (&start);
       fptr = fopen ("out.dat", "w+b");
       for (i = 0; i < 81; i++)
          fputc (head[i], fptr);
       for (i = 0; i < LEN; i++) {
          for (j = 0; j < LEN; j++)
             fputc (orig[i][j], fptr);
       }
       fclose (fptr);
       time (&finish);
       printf ("\n %f seconds save the image.\n\n",
                difftime(finish,start));

       printf ("\n %f SECONDS - TOTAL TIME. \n", difftime
              (finish,begin));
}
```

APPENDIX B

IMPLEMENTATION ON THE VAX-11/780 USING PASCAL

```
 Program LIMPELI (Input,Infile,Output,Outfile);

{ Version 3.1: K factor has two components K1 & K2 }

type
    byte=0..255;
    imagero=packed array[0..511]of byte;
    unpimgr=array[0..511]of integer;
    image=array[0..511,0..511]of integer;
    realim=array[0..511,0..511]of real;

var
    photo:image;
    fln,fhn,fln1,fhn1:realim;
    I,J,thpt1,thpt2,thpt3:integer;
    factor:real;
    infile,outfile:file of imagero;


Procedure Getimage (var Picture:image);

var
    I,J:integer;
    row:imagero;
    unp:unpimgr;

begin
    open (infile,'turbid.dat',history:=old,access_method:=
          direct,record_type:=fixed);
    reset (infile);
    for I:=0 to 511 do
    begin
        row:=infile^;
        get (infile);
        unpack (row,unp,1);
        for J:=0 to 511 do
            picture[I,J]:=unp[J];
    end;
    close (infile);
end; { Procedure Getimage }


Procedure Putimage (var Picture:image);

var
    I,J:integer;
    row:imagero;
```

```
    unp:unpimgr;

begin
    open (outfile,'clear.dat',history:=new,access_method:=
          direct,record_type:=fixed);
    rewrite (outfile);
    for I:=0 to 511 do
    begin
        for J:=0 to 511 do
            unp[J]:=picture[I,J];
        pack (unp,1,row);
        outfile^:=row;
        put (outfile);
    end;
    close (outfile);
end; { Procedure Putimage }


Procedure Initialize (var picture:realim);

var
    I,J:integer;

begin
    for I:=0 to 511 do
    begin
        for J:=0 to 511 do
            picture[I,J]:=0;
    end;
end; { Procdure Initialize }


Procedure Convert (file1:image;var file2:realim);

var
    I,J:integer;

begin
    for I:=0 to 511 do
    begin
        for J:=0 to 511 do
            file2[I,J]:=file1[I,J];
    end;
end; { Procdure Convert }


Procedure Truncate (file1:realim;var file2:image);

var
    I,J:integer;

begin
```

```
    for I:=0 to 511 do
    begin
        for J:=0 to 511 do
        begin
            file2[I,J]:=trunc(file1[I,J]);
            if file2[I,J]>255 then file2[I,J]:=255;
            if file2[I,J]<0 then file2[I,J]:=0
        end;
    end;
end; { Procdure Truncate }


Procedure Equate (var file1:realim;file2:realim);

var
    I,J:integer;

begin
    for I:=0 to 511 do
    begin
        for J:=0 to 511 do
            file1[I,J]:=file2[I,J];
    end;
end; { Procedure Equate }


Procedure Average (var Picture:realim;file1:realim;Numpts:i-
nteger);

var
    I,J,K,L,Shift,xshift,yshift:integer;
    Ave:Real;

begin
    shift:=numpts div 2;
    for I:=0 to 511 do
    begin
        for J:=0 to 511 do
        begin
            ave:=0;
            for K:=0 to numpts-1 do
            begin
                for L:=0 to numpts-1 do
                begin
                    xshift:=I+k-shift;
                    yshift:=J+L-shift;
                    if xshift<0 then xshift:=0;
                    if xshift>511 then xshift:=511;
                    if yshift<0 then yshift:=0;
                    if yshift>511 then yshift:=511;
                    ave:=ave+file1[xshift,yshift]
                end;
```

```
            end;
            picture[I,J]:=ave/(sqr(numpts));
        end;
    end;
end; { Procedure Average }


Procedure Comvar (var file1:realim;file2:realim);

var
    variance:real;
    I,J,K,L,xshift,yshift:integer;

begin
    for I:=0 to 511 do
    begin
        for J:=0 to 511 do
        begin
            variance:=0;
            for K:=0 to 4 do
            begin
                for L:=0 to 4 do
                begin
                    xshift:=I+k-2;
                    yshift:=J+L-2;
                    if xshift<0 then xshift:=0;
                    if xshift>511 then xshift:=511;
                    if yshift<0 then yshift:=0;
                    if yshift>511 then yshift:=511;
                    variance:=variance+sqr(file2[xshift,
                                            yshift])
                end;
            end;
            file1[I,J]:=variance;
        end;
    end;
end; { Procedure Comvar }


Procedure Combine
     (var File1:realim;file2:realim;factor:real;increment:
          integer);

var
    I,J:integer;

begin
    for I:=0 to 511 do
    begin
        for J:=0 to 511 do
            file1[I,J]:=file1[I,J]+factor*file2[I,J]+
                        increment;
```

```
    end;
end; { Procedure Combine }


Procedure Operate
    (var file1:realim;file2:realim;factor:real;thpt1,thpt2,
         thpt3:integer);

var
    I,J:integer;
    K1,K2,m,d:real;

begin
    d:=thpt2-thpt1;
    m:=(factor-1)/d;
    for I:=0 to 511 do
    begin
        for J:=0 TO 511 do
        begin
            K1:=fln[I,J]*0.002+1;
            if (file2[I,J]>=thpt1) and (file2[I,J]<thpt2) then
                file1[I,J]:=(file1[I,J]*K1)*((file2[I,J]-
                                thpt1)*m+1);
            if (file2[I,J]>=thpt2) and (file2[I,J]<thpt3) then

                file1[I,J]:=(file1[I,J]*K1)*factor;
            if (file2[I,J]>=thpt3) then
            begin
                K2:=((thpt3-file2[I,J])*m+factor)*K1;
                if K2<1 then K2:=1;
                file1[I,J]:=file1[I,J]*K2
            end;
        end;
    end;
end; { Procedure Operate }


begin { Main Program LIMPELI }
    writeln (output);
    writeln  (output,'LIM  &  PELI  ALGORITHM  IMPLEMENTA-
                      TION: (ver 3.1)');
    writeln (output);
    writeln (output,'Enter three variance threshold
                      points:');
    writeln (output);
    read (input,thpt1,thpt2,thpt3);
    writeln (output);
    writeln (output,'Enter the scale factor for local
                      contrast operation:');
    writeln (output);
    read (input,factor);
    writeln (output);
```

```
    writeln (output,'Image is being read . . .');
    getimage (photo);
    writeln (output);
    convert (photo,fln),
    equate (fhn,fln);
    writeln (output,'Process begins . . .');
    writeln (output);
    writeln (output,'Averaging begins . . .  ');
    average (fln,fhn,5);
    combine (fhn,fln,-1,0);
    writeln (output,'Completed.');
    writeln (output);
    equate (fhn1,fhn);
    writeln (output,'Variance computation begins . . .  ');
    comvar (fhn1,fhn);
    writeln (output,'Completed.');
    writeln (output);
    writeln (output,'Local contrast operation
                     begins . . .  ');
    operate (fhn,fhn1,factor,thpt1,thpt2,thpt3);
    writeln (output,'Completed.');
    writeln (output);
    writeln (output,'Local liminance operation
                     begins . . .  ');
    initialize (fln1);
    combine (fln1,fln,0.9,15);
    writeln (output,'Completed.');
    writeln (output);
    combine (fln1,fhn,1,0);
    writeln (output,'Process completed, image being
                      stored . . .');
    writeln (output);
    truncate (fln1,photo);
    putimage (photo);
end.
```

LIST OF REFERENCES

1. Peli, T. and Lim, J. S., "Adaptive Filtering for Image Enhancement," Optical Engineering, Vol.21 No. 1., January/-February, 1982.

2. Franco, Jorge A., Enhancement of Video Images Degraded by Turbid Water, Master's Thesis, Naval Postgraduate School, December 1986.

3. Lim, J. S. and Therrien, C. W., Study on Image Processing for Turbid Water Viewing, Research Report, Naval Postgraduate School, May 1987.

4. Tanembaum, Andrew S., Operating Systems: Design and Implementation, Prentice-Hall, Inc., 1987.

5. Hwang, K and Briggs, F. A., Computer Architecture and Parallel Processing, McGraw-Hill, Inc., 1984.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Department Chairman, Code 62 Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943	1
4.	Professor C. W. Therrien, Code 62Ti Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943	3
5.	Professor Roberto Cristi, Code 62 Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943	1
6.	Señor Almirante Comandante Armada Nacional CAN, Avenida ElDorado Bogotá, D. E., Colombia	1
7.	Señor Contralmirante Director Escuela Naval "Almirante Padilla" Cartagena, Colombia	1
8.	Mr. Richard Evans, Code 7021 NUWES, Code 7021 Naval Undersea Weapons Engineering Station Keyport, Washington 98345	1
9.	Mr. Robert Marimon Code 70 Naval Undersea Warfare Engineering Station Keyport, Washington 98345	1
10.	CDR Hiller Code 70 Naval Undersea Warfare Engineering Station Keyport, Washington 98345	1

11. Ms. Tami Peli 1
Gr. 21
M.I.T. Lincoln Laboratory
Lexington, Massachussetts 02173-0073

12. Professor Jae S. Lim 1
Department of Electrical and Computer Engineering
Room 36-653
M.I.T.
Cambridge, Massachussetts 02139

13. Professor O. B. Wilson, Code 61Wl 1
Department of Physics
Naval Postgraduate School
Monterey, California 93943

14. Teniente de Navio 3
Roberto M. Ventura
Escuela Naval "Almirante Padilla"
Cartagena, Colombia