/02719

Image Analysis for Microscope-based Observations: An Inexpensive Configuration

S. E. Campana

Marine Fish Division
Biological Sciences Branch
Bedford Institute of Oceanography
Department of Fisheries and Oceans
P. O. Box 1006, Dartmouth,
Nova Scotia, Canada
B2Y 4A2

September 1987

Canadian Technical Report of Fisheries and Aquatic Sciences No. 1569

Canadian Technical Report of Fisheries and Aquatic Sciences No. 1569

September 1987

IMAGE ANALYSIS FOR MICROSCOPE-BASED OBSERVATIONS: AN INEXPENSIVE CONFIGURATION

by

S.E. Campana

Fisheries and Oceans
Scotia-Fundy Region
Biological Sciences Branch
Marine Fish Division
Bedford Institute of Oceanography
P.O. Box 1006, Dartmouth
Nova Scotia, B2Y 4A2

PREFACE

The use of proprietary names does not imply endorsement of any product or compliance. No reference to Department of Fisheries & Oceans or to this publication shall be made in any advertisement or sales promotion which would indicate or imply endorsement of any product.

Correct citation for this publication:

Campana, S.E. 1987. Image analysis for microscope-based observations: An inexpensive configuration. Can. Tech. Rep. Fish. Aquat. Sci. 1569: iv + 20 pp.

TABLE OF CONTENTS

ABSTRACT

Campana, S.E. 1987. Image Analysis For Microscope-based Observations: An Inexpensive Configuration. Can. Tech. Rep. Fish. Aquat. Sci. 1569: iv + 20 pp.

Image analysis systems allow for image enhancement, manipulation, storage and quantification. They are of particular benefit to those conducting microscopic examinations, since they can display detail and quantify features that would not otherwise be possible. The image analysis system described here is a state-of-the-art microcomputer-based system that operates in real-time. Its substantial price advantage (<$7000 includes software and hardware) over commercially-available systems derives from the requirement for user configuration and some basic programming in the C language. This paper details one such configuration, provides an overview of system capabilities, presents some fisheries applications and provides examples of two commonly-used programs. The intent is to provide enough of a system overview to allow those unfamiliar with the technology to decide if it may meet their needs. The financial and logistic hurdles can be circumnavigated by researchers with a modest budget and a modicum of microcomputer background.

RÉSUMÉ

Campana, S.E., 1987. Analyse d'image en microscopie: une configuration économique. Can. Tech. Rep. Fish. Aquat. Sci. 1569: iv + 20 pp.

Les systèmes d'analyse d'image autorisent l'amélioration, la manipulation, la mise en mémoire et le traitement quantitatif des images. Ils sont tout indiqués pour les observations au microscope puisqu'ils permettent de visualiser des détails et de quantifier les caractéristiques observées. Autant de possibilités qui ne sont pas données au microscope proprement dit. Nous décrivons dans ce rapport un système de technologie de pointe géré par micro-ordinateur et exploité en temps réel. Ce système est offert à un prix nettement avantageux (moins de $7,000 pour le matériel et le logiciel) par rapport aux systèmes que l'on trouve sur le marché. Par contre, l'utilisateur doit en réaliser lui-même la configuration et élaborer certains programmes de base en C (language informatique). Nous présentons un modèle de configuration, un aperçu des possibilités du système, quelques applications dans le domaine des pêches et, à titre d'exemple, deux programmes qui sont couramment utilisés. Ce survol du système se veut suffisamment complet pour que les non-initiés puissent déterminer si le système est à même de répondre à leurs besoins. Les chercheurs qui ont un budget raisonnable et une certaine connaissance des micro-ordinateurs pourront ainsi contourner les difficultés financières et logistiques inhérentes à l'acquisition de matériel de ce genre.

INTRODUCTION

Image analysis is a generic term used to refer to the digitization and manipulation of visual images, usually by a computer. In its simplest form, an image analysis system (IAS) can store a picture in memory and allow for its subsequent recall and display upon command. Such a system is capable of reproducing the original image, unaltered. In practise however, images entered into an IAS are generally enhanced and/or quantified before re-display; therein lies their advantage over visual examination. The end product is invariably an image (or data) which can be more easily interpreted than the original. Interpretation can be computer-assisted or visual, depending upon the application.

The applications of an IAS are diverse, but most systems can be aggregated into one of two groups on the basis of the type and volume of input data they process. Satellite images (ie. Landsat, Seasat) generally reach an IAS in the form of digital data on magnetic tape. Systems devoted to the analysis of these images usually require rather complex software and large memory storage capabilites, due to the volume and format of data being processed. As a result such systems tend to be associated with a mini- or mainframe computer. In contrast, systems not associated with remote sensing applications are often designed to process images derived directly from a video camera. Since the data flow of such systems is generally lower, the system itself can be smaller. Industrial centres often make use of this type of configuration for automated quality control. At the biological end of the spectrum, analogous configurations are common, although an interface between the video camera and a microscope allows smaller objects to be examined.

IAS-microscope combinations offer a considerable advantage over simple microscopic observations. Procedures such as shape analysis and automated object measurement/ classification are impossible without an IAS. However, even manual measurements are simplified, since an electronic cursor is easier to position on a TV monitor than is an ocular micrometer within a field of view. Further, the measurement can be recorded automatically in memory, rather than recorded manually. Finally, the image enhancement capabilities of an IAS can bring out detail in a field of view that is not otherwise visible through the microscope. All of these capabilities have existed in image analysis systems since the 1970's. However, recent technological advances in image analysis and microcomputer hardware have brought microcomputer-based systems with these and other capabilities within the financial reach of an individual researcher. Such systems are currently available from a variety of commercial sources, although the price might deter some. With the willingness to do some simple programming and to configure one's own system, substantial savings over a commercial system are possible. The objective of this paper is to outline how this can be accomplished.

The IAS described here consists of a microcomputer, video camera, TV monitor and image analysis software/hardware, and is interfaced with a microscope. The system is as powerful and more flexible than most microcomputer-based systems now on the market, but can be configured for less then $7000 (Canadian 1987 dollars, not including the microscope). The system is relatively easy to use, does not require special training or an inordinate amount of time to configure and can be used as a stand-alone microcomputer at any time without any configuration changes. What follows is not meant to be an exhaustive description of this system. The intent is merely to provide enough of an overview of the capabilities and configuration to allow those unfamiliar with the technology to decide if it may meet their needs. If it does, this report should convince them that the financial and logistic hurdles can be circumnavigated by those with a modicum of microcomputer background.

HARDWARE, CONFIGURATION AND SOFTWARE

NOTE: The system description that follows is by no means the only, or even the best, configuration. While brand names will be mentioned in the text, I make no endorsement, implicit or otherwise, as to their quality or price. Brand names are presented solely as a means for comparison of specifications and because they have been tested and found suitable for the job.

The basic configuration is presented in Fig. 1. The video camera and video monitor are black and white, each with medium to high resolution. The microscope can be of any type, compound or dissecting, and is interfaced with the camera through a standard photoadaptor tube. The microcomputer is IBM-compatible with a 20 Mb hard disk and a standard monitor. The microcomputer also

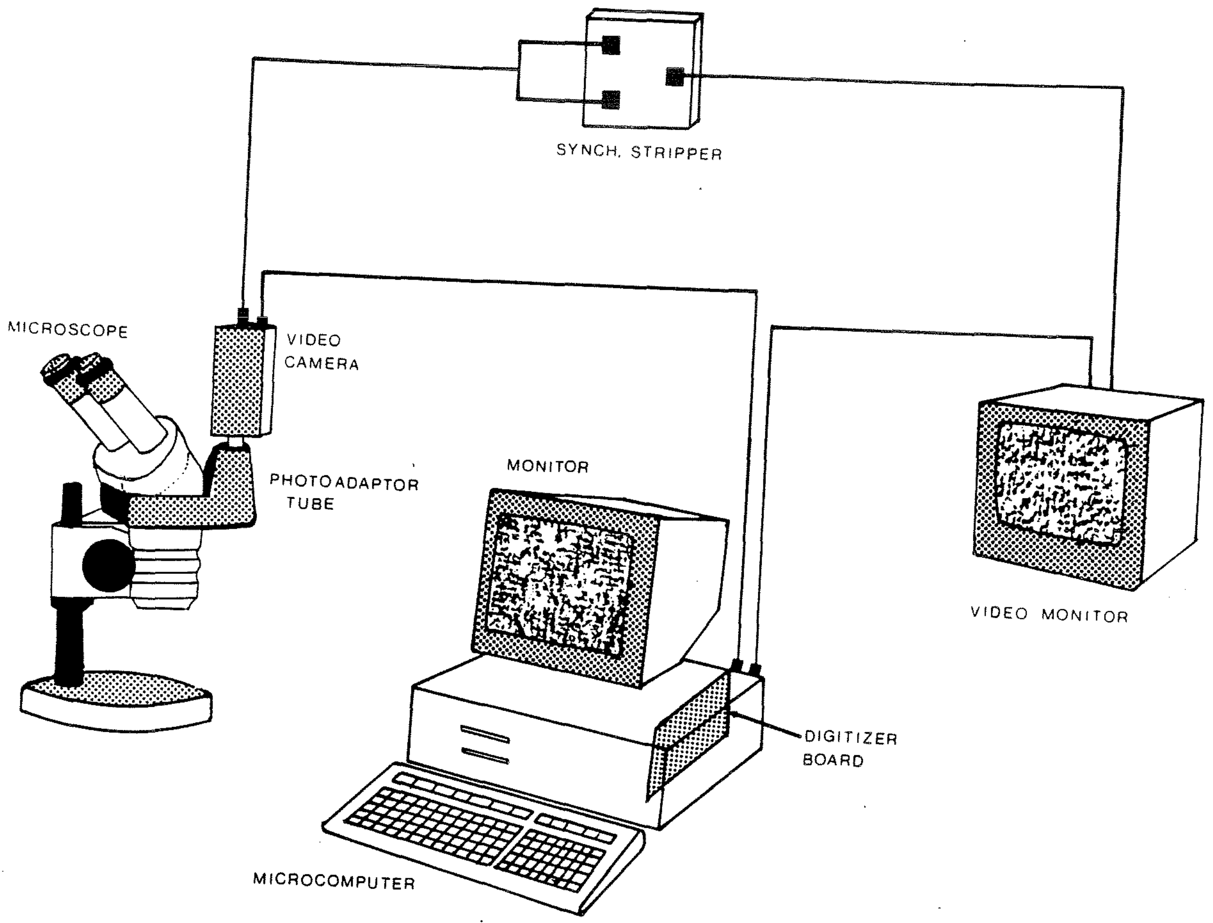

Figure 1. Basic orientation of image analysis system hardware described in the text.

contains an image analysis card in one of its expansion slots.

Hardware Specifications

Microcomputer: IBM-compatible with an 8 MHz clock, 20 Mb hard disk, 640Kb RAM, 1 floppy disk drive, standard keyboard, standard monitor This is a common microcomputer (=PC) configuration suitable for many non-IAS procedures. Some image analysis programs require a colour graphics adaptor (video card). If high resolution graphics capabilities are desired for the non-IAS functions of the PC, an ATI Graphics Solution Card provides either mode upon command.

*Price - $1800

IAS Hardware: Coreco's Oculus-200 Framegrabber video digitizer board. This is a real-time video digitizer board; it and its analogs are called framegrabbers because of the speed with which they write the image into memory. Not to be confused with a digitizing pad which sits on a desk, the Oculus board is a card which is inserted into the PC. It converts a standard RS-170 video image into an array of 480x512 points at the rate of 30 images per second. The digitized image is written into 256K bytes of RAM on the board which is subsequently accessed by the PC.

*Price - $2400

*Approximate price in 1987 Canadian dollars

Video Camera: Hitachi HV-730C closed-circuit TV (CCTV) camera. This is a medium to high resolution, black and white surveillance-type camera which can either be interfaced with a microscope through a microscope photoadaptor tube (camera lens not required) or used by itself (with a lens). Synchronization can be either external (recommended: see below) or internal, as desired. It uses a 2/3" vidicon tube with 650 lines of horizontal and 350 lines of vertical resolution. Alternatively, a video signal can be supplied from a VTR in which case a camera is not required.

*Price - $500

Video Monitor: Hitachi VM-129 12" video monitor. This is a standard black and white TV monitor with medium-high resolution. It has 700 lines of horizontal and 350 lines of vertical resolution.

*Price - $500

"Synch Stripper": Custom-made. This small box strips the synchronization signal from the coaxial cable entering the video camera through the external connector. The remaining video signal is dumped to ground (see synchronization below). If the camera has a separate synchronization connector, the synch stripper is not required.

*Price - $250

Configuration

The basic configuration is displayed in Fig.1. With the possible exception of the synch stripper, the signal flow is straightforward. The video signal (composite video, RS-170) leaves the camera and travels to the digitizer board in the PC, where it is digitized. The digitized image is then reconstituted and sent to the monitor for display. The entire process takes 1/30th second and can be placed in continuous mode such that the image is displayed in what is effectively real-time. The potentially confusing factor is the synchronization signal. The synchronization signal ensures that the video camera, digitizer board and monitor all operate in phase at the same frequency. Deviations from full synchronization will distort the image, although small deviations may produce only slight waves.

Most image analysis systems can be synchronized in one of two ways: the digitizer board will attempt to synchronize to the camera, or the board will provide the synchronization signal that drives both the camera and the monitor. The former mode, of which there are several variations, requires a good RS-170 camera signal. In my experience, the variation known as digital synchronization can produce good image quality. However, for total image

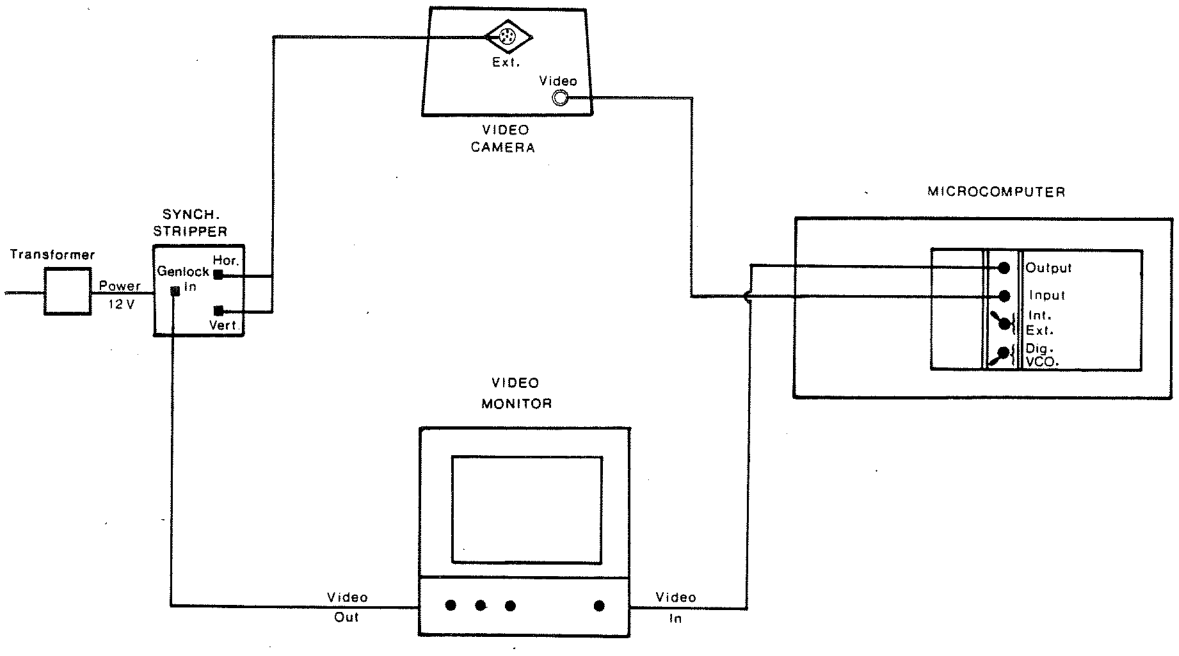

Figure 2. Schematic diagram of cable connections required for the image analysis system described in the text. Connections may differ with other system configurations. Note the orientation of the microcomputer, which faces backwards in order to demonstrate the ports on the digitizer boards.

stability, digitizer-driven synchronization (known as external Genlock) is to be preferred. The IAS described here is configured for external Genlock. Since the synchronization input port on the camera is designed for a composite video-synchronization signal, a "synch stripper" (described above) has been introduced between the digitizer board and the camera. However, there are cameras available (slightly higher-priced) which have a separate synchronization port. With such cameras, the synch stripper is not required.

The connections required for the complete IAS, including synch stripper, are presented in the schematic diagram of Fig. 2. Note that full assembly and configuration, including insertion of the digitizer board into the microcomputer, requires less than 10 minutes.

Software

One of the primary reasons for the substantial price advantage of this particular system over commercially available systems is the state of the software. Commercial systems generally come complete with ready-to-use software applications packages; this one does not. Software for this system comes in the form of a library of 74 image analysis functions (Coreco's Gray Library). These "black box" functions perform all of the basic, and many of the sophisticated procedures that one would ever use in an IAS. However, the final applications package requires programming in the C language, through which the required Gray Library functions are called up. Since the functions handle all of the IAS manipulations (which can be complex), the degree of C programming expertise that is required is minimal - a welcome feature to those who are not avid programmers. However, it is not possible to use the functions without a surrounding program.

The Gray Library provides all of the IAS functions required for programming. It also comes with an all-purpose Utility program, which is menu-driven and ready for immediate application to a variety of IAS situations. Where custom programs are being developed, the user will also require a compiler for the source code. Lattice C is the compiler recommended by Coreco for the Gray Library. Of course, the PC also requires DOS for operation, independent of any image analysis programs.

*Price - Gray Library - $800
- Lattice C compiler - $500

BASIC PRINCIPLES OF OPERATION

The basis of operation for all image analysis systems in the conversion of images into arrays of numbers - in other words, image digitization. This is accomplished at the digitizer board once the video signal has arrived from the camera. At the board, the image is treated as a grid of 512 columns x 480 rows, with each resulting square termed a pixel. The light intensity of each pixel is quantified and assigned to an integer scale of 0-255. Pixels with an intensity of 0 are black while those with an intensity of 255 are white. Pixels in the intermediate range are various shades of gray; hence the levels between 0 and 255 are known as gray levels. Since the number of gray levels displayed is under operator control, images can be viewed with as few levels as desired, even though the incoming video image has the full suite of gray levels. In the most extreme case, a threshold gray level can be defined, below which all pixels are displayed as black and above which all are white. Not surprisingly, the resulting black and white image is known as a binary image.

Each pixel stores its video information in a 1-byte word (1 byte= 8 bits =256 values); the resulting memory requirement is met in the 256 K RAM on the digitizer board. Despite the relatively large amount of data being stored, an entire image can be stored (and redisplayed on the video monitor) 30 times per second. Clearly, the term "framegrabber" is an appropriate one. It also demonstrates the real-time capability of the system.

While it is possible to use all 256 gray levels in digitizing an image, the 8th bit of each 8-bit pixel word is generally reserved as a "graphics plane". Use of the graphics plane enables the display of graphics features (such as cursors, windows, etc.) with the brightest possible gray level. Since the upper 128 gray levels contribute little to our visual impression of an image, use of the graphics plane has a negligible impact upon image quality. Accordingly images are generally displayed with 128 gray levels and a graphics plane.

The flexibility and power of an image analysis system becomes more clear when thought is given to the basic underlying processes. The system converts an image into an array of numbers, with each position in the array representing a pixel in the image, and each numerical value representing a gray level for that pixel. Therefore, anything that can be done with a matrix of numbers can be done to an image. For example, creating the inverse of an image is merely a matter of subtracting each pixel's gray level from 256 (or 128 as the case may be). Image contrast can be doubled by doubling all gray level values. Since the result of a matrix manipulation is seen immediately on the monitor, image manipulations can be as interactive or as automated as desired.

IMAGING CAPABILITIES

The capabilities of the image analysis system described here can be summarized as those associated with image manipulation and control. Applications progams (discussed in a later section) usually incorporate a number of these capabilities. While the features discussed below are characteristic of the Gray Library, they are generic in nature and found in most systems. What follows is the most cursory examination of imaging capabilities; for a more thorough treatment, the reader is referred to other works (Ballard and Brown 1982; Gonzalez and Wintz 1977; Hall 1979).

The most fundamental procedure of any IAS application is the image digitization, or grab. Grabs can be performed in continuous mode for real-time (30/second) viewing, or made individually. Virtually all IAS processes work with a grabbed (frozen) image. However, a variety of image manipulations are possible while in continuous grab mode; examples would include the display of reversed, high contrast, reversed high contrast and binary images. In most applications, the resulting image would be adjusted for optimum focus, contrast, etc. and then grabbed for subsequent processing.

Image control is possible at both the pixel and whole-image level, as well as the intermediate level of image windows. Simple examples of pixel control would include a reading of gray level (light intensity) at any user-defined coordinate or the display of graphics characters at any location on the video monitor. Image control is exemplified by the duplication and reduction of a grabbed image, and its subsequent display on the monitor beside the original.

Image enhancement is one of the most important and widely-used features of an IAS. A high contrast display can be viewed in real-time. However, simple procedures allow the operator to subtract an image background from the entire image, average several noisy images or use high or low-frequency filters to add or remove detail. Gray level expansion, whereby the gray levels in a poorly-contrasted image are spread out over all 128 (or 256) levels, can bring out detail that is totally invisible to the unaided eye. All of these enhancement procedures are effective because of the limited capability of the human eye - differentiation of 128 gray levels is well beyond our visual capacities.

Alternate means of image enhancement operate on the spatial gradients in gray level that are present. Edge detection procedures highlight objects by producing binary images where edges are white and everything else is black. Convolutions operate in an almost inverse manner, eliminating noise and smoothing contours. Convolutions, which are essentially two-dimensional running means of image brightness, are particularly useful in the interpretation of noisy images.

While increased ease of visual interpretation may be the sole objective of a particular IAS application, other applications rely upon the digital nature of the image for quantification. Examples would include procedures where the number of pixels at each gray level are counted to provide a measure of image brightness. Such procedures are particularly useful in binary images, in which objects are black and the background is white, thus allowing for object quantification. Intensity profiles across a user-defined axis provide a graphical representation of image brightness. A totally different form of quantification is provided by a function which displays the optical Fourier transform of an image (this is one of the few procedures that takes minutes rather than seconds to run).

While not strictly an IAS procedure, the system described here also allows for storage of images and image windows (segments of images) in memory. A full image requires 256K bytes of disk space for storage, so an image archive would require large amounts of memory. However, images can also be stored in their run-length encoded form (a compressed format for image storage, suitable only for binary images), which requires considerably less memory. Of course, image windows require even less memory.

The most sophisticated of the image analysis functions are those associated with object identification and classification. All of them operate only on binary images. During processing, each group of contiguous dark pixels is classified as an object. Once so classified, other functions can provide counts of the number of objects in the image, as well as their geometric features (length, width, area, perimeter, number of holes, etc). Note that both the identification and measurement procedures are fully automated, although it is possible to interactively select the objects of interest for further processing.

At the apex of this group of "smart" functions are those dealing with object recognition. In this mode, it is possible to train the system to recognize various classes of objects, based upon the appropriate geometric characteristics. Once the training has been completed, the system can be used to automatically categorize, count and measure each object in a field of view. Such capabilities are well beyond those of all but the most expensive systems of a few years ago.

A summary of the IAS procedures available from this system is presented in Table 1. Note that the procedures available on the Utility program are menu-driven and interactive, allowing for immediate and real-time use. The remaining functions must be called up from a user-written program.

APPLICATIONS

Many of the potential applications of this IAS are readily extrapolated from the list of capabilities in Table 1. All are suited to video images of any kind, not just those derived from a microscope. However, microscope-based observations generally take advantage of certain operations more than others. Enhanced image contrast is the most obvious of these applications especially since, as mentioned previously, details that would be invisible to the unaided microscope user can be made clear with an IAS. The increased size of the video monitor image also facilitates microscopic viewing.

Many microscope-based observations could take advantage of a general-purpose application package designed for interactive linear measurements, either of individual objects or of periodic features such as scale annuli or otolith daily growth increments. The advantages of this type of measurement over microscope-based observations include:

Table 1. A summary of image analysis capabilities available from this (and many other) systems. Those marked with * are available for real-time use in the Utility program; others must be incorporated from the Gray Library into user-written programs.

Grab Mode*	Pixel Control	Image Control	Image Enhancement
- normal	- read or write individual pixels*	- image reduction	- background subtraction
- reverse	- access adjacent pixels	- zoom*	- edge detection*
- high contrast		- add	- histogram (=gray level) expansion*
- reverse high contrast		- average	- convolutions
		- subtract	- high and low pass filters
			- control of number of gray levels*

Image Quantification	Image Storage*	Graphics Plane Control	"Smart" Functions
- optical fast Fourier transform and inverse	- image/window to buffer or disc	- cursor*	- run-length encoding
- intensity profile*	- memory to display	- target*	- object counts and highlighting
- pixel count		- box*	- contour encoding
		- test pattern	- calculation of geometric features of object
		- alphanumerics	- contour fill
			- object recognition and classification

a) enhanced contrast

b) ease of measurement - a target under cursor-arrow control is easier to position on a video monitor than is an ocular micrometer.

c) resolution - distances of less than 1µm are virtually impossible to measure with an ocular micrometer. However, interpixel distances of 0.16 µm are possible with the IAS.

d) data collection - measurements can be stored directly in memory eliminating the need (and error potential) of handwritten transcription.

As an example of this type of program, the source code for a program designed to measure periodic features on an otolith has been appended (Appendix 1)[1]. The program prompts the user at every stage of the procedure, beginning with prompts for sample information such as collection date, fish length, sample number, etc. The next step is menu-driven, allowing the user to enhance the image appropriately before beginning data collection. Through use of a keyboard-controlled target/cursor visible on the video monitor, the user next moves the cursor sequentially to each of the features of interest, digitizing their coordinates with the stroke of a key. When applied to a microstructural preparation of an otolith, the program allows the operator to move the cursor quickly and accurately from one daily increment to another, with the increment widths calculated automatically upon digitization. Each measurement is calibrated to the unit of choice. The X-Y coordinates of the cursor are continuously updated and displayed on the PC monitor, along with a running count of increments digitized and the coordinates of the last-digitized increment. Upon completion, the user has the option to digitize the start and end points of the measurement radius, in the event that increment widths must be standardized to a pre-defined otolith radius. The program ends by storing the number of increments digitized, the increment width data, otolith radius and sample information in a single-record DOS file for subsequent analysis. The above program is ideally suited for measuring up to 200 otolith increment widths, or, on a larger scale, distances between scale/otolith annuli. Both types of measurements are commonly used in back-calculating growth rates or size at age. While this particular program has been tailored to the author's needs, those with a modicum of C knowledge would require very little time to modify the program for their own measurement purposes.

Where more complex measurements are required, and particularly when there is a variety of distinct objects in the field of view, an automated measurement scheme may be preferable. An example of such a program is presented in Appendix 2[1]. This program is suited for histological examinations, such as area measurements of oil globules, and is ideal for certain types of ichthyoplankton samples. The latter would include automated counts and measurements of large numbers of sorted fish eggs. Acting upon prompts, the operator first converts the image into a binary image with an appropriate threshold. Automatic output after that point includes a count of the total number of objects and sequential highlighting of each object in turn. Any or all of the objects of interest are further processed to provide 11 geometric measurements, including area, number and area of holes, perimeter, coordinates of the centroid, angles and length of the major and minor axes, and dimensions of the smallest box that would enclose the object. All measurements are calibrated to the unit of choice. It should be noted however, that this and analogous programs are sensitive to the threshold selected for the binary image: the threshold must be high enough to ensure that all objects to be measured are completely separated from each other.

A surprisingly simple extension of the above program can teach the system to recognize and classify objects, based upon the mean and variance of some (or all) of the geometric measurements. Again, such an application is well suited to samples of sorted fish eggs, at least where species can be characterized solely on the basis of diameter, oil globules, etc. Note however, that automatic identification of larvae and zooplankton can be difficult due to the variety of postures that can be present; a human can recognize if an appendage is curled up beneath the body, but complex programming would be required to teach that to an IAS.

An increasingly popular technique for stock identification these days employs Fourier

---

1 - Upon request, the author can provide this source code on a floppy disc. Note that the user must compile this code in conjunction with the Gray library to make it operative

analysis to convert scale or otolith shapes into mathematically-tractable values. The IAS system described here was originally configured for that purpose. Note also that manual tracing (with, for instance, a mouse) is not required with this system.

A whole host of other biological applications exist for image analysis systems. While they will not be detailed here, it is useful to note the existence of some prominent examples:

a) Template matching - While more commonly used in industrial quality control, biological applications are possible where a "standard object" exists. The system can be programmed to alert the user when the unknown object and the standard are sufficiently different.

b) Lipid concentration in larval scallops - After application of an appropriate stain, lipid concentration can be estimated through pixel counts of a binary image.

c) Electronic cut and paste - Images and sections of images can be reduced, zoomed, moved and added to other images at will. Image copies retain all of the quality and detail of the original. This application is particularly useful in preparation of AV aids and publications.

d) Age determination - All of the automated and semi-automated fish age determination techniques on the market rely on a process similar to the intensity profile described earlier. After a profile of an enhanced otolith/scale image is taken, algorithms interpret the amplitude of the profile in terms of the presence/absence of annuli.

e) Morphometrics - With or without a microscope, fish can be centred under a video camera and the image grabbed for subsequent morphometric measurements. Of course, all measurements can be recorded automatically in memory.

f) False-colour images - False-colour images can be helpful in image interpretation in select instances. The IAS discussed here does not have colour capabilities. However, adaptation for colour is not a major modification, at least from the point of view of software. All colour image analysis systems apply colour to designated gray levels; therefore their operation is conceptually similar to monochrome systems. Of course, a colour system requires the purchase of a colour camera and monitor.

While the potential applications of the IAS are numerous, it is also important to note the limitations. Most important of these is the inability to handle remote sensing (satellite) data; not only is the data flow too large, this is one application where colour imaging is desirable. A second limitation is that associated with preparing printed images. The latter are possible with MX-80 printers, but are slow to produce and restricted to 16 gray levels. High quality Laser printer output should be possible, but this has not been attempted for this system. A final constraint concerns the amount of memory required for image storage; image archives are quite possible, but at 256K bytes per image, become major memory users if stored without run-length encoding.

SUMMARY

Image analysis systems provide a substantial advantage over unaided microscopic observations, both in terms of image enhancement and quantification of the features of interest. Most of the microcomputer-based systems now available provide enhancement, manipulation and quantification capabilities. The advantages of the system outlined here are primarily those of price and flexibility; substantial savings over commercially-available packages are possible if the researcher is willing to configure his own system. This paper outlines the ease with which this can be done, resulting in a state-of-the-art, high resolution system with real-time processing. The system has the capability of being as interactive or as automated as desired, and can perform macroscopic image analysis as well. It can also serve as a stand-alone PC without modification. The disadvantage is the requirement for some basic programming in the C language, although the image analysis functions are provided. While there is no question that the pre-packaged applications software available commercially is more convenient, this convenience is a two-edged sword: what one gains on one hand, one loses in terms of program and hardware flexibility. The intention of this paper is to provide enough of an overview of a particular image analysis configuration to allow an informed decision on whether or not it may benefit perceived needs.

LITERATURE CITED

Ballard, D.H. and C.M. Brown. 1982. Computer vision. Englewood Cliffs. Prentice - Hall, N.J. 523 pp.

Gonzalez, R.C. and P. Wintz. 1977. Digital image processing. Addision - Wesley Publ. Co., Don Mills: 431 pp.

Hall, E.L. 1979. Computer image processing and recognition. Academic Press, NY: 584 pp.

ACKNOWLEDGEMENTS

I thank D. Heffler for having introduced me to the Oculus board, and B. Topliss and L. Payzant for an overview of remote sensing applications. R. Stronach of Sigma Electronics provided welcome technical assistance with the system. P. Hurley provided helpful comments on the manuscript. Thanks also to J. Simon for having drafted the figures.

APPENDIX I

```
#include "math.h"
#include "oc200.h"
#include "stdio.h"

int _stack=64500;
main (argc,argv)                    /* prompt for targeted data file */
int argc;
char *argv[];
{
/********************* Variable declaration ****************************/

int i, gg, lut, setnum, fishnum,slide, age, z, nx, ny;
int ix[200], iy[200], curs;
register int x, y;
char ch, ch2, resp, res, cr, lf, spec[4];
float tl, npdist;
long int date;
double npx, npy, px, py, magpn;
double calib, xxx, yyy, incwid[200], sumiwid, npcal;
FILE *fp;
if (argc != 2) {
    printf("You forgot to enter the targeted data file name \n");
    printf("on the command line\n");
    exit(0);
                }
screen(0);
printf("/*********************************************************************/
printf("/*                               INCWIDTH                             */
printf("/*  Program to digitize and store widths of otolith daily increments  */
printf("/*                       Version 1.2 - 18/08/87                       */
printf("/*                        by Steven E. Campana                        */
printf("/*********************************************************************/

/********************** Variable initialization ************************/
x=y=250;
tl=npdist=date=xxx=yyy=sumiwid=i=gg=lut=0;
setnum=fishnum=slide=age=z=nx=ny=magpn=npx=npy=px=py=0;
npcal=calib=6.26;                   /* pixel to um conversion at 1250X */
ch=ch2='x';
cr='\r';
lf='\n';
for (z=0;z<200;z++) incwid[z]=ix[z]=iy[z]=0;
curs=3;
grdeflut();
grclo(100);
grtarg(x,y);
/********************** Enter header information **********************/
printf ("Enter species (3 letters)\n");
scanf ("%s", spec);
printf ("Enter set number\n");
scanf ("%d", &setnum);
printf ("Enter fish number\n");
scanf ("%d", &fishnum);
printf ("Enter slide number\n");
scanf ("%d", &slide);
```

```
printf ("Enter TL\n");
scanf ("%f", &tl);
printf ("Enter date of collection (ddmmyy)\n");
scanf ("%ld", &date);
printf ("Enter age\n");
scanf ("%d", &age);
printf ("Select procedure from menu\n");
printf ("Use cursor arrows to move cursor\n");

/********************** Menu for Procedures ******************************/
  printf ("          Menu Selection\n");
  printf (" \n");
  printf ("C: set cursor increment (default=3)\n");
  printf ("F: finished\n");
  printf ("G: grab image\n");
  printf ("I: digitize increment coordinates\n");
  printf ("M: reset magnification calibration (Default=100X objective)\n");
  printf ("N: digitize nuclear coordinates\n");
  printf ("P: digitize peripheral coordinates\n");
  printf ("R: repeat last digitization on present image\n");
  locate (1,1);
  for (z=0;z<13;z++) printf ("                                  \n");
do
  {
  ch=getch();
  switch (ch)
     {
     case '\0':                    /* call for cursor */
          ch2=getch();
          switch (ch2)
                 {
                 case  72:                    /* cursor up */
                       gretarg(x,y);
                       y=y-curs;
                       grtarg(x,y);
                       break;
                 case  80:                    /* cursor down */
                       gretarg(x,y);
                       y=y+curs;
                       grtarg(x,y);
                       break;
                 case  75:                    /* cursor left */
                       gretarg(x,y);
                       x=x-curs;
                       grtarg(x,y);
                       break;
                 case  77:                    /* cursor right */
                       gretarg(x,y);
                       x=x+curs;
                       grtarg(x,y);
                       break;
                 default:
                       locate(1,1);
                       printf("Keystroke not recognized\n");
                       break;
```

```
              }                                     /* end of cursor switch*/
        locate (1, 60);
        printf ("Cursor x: %d  \n", x);         /* indicate cursor location */
        locate (2,60);
        printf ("Cursor y: %d  \n", y);
        break;
case 'c':
case 'C':
        locate(1,1);
        printf("Enter cursor increment\n");
        scanf ("%d", &curs);
        locate(1,1);
        printf("                        ");
        break;
case 'f':
case 'F':                           /* finish */
        break;
case 'g':
case 'G':                           /* grab image */
        locate (1,1);
        printf ("Select lut: 0-Normal 1-Reverse 2-HC 3-Reverse HC\n");
        scanf ("%d", &lut);
        printf ("Hit key to grab image\n");
        grgrab (lut, 0, 0, 0);
        resp=getch();
        if (resp='\0') resp=getch();
        locate(1,1);
        for (z=0;z<3;z++) {
          printf("                                                   \n");
          }
        locate(1,1);
        printf ("Redigitize last increment\n");
        printf("Note that calibration default is set at 100X (use 'M' to chang
        gg=i;
        break;
case 'm':
case 'M':                     /* set pixel-measurement unit conversion */
        locate(1,1);
        printf("Enter calibration coefficient (no. of horiz. pixels per unit m
        scanf("%lf", &calib);
        locate(1,1);
        for (z=0;z<2;z++) {
        printf("                                                     \n"
                  }
        locate(1,1);
        break;
case 'i':
case 'I':                  /* calculate increment width */
        ++i;
        if (gg == i-1 && i>1) i=i-1;    /* note that this makes ix and iy */
                                        /* arrays somewhat inaccurate */
        ix[i-1]=x;
        iy[i-1]=y;
        update:                           /* update digitization display */
        locate (3,60);
```

```
          printf ("X of last inc: %d  \n", x);
          locate (4,60);
          printf ("Y of last inc: %d  \n", y);
          locate (5,60);
          printf ("Increment number: %d  \n", i);
          if (i>1 && gg!=i){           /* checks to see if this is 1st dig */
                           xxx=ix[i-1] - ix[i-2];       /* after grabbing */
                           xxx=xxx*xxx;
                           yyy=((double) (iy[i-1] - iy[i-2]))*0.8;
                           yyy=yyy*yyy;
                           incwid[i-2]=(sqrt(xxx+yyy))/calib;
                           sumiwid=sumiwid+incwid[i-2];
                           }
          else --gg;
          break;
     case 'n':
     case 'N':                          /* nuclear coordinates */
          nx=x;
          ny=y;
          break;
     case 'p':                          /* peripheral coordinates */
     case 'P':
          px=x;
          py=y;
          locate(1,1); .
          printf("Enter calibration coefficient (no. of horiz. pixels per unit m
          printf ("for this objective \n");
          scanf ("%lf", &npcal);
          locate(1,1);
          for (z=0;z<2;z++) {
 printf("                                                          \n");
                            }
          locate(1,1);
          break;
     case 'r':                         /* repeat last dig without new grab */
     case 'R':
          ix[i-1]=x;
          iy[i-1]=y;
          goto update;
     default:
          locate(1,1);
          printf ("Unrecognized keystroke\n");
          break;
     }                                 /* end of switch */
  } while (ch != 'f' && ch != 'F');    /* end of do-while */
 npx=(double) nx-px;           /* calculate nucleus to periphery distance */
 npy=(double) (ny-py)*.8;
 xxx=npx*npx;
 yyy=npy*npy;
 npdist=(sqrt(xxx+yyy))/npcal;
 fsave:                                 /* transfer data to file */
 locate(1,1);
 if ((fp=fopen(argv[1],"a")) == NULL) {
    printf("Cannot open file\n");
    exit(0);
```

```
                                                     }
fprintf(fp, "%s %d %d %d %4.1f %ld %d %d %7.2f %7.2f ", spec,setnum,
        fishnum,slide,tl,date,age,i,sumiwid,npdist);
for (z=0;z<i-1;z++) fprintf(fp,"%5.2f ", incwid[z]);
fprintf(fp, "%c%c", cr,lf);
fclose(fp);
printf("DOS file = %s\n", argv[1]);
}
```

APPENDIX II

```
#include "stdio.h"
#include "oc200.h"

int _stack=64500;
int matrix[480][32];
main()
{
/****************** Variable declaration *****************************/
register int kk, k, cc;
int lastpg, nowpg, pg, numanc, numdes, anc1, anc2, des1, des2;
int xx, yy, curs;
int y3, x3, *line, *start, *length;
int imax, maxnb, nobj;
int *object, *fgrp, *next, *panc, *pdes, *anc, *des;
int mode, thresh, lut;
int i, xlen, ylen, x1, x2, y1, y2, xword, xadj;
int xmin,xmax,ymin,ymax,y;
int mask, er, respon, error;
int *jline, *jstart, *jlen, lastj;
int nfeat[12];
float feat[12], calib;
char *malloc();
char ch, ch2;
unsigned int nbyte;

screen(0);
printf("/*********************************************************************/
printf("/*                            FEATURES                               */
printf("/*         Program to automatically count and measure objects        */
printf("/*                       Version 1.1 - 18/08/87                      */
printf("/*                        by Steven E. Campana                       */
printf("/*********************************************************************/

/*-------------------------Memory Allocation----------------------*/
maxnb=15000;
nbyte=maxnb*2;

next=(int *)malloc(nbyte);
if (next == NULL) {er=1; goto end;}
object=(int *)malloc(nbyte);
if (object == NULL) {er=2; goto end;}
panc=(int *)malloc(nbyte);
if (panc == NULL) {er=3; goto end;}
pdes=(int *)malloc(nbyte);
if (pdes == NULL) {er=4; goto end;}
line=(int *)malloc(nbyte);
if (line == NULL) {er=5; goto end;}
start=(int *)malloc(nbyte);
if (start == NULL) {er=6; goto end;}
length=(int *)malloc(nbyte);
if (length == NULL) {er=7; goto end;}
fgrp=(int *)malloc(nbyte);
if (fgrp == NULL) {er=8; goto end;}
jline=(int *)malloc(nbyte);
if (jline == NULL) {er=9; goto end;}
```

```
jstart=(int *)malloc(nbyte);
if (jstart == NULL) {er=10; goto end;}
jlen=(int *)malloc(nbyte);
if (jlen == NULL) {er=11; goto end;}

nbyte=maxnb*3*2;
anc=(int *)malloc(nbyte);
if (anc == NULL) {er=9; goto end;}
des=(int *)malloc(nbyte);
if (des == NULL) {er=10; goto end;}

/*--------------------- Prepare binary image --------------------------*/

screen(0);
mode=0;
thresh=50;
grclo(100);
grdeflut();
ch=ch2='x';
do
    {
    printf ("Image: 0-Normal 1-Reverse 2-HC 3-Reverse HC\n");
    scanf ("%d", &lut);
        if (lut == 0 || lut == 2) mode=0;
        else mode=1;
    printf ("Use > and < to increase/decrease threshold\n");
        printf("Type T when threshold selected\n");
        do {                          /* loop for binary threshold control */
           grgrab (lut,0,0,0);
           ch=getch();
           switch (ch)
                  {
                  case ',':
                  case '<':
                       thresh=thresh-5;
                       grthrs (lut,thresh,mode);
                       break;
                  case '.':
                  case '>':
                       thresh=thresh+5;
                       grthrs (lut,thresh,mode);
                       break;
                  case 't':
                  case 'T':
                       break;
                  default:
                       printf("Unrecognized keystroke\n");
                       break;
                  }                          /* end of switch */
           } while (ch != 't' && ch != 'T');  /* end of do-while */
           printf ("Enter 'a' to adjust, any other key to continue\n");
           ch=getch();
        } while (ch == 'a' || ch == 'A');      /* end of do-while */

/*--------Window development in preparation for object encoding -------------*/
```

```
screen(0);
x1=y1=50;
x2=y2=450;
grbox(x1,y1,x2,y2);                     /* draw default box */
printf ("Is window OK? Y/N\n");
ch=getch();
if (ch == 'n' || ch == 'N') {           /* prepare own window */
grebox(x1,y1,x2,y2);
ch=ch2='x';
xx=yy=x1=x2=y1=y2=250;
curs=20;
grtarg(xx,yy);
printf ("Use cursor arrows to move monitor cursor\n");
printf("Type U to mark upper left corner of desired window\n");
printf("Type L to mark lower right corner of desired window\n");
printf ("Type 'F' to finish\n");
do                                  /* interactive loop for box prep */
  {
  ch=getch();
  switch (ch)
     {
     case '\0':                     /* call for cursor */
     ch2=getch();
          switch (ch2)
             {
             case  72:                      /* cursor up */
                   gretarg(xx,yy);
                   yy=yy-curs;
                   grtarg(xx,yy);
                   break;
             case  80:                      /* cursor down */
                   gretarg(xx,yy);
                   yy=yy+curs;
                   grtarg(xx,yy);
                   break;
             case  75:                      /* cursor left */
                   gretarg(xx,yy);
                   xx=xx-curs;
                   grtarg(xx,yy);
                   break;
             case  77:                      /* cursor right */
                   gretarg(xx,yy);
                   xx=xx+curs;
                   grtarg(xx,yy);
                   break;
             default:
                   printf("Use arrows only\n");
                   break;
             }                                     /* end of cursor switch */
      locate (1, 60);
      printf("Cursor x: %d  \n", xx);    /* update cursor location display */
      locate(2,60);
      printf("Cursor y: %d  \n", yy);
      break;
```

```
    case 'u':
    case 'U':                           /* fix upper left box corner */
          grebox(x1,y1,x2,y2);
          x1=xx;
          y1=yy;
          grbox (x1, y1, x2, y2);
          break;
    case 'l':
    case 'L':                           /* fix lower right box corner */
          grebox(x1,y1,x2,y2);
          x2=xx;
          y2=yy;
          grbox(x1,y1,x2,y2);
          break;
    case 'f':
    case 'F':                             /* box finished */
          grebox(x1,y1,x2,y2);
          gretarg(xx,yy);
          break;
    default:
          printf("Unrecognized keystroke\n");
          break;
    }                                         /* end of switch */
  } while (ch != 'f' && ch != 'F');        /* end of do-while */
                           }            /* end of if */
else grebox(x1,y1,x2,y2);

/*----------- Fine-tune window prep for object encoding-----------*/

xadj=xword=xlen=ylen=0;
xlen=x2-x1+1;                   /* from the box coordinates   */
ylen=y2-y1+1;                   /* we calculate dimensions    */
if (ylen &1) ylen-=1 ;          /* test if number is even     */
                                /* adjust if not              */
xword=(int)xlen/16;             /* for encoding in the matrix*/
xadj=xlen-xword*16;             /* xadj is the number of emp-*/
if (xadj!=0) xword+=1;          /* ty bytes in the leftmost   */
                                /* word to be filled with 1   */
if (xword>32) xword=32;
x3=xword-1;
xadj=xword*16-xlen;
y3=ylen-1;
grwindo (matrix,x1,y1,xword,ylen);/* load the matrix */

if (xadj!=0)                    /* mask the unused bit with   */
   {                            /* 1 in the window            */
   mask=0xffff ;
   mask= mask >> (16-xadj) ;
   for (y=0; y<=y3; y++)
       matrix[i][xword]=matrix[i][xword] | mask;
   for (y=y3+1; y<=2*y3+1; y++)
       matrix[i][xword]=matrix[i][xword] | mask ;
   }
/*--------------Encode objects in window ---------------------*/
screen(0);
```

```
locate(1,1);
printf ("Encode and link in progress...\n");
er=grencod(matrix,y3,x3,line,start,length,&imax,maxnb);
if (er) goto end;
er=link(line,start,length,object,fgrp,next,imax,panc,pdes,anc,des,&nobj);
if (er) goto end;
printf ("Completed...\n");
printf ("Number of objects=%d\n",nobj);
printf("Enter calibration coefficient (no. of horiz. pixels per unit measure)\n\
scanf("%f", &calib);
screen(0);
                                                 /* select object of interest */
for (k=1;k<=nobj;++k) {
    printf("Object %d: \n", k);
    printf("Strike key to end display \n");
    grcurs(x1,y1,line,start,length,fgrp[k],next,&xmin,&xmax,&ymin,&ymax);
    printf ("Enter 'C' to continue to next object\n");
    ch=getch();
    if (ch == 'c' || ch == 'C') continue;
    nfeat[0]=11;
    for (kk=1;kk<12;kk++) nfeat[kk]=kk;
    er=features (fgrp[k],line,start,length,next,panc,pdes,anc,des,nfeat,feat,0.8
    if (er) goto end;
    printf ("feature # 1:area including holes: %f\n",feat[1]/(calib*calib));
    printf ("feature # 2:number of holes: %f\n",feat[2]);
    printf ("feature # 3:total area of holes: %f\n",feat[3]/(calib*calib));
    printf ("feature # 4:outer perimeter: %f \n",feat[4]/calib);
    printf ("feature # 5:x coord of centroid: %f\n",feat[5]/calib);
    printf ("feature # 6:y coord of centroid: %f\n",feat[6]/calib);
    printf ("feature # 7:angle of major axis w.r. to horiz.: %f \n",feat[7]);
    printf ("feature # 8:length of major axis: %f\n",feat[8]/calib);
    printf ("feature # 9:length of minor axis: %f\n",feat[9]/calib);
    printf ("feature #10:width of smallest box enclosing: %f\n",feat[10]/calib);
    printf ("feature #11:height of smallest box enclosing: %f\n",feat[11]/calib)
    for (k=1;k<3;k++) printf("                      \n");
    printf("Enter 'C' to continue or 'F' to finish\n");
    ch=getch();
    if (ch == 'c' || ch == 'C') continue;
    else break;
                         }               /* end of for */
/*********************************************************************/
end:
if(er)                                           /* error summary */
   {
   printf("error=%d\n",er);
   while(!kbhit());
   printf("Strike key to continue\n");
   respon=getch();
   }

/*--------------------- deallocation of dynamic memory ----------------------*/

if (free((char *)next) != 0)
   sound (200,100);
if (free((char *)object) != 0)
```

```
   sound (200,100);
if (free((char *)panc) != 0)
   sound (200,100);
if (free((char *)pdes) != 0)
   sound (200,100);
if (free((char *)line) != 0)
   sound (200,100);
if (free((char *)start) != 0)
   sound (200,100);
if (free((char *)length) != 0)
   sound (200,100);
if (free((char *)fgrp) != 0)
   sound (200,100);
if (free((char *)anc) != 0)
   sound (200,100);
if (free((char *)des) != 0)
   sound (200,100);
if (free((char *)jline) != 0)
   sound (200,100);
if (free((char *)jstart) != 0)
   sound (200,100);
if (free((char *)jlen) != 0)
   sound (200,100);

}
```