Implementation of Automatic Focusing Algorithms for a Computer Vision System with Camera Control

John F. Schlag, Arthur C. Sanderson,
Charles P. Neuman and Francis C. Wimberly

CMU-RI-TR-83-14

Department of Electrical Engineering
Robotics Institute
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

15 August 1983

This work was supported in part by the Westinghouse Electric Corporation and The Robotics Institute, Carnegie-Mellon University.

Abstract

The POPEYE system is a grey level computer vision system developed at CMU for research and development. The system provides a convenient environment for research by coupling a powerful microprocessor with a large base of support software. The particulars of the system's hardware configuration and software support are given after an explanation of the desires which motivated its fabrication.

In addition to providing general computation and display capabilities, the POPEYE system provides open loop manual or software control over the camera parameters of pan, tilt, focus and zoom. This offers many advantages over fixed arrangements, such as the ability to investigate focusing and elementary tracking algorithms.

Automatic focusing has been implemented before and has been found to be a particularly useful camera accommodation. In these cases, however, the approach was to implement a single algorithm. This work describes the implementation of several standard automatic focusing algorithms on the POPEYE system and provides experimental evaluation and comparison. This leaves the POPEYE system with a valuable enhancement and provides a starting point for the implementation of a production focusing system. There are many possible uses for such a system, including robotic assembly and inspection tasks.

One of the applications to which the POPEYE system has been applied is the development of industrial inspection algorithms for the Westinghouse-CMU Factory of the Future Project. Part of this project involved the inspection of fluorescent lamp mount assemblies. Algorithms for the automated inspection of the assemblies are described which represent the solutions to difficult inspection problems currently beyond the capabilities of commercial vision systems.

Suggestions for the implementation of a production focusing system are given along with suggestions for possible hardware improvements to the POPEYE system.

Table of Contents

List of Figures

Chapter 1
Introduction

This chapter introduces the project, enumerates its original objectives, gives background material relevant to the applications and provides an overview of the rest of the document.

Computer vision is a wide field of research which few others match for diversity and richness of subject matter. Because of the potential payoffs, the level of funding and subsequently the amount of research underway in vision is huge. On one level, the need for intelligent vision systems for robotic applications is acute. Universities and laboratories currently conducting computer vision research are inundated with requests from industry for quicker, more reliable and more intelligent systems. On another level, the people conducting vision research clamor for a convenient environment in which to test new algorithms. They rightly wish to be able to move from conception to demonstration of new ideas in as little time as possible. This report describes POPEYE, a grey-level computer vision system developed at CMU with exactly this goal in mind. The system itself is described first. Then, several automatic focusing algorithms are presented and their implementations are described. In addition, the solutions to two industrial inspection problems to which the system has been applied are presented.

1.1. Project Objectives

Several considerations motivated the construction of the new vision system. First, there was the desire to be able to conduct experiments quickly and conveniently; little time should be spent on setting up and taking down apparatus. A recent example of this involved the comparison of theoretical predictions of surface shading with actual data obtained from a TV camera. Within the space of an hour, data were obtained which both validated some parts of the theory and suggested revisions to others.

Second, there was the issue of processing speed. Although elaborate frame buffer systems connected to general purpose mainframe computers offer more flexibility and more features than

personal workstations, they are often too slow to be considered interactive. Again, the emphasis is on minimizing mind-to-screen time. The POPEYE system dedicates a Motorola 68000 microprocessor to a single user, so the system is limited by software rather than hardware. Software attention can be focused on the development of a powerful user interface rather than obscure techniques to compensate for slow hardware response.

Third, and perhaps most importantly, it was desired that test images be readily obtainable from real data, rather than relying on a stock database. Often, computer vision and image processing algorithms become subconsciously tuned to specific images for lack of input. The POPEYE system avoids this source of frustration and embarrassment.

In addition, there is always a market for demonstrating industrial inspection algorithms. One of the uses foreseen for the POPEYE system was as a testbed for concept demonstration. In fact, the system turned out to be ideal for this type of activity, since the development of inspection algorithms – typically a more high pressure activity than research – spurred software development, particularly in the area of user interface. These desires point directly to several specifications:

- GREY SCALE CAPABILITY. The problems which *can* be solved with binary vision *have* been. Most applications now demand grey scale capability to overcome inadequacies in image acquisition and to relax lighting constraints. Also, most vision research requires grey scale data.

- PROGRAMMABILITY. A system intended for use in a research laboratory is significantly different in its user interface from a system destined for the assembly line. The only way to achieve real versatility is to make the system completely programmable.

- SINGLE USER ENVIRONMENT. To make response as quick as possible, the system software should avoid balooning to full operating system proportions. A single user, single process environment is quick and uncomplicated.

- ACCESS TO A LARGER RESOURCE POOL. The system should do what it does best and let general purpose mainframes do what they do best: mass storage, hardcopy production, editing, compilation and local network communication.

- FABRICATION FROM OFF-THE-SHELF COMPONENTS. With the proliferation of small, cheap and reliable microcomputer peripherals, there is no reason to design new hardware.

To summarize, the project objectives were: (1) to fabricate a testbed system for computer vision research, (2) to implement and compare automatic focusing algorithms for the system, and (3) to demonstrate the usefulness of the system through applications. In particular, the applications involved the development of industrial inspection algorithms for the Westinghouse-CMU fluorescent lamp project.

1.2. Background

Interest in automatic focusing arose from several quarters. The flexible assembly project, another Westinghouse-sponsored endeavor, was (and is) aimed at economizing midsized fabrication runs where development of hard automation is unwarranted and the use of human labor is expensive and demeaning. An assembly station consisting of a rotary table and two Puma robots was constructed for concept demonstration. In a production environment, automatic focusing would be a powerful assett to such a station. Also, automatic focusing would be the icing on the cake of a computer vision system, making research activities more convenient. In general, automatic focusing is a useful tool, often taken for granted by humans.

Previous implementations of automatic focusing include those of Horn at MIT [14], who used a frequency domain technique based on the FFT, and Tenenbaum at Stanford [33], who used a gradient operator. In both cases, however, the desire was to implement something that worked, rather than comparing different algorithms.

The fluorescent lamp project officially began in 1981 as part of the research toward the factory of the future [35]. It was one of the first projects undertaken by the Robotics Institute and one of several sponsored by Westinghouse. The major goals of the project were to reduce shrinkage (losses), improve lamp quality, increase manufacturing flexibility and reduce direct labor costs. The use of computer vision systems was considered to be an integral part of acheiving these goals.

The work on industrial inspection algorithms presented in this document is an outgrowth of the work by Maddox [16], who used a binary vision system to detect and classify fluorescent lamp filament defects.

1.3. Overview

Chapter 2 describes the vision system in terms of its hardware and software. In addition to the standard pedantic description of its capabilities, a bit of historical information is given on how it came into being. Chapters 3 and 4 describe the application of the vision system to the problem of automatic focusing. Chapter 3 introduces the focusing problem and covers candidate approaches to its solution, while Chapter 4 details the experimental procedures used to test the focusing algorithms and presents the results obtained from the experiments. Chapter 5 describes the industrial applications which were explored using the system, and is completely self-contained. Chapter 6 concludes the report with an evaluation of the POPEYE system as a research tool, a set of suggestions for

implementing a production focusing system, another set of suggestions for hardware improvements to the vision system, and finally, a call for better sensors.

Chapter 2
The POPEYE Computer Vision System

This chapter gives the particulars of the POPEYE *computer vision system. Each of the major components of the system is described. The custom hardware built to support the work on automatic focusing and the software written to interface the hardware to the vision system are also described. Where necessary, characterization of the components whose action affects the software interface is given.*

2.1. Introduction

While the Master's Project Proposal reproduced in Appendix A was forming, it became apparent that some type of vision system would be needed. Three options were available.

- **Use the MIC vision module.** The MIC module is a binary vision system which can be trained to classify objects on the basis of various parameters such as length, width, area, perimeter, first and second moments and connectivity analysis results. It is typically used in assembly line applications where classification and rejection capabilities are required.

  There are two major limitations to the MIC module. First, it is a binary vision system, where all the pixels are either black or white. The applications described in the proposal clearly call for grey-scale capability. In addition, Maddox [16] had used the MIC module for another project and found its range of capabilities too limited. Second, the system is not programmable. The point of the project was to work *with* a system, rather than *around* it.

- **Use the Grinnell frame buffer.** The Grinnell frame buffer system is a 512 by 512 pixel color memory attached to a general purpose computer. It is used mostly for off-line image processing applications such as satellite imaging and power spectrum computation. Its major problem is lack of speed. Anywhere from a few seconds to several hours are necessary to perform the image processing, depending on the complexity of the algorithm. Display of the results takes between ten and twenty seconds. Only in the cases of simple algorithms can this be considered an interactive system. In addition, the load imposed on the rest of the computing public is large.

- **Develop a new system.** A vision system with the negation of the above parameters was desired: grey-scale capability, programmability, speed, ease of use and low host com-

puter load. Normally, undertaking the development of a completely new system to support a project of this type would have been suicide. As part of other activities, however, the hardware talent at CMU had developed a MULTIBUS processor board based on the Motorola MC68000, a small but powerful 16/32 bit microprocessor chip.[1] The processor board was being inserted into small systems with an off-the-shelf memory board and the combination being billed as a powerful but inexpensive processor for special purpose control applications. Sanderson [26] had conceived the idea of using such a system as the basis for a computer vision system. In short, the technology was all there. Only system configuration and software development [7] were necessary.

2.2. The Basic System

A detailed description of the POPEYE system's hardware and software, including manufacturer names, product numbers, specifications and projected future enhancements is given in reference [6]. The purpose of this section is to provide a brief summary of the system's general capabilities.

2.2.1. Hardware Configuration

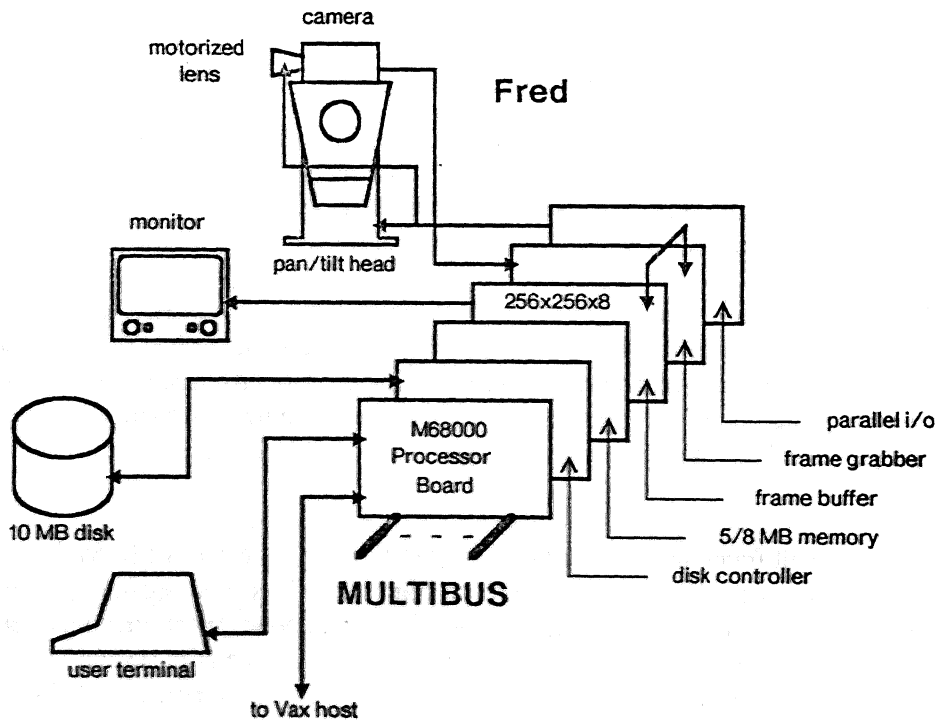


Figure 2-1: The configuration of the image processing system.

The hardware configuration of the POPEYE system as it existed during this project is shown in Figure 2-1. The principle components were as follows.

[1] At the time, there was no board level product based on this device available.

Processor Board	provides general purpose processing capability and controls all the peripheral devices. Contains two serial lines which connect to the user terminal and the host computer.
Memory Boards	provide 5/8 MB of semiconductor memory.
Disk Controller	interfaces to a 10 MB Winchester hard disk for mass storage of images and programs.
Frame Buffer	holds one 256 by 256 by 8 bit pixel image and continuously displays the contents on a black and white television monitor.
Frame Grabber	digitizes an image from the television camera and places the output into the frame buffer.
I/O Board	provides a path for communication between the processor and the lens controller interface which controls the camera parameters.

During this project, the only pieces of custom hardware in the POPEYE system were the processor board and lens controller interface (section 2.3.1). The processor board is currently being replaced by a similar commercial product, now that one is available. The lens controller interface was intended to serve only temporarily.

2.2.2. Software Support

The software for the POPEYE vision system can be divided into four levels: host level support, device level support, object level support and applications programming. Each level consists of several programs and subroutine libraries. The software is written mostly in C, and was created, edited and compiled on the host computer, a DEC Vax 11/750 running the UNIX operating system. This machine serves as a support facility for several projects of this type.

Host Level Support

Host level support consists of the following features.

- EDITING AND COMPILING. The CMU-standard UNIX editor is used for creation, editing and control of compilation. The C cross compiler package for the 68000 is very similar to the native C compiler for the VAX.

- DEBUGGING. When a program dies unexpectedly, the POPEYE monitor prints a cryptic diagnostic on the user terminal which shows the contents of the program counter, status register and possibly some other information. Given the address where the program died, the debugger will search the symbol table file for that program, figure out which subroutine contains the address and disassemble the subroutine. Like its UNIX counterpart,

the debugger can manipulate several programs with their associated symbol tables and executable segments.

- DOWNLOADING AND UPLOADING. At the end of the compilation process, an extra phase of the C cross-compiler produces an ASCII version of the executable program in Motorola VERSABUG format. At the request of the MPU, the host machine dumps this file over the serial line connecting the two processors. The MPU executes a subroutine which reads the file, decodes the VERSABUG records and loads the executable code into main memory.

  In addition to trading in VERSABUG format, the host machine also implements a generalized upload/download protocol designed to support the transfer of image data. The black and white camera attached to POPEYE can be used with color filters to obtain component color images, which can then be uploaded to the host machine, recombined and displayed on the Grinnell color frame buffer system.

- LANGUAGE DEVELOPMENT. Many of the application programs for POPEYE are simple enough to need only a single character menu driven input paradigm. In certain cases, however, the input is structured enough to warrant a parser and/or a lexical analyzer. The host UNIX system has tools for building just such items, and which output code in C. With only minor modifications relating to i/o, this code can be cross-compiled and executed on POPEYE's MPU.

- HARDCOPY. Often, hardcopy of some entity such as an image, a line scan or a histogram plot is desired. The high resolution laser printer connected to CMU's Ethernet is used for this purpose. The information is uploaded to the host in one of several data formats, converted by some program or sequence of programs into a printable file and finally shipped over the Ethernet to the printer. The printable files can also be included as illustrations in documents. Because of printer limitations, images must be binarized before being printed.

Device Level Support

At the heart of the device level software lies the monitor. This program is stored in EPROM in the MPU and is executed on power-up and on receipt of fatal exceptions such as bus errors. The monitor provides enough capability to download and execute programs through the implementation of the following features.

- TALK-THRU MODE. The monitor can make a software connection between the two serial lines on the MPU board to allow the user access to the host as if there were no vision system between the two. This is the mode of operation during logins, editing and compiling. After editing and recompiling a program, the user can exit talk-thru mode and return to POPEYE.

- DOWNLOADING. When the user wishes to download and execute a program, he gives the name of the program to the monitor. The monitor requests the program from the host and enters download mode. During the downloading process, the monitor takes apart the VERSABUG format file produced by the cross-compiler and sets the executable code into main memory.

- DEBUGGING. For simple hand debugging jobs, the monitor allows the user to examine and change the contents of memory on an 8, 16 or 32 bit word basis, single step through a program or trace a number of assembly language instructions and set breakpoints.

- DYNAMIC MEMORY ALLOCATION. To make the application programs smaller, cleaner and easier to write, a dynamic memory allocation package was installed in the monitor. The package is initialized before the execution of each program and provides whatever space the program may request for temporary storage.

To maintain independence of hardware configuration, the monitor knows nothing about any hardware outside of the MPU.

The remainder of the device level support layer is a collection of device drivers for the various hardware subsystems described in section 2.2.1.

- The serial i/o package communicates with the terminal and host computer.

- The parallel i/o package communicates with the special purpose hardware interface to provide the MPU with control over the pan/tilt head and the motorized zoom lens.

- The disk i/o package handles the lowest level of data transfers to and from the disks and consists of a primitive space manager and the interface to the DMA controller.

- The frame i/o package talks to the image acquisition and display subsystem, controlling the transfer of data to and from the frame buffer and main memory and the grabbing of frames from up to four television cameras.

Object Level Support

The object level support layer consists primarily of the image manipulation package, a subroutine library which provides primitives for the manipulation of images on disk, in main memory and on the screen. The following conventions are used. (Refer to Figure 2-2.)

- A collection of pixels on disk is called an **image**. Images are constrained to be a multiple of 16k bytes in length. This means that a 256 x 256 pixel image typical of POPEYE is of length 4, while a 512 x 512 pixel image typical of the Grinnell system attached to the Vax is of length 16.

- To process an image, the pixels must be moved from disk to main memory, where they reside in a **window**. Windows can be of arbitrary size and shape.

- To aid in processing a window, there exists another object, a rectangular subset of the pixels in a window called a **pane**. Once the pixels is a window have been moved to main memory, the Pane can be moved about within the window, thus eliminating the need to reread the pixels from the disk or from the frame buffer each time the area of interest changes.

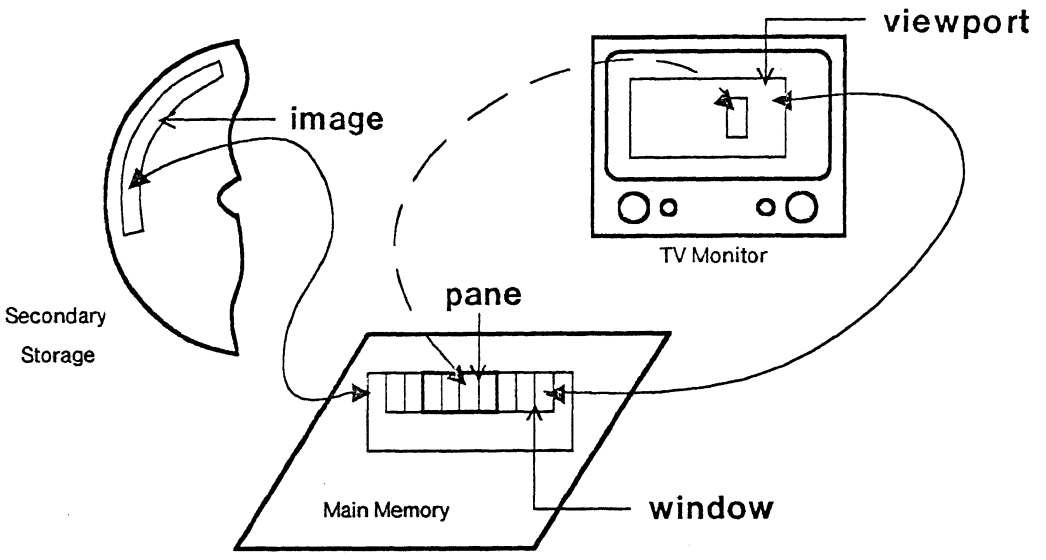


Figure 2-2: Representation of the Image Manipulation Package.

- To view the contents of a window on the monitor, a **viewport** is created and linked to the window. Viewports must have identical dimensions to the windows to which they are linked, but are free to occupy any position on the screen. The size and location of a viewport may be changed interactively by using the cursor movement commands of the terminal. Several viewports may be linked to a single window. Changes made to the contents of a window will be reflected in each viewport to which it is linked.

- The last type of object, the Cursor, is used for pointing to specific locations on the screen.

Application Programs

The remainder of the vision system software is a collection of application programs and subroutines. A large piece in this category is a subroutine library full of garden-variety image processing algorithms such as high pass and low pass filter convolution kernels, the Sobel edge detector, a temporal averaging subroutine to reduce the effects of camera noise, histogram manipulation subroutines, a contrast enhancement package, binarization and cellular logic transform operators and a temporal differencing subroutine. All of these subroutines operate on one or more of the objects described previously.

Above this rather standard library is a collection of more advanced image processing algorithms which we have written for our own purposes.

- The standard binary cellular logic idea has been extended to operate on grey scale images, resulting in Adaptive Cellular Logic, or ACL. This is useful for performing a more intelligent binarization than can be obtained by simple thresholding as well as for edge detection and blob smoothing in grey scale images.

- Several segmentation and data compression schemes have been implemented for the purpose of reducing the amount of processing necessary to perform pattern recognition to a level compatible with real time control [31].

- A small interpretive language for multipass image filtering has been designed and implemented using the compiler development tools on UNIX.

- A large support program has been provided as a base for algorithm development. This program contains most of the subroutines described above, including the software for controlling the pan/tilt head and zoom lens, so that test programs may remain small. The support program is capable of downloading and executing test programs without returning to the monitor, and so does not have to be reinitialized after each program call.

- A general purpose command interpreter package has been written to make the construction of menu driven programs as painless as possible. The package includes facilities for recognizing and executing commands, changing variables during execution and on-line help information.

- A simple tracking algorithm utilizing the image positioning system was implemented to see how close the processing power of the vision system could pull toward real time. The program grabs a frame from the camera and simultaneously binarizes and computes the area and center of energy while reading the pixels from the frame buffer. The area and center of energy are compared to their previous values and the differences used to deliver control signals to the image positioning system. Movement in the x direction generates pan signals, movement in the y direction generates tilt signals and movement in the z direction (change in area) generates zoom signals. While processing the full 256 x 256 pixel frame size, the sampling period is just under one second and all processing is done in the MPU.

- A number of application programs have come from the implementation of industrial inspection algorithms. Typically, a concept demonstration is carried out which evaluates the speed of performance, computational complexity and cost of implementation for a given algorithm. The application packages written for this system have served not only to demonstrate the feasibility of specific inspection algorithms, but have also driven the software development of the system to a significant extent. Industrial inspection applications are described in Chapter 5.

- Automatic focusing algorithms are described in Chapter 3.

2.3. Extensions for Camera Parameter Control

This section describes the hardware and software added to the POPEYE system in support of the work on automatic focusing.

2.3.1. Hardware Configuration

The organization of the camera parameter control system is shown in Figure 2-3. The principal element in this system is the VICON V129-8PP controller which communicates with the motorized zoom lens and pan/tilt head. Although originally intended for surveillance applications, it was modified with relatively little effort to accept positioning commands from the POPEYE system.

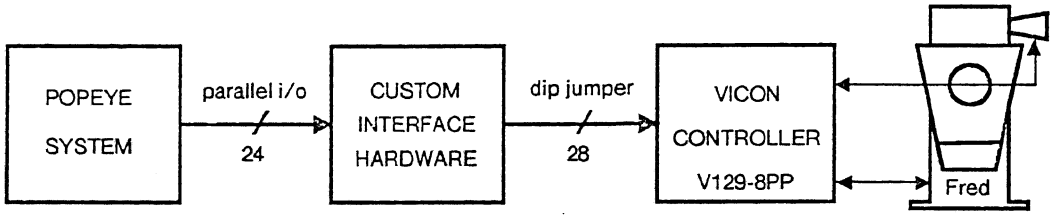


Figure 2-3: Open-loop control path from the POPEYE system to Fred.

The lens used for the project was a VICON V12.5-75 motorized zoom lens. The zoom, focus and iris rings of the lens each have a DC servo motor and feedback potentiometer attached. The control signals to the motors and feedback signals from the potentiometers are carried through a connecting cable to the controller.

The pan/tilt head is a VICON V300PT and is similar in operation to the lens. Both degrees of freedom are driven by DC servo motors and monitored by feedback potentionmeters. Total pan range is approximately $\pm 170^{\circ}$ from the forward-looking position. Tilt range is usually confined by hardware limit stops to within 50° of level.

The controller provides manual control of pan, tilt, focus, zoom and aperture, and can also store eight preset positions of pan, tilt, focus and zoom. Since a single EAROM[2] chip is responsible for remembering the presets, it was decided that the easiest method of interfacing the POPEYE system to the controller would be to emulate the chip. Now, when the controller calls for the contents of the 8^{th} preset, the interface hardware returns the processor's idea of what the pan, tilt, focus and zoom parameters should be instead. The four parameters are independently controllable. The details of the interface design are given in Appendix B.

The camera used was an RCA TC2000 black and white vidicon camera. This is a standard item in many vision systems and is very cheap. The GE TN2500, a CCD array camera, was considered for

[2] Electrically Alterable Read-Only Memory.

the project but when tried experimentally did not work well with the MATROX frame grabber. This was due to sampling misalignment between the camera controller and the frame grabber.

For convenience, the trio of the camera, lens and pan/tilt head is referred to as *Fred.*

There are several problems with the hardware.

- The control signals between the controller and the pan/tilt head are binary. The head is either moving at constant speed or stationary, which makes even proportional control impossible.
- The pan and tilt motions are extremely slow (approximately 6° per second). This makes tracking of moving objects faster than a sprightly snail impossible.
- The control path from the processor to the controller is open-loop. The only idea the processor has of where Fred is comes from an internal data structure in the software updated each time a positioning command is given. This was judged not to be a fatal loss for focusing since the feedback would come thorugh the image path.
- There is a moderate tolerance (2—3 degrees) allowed by the controller in positioning the pan and tilt mechanisms. This results in slight positioning errors which are barely noticeable when the lens is zoomed out but which are appreciable when zoomed in.

2-3.2. Software Support

The software interface to Fred provides the following capabilities.

- Initialization to a known position.
- Independent positioning of each of the parameters. The parameters are all controlled with 12 bit accuracy by the D/A hardware in the VICON controller. In each case, however, only a portion of the full range is actually usable due to the limit stops set on the lens and pan/tilt head. The usable range is known by the software,
- After each positioning command, an internal data structure is updated with the position given. The contents of the structure may be interrogated by the user program.
- Fred can be interactively positioned using the cursor movement keys on the user terminaL
- Data conversion routines are available to convert between the internal 12 bit representation used for positioning commands and more useful physical units. The 12 bit focus parameter is converted to focal length in centimeters and the zoom parameter is converted to field of view in seconds of arc.

To perfonn the conversion between internal and physical parameter values, the emprically 6b-

tained data points shown in Figures 2-4 and 2-5 are used. The points are stored in the software as arrays and accessed by table search. Interpolation is performed between points. The accuracy of the data is of course subject to the positioning tolerance mentioned above.

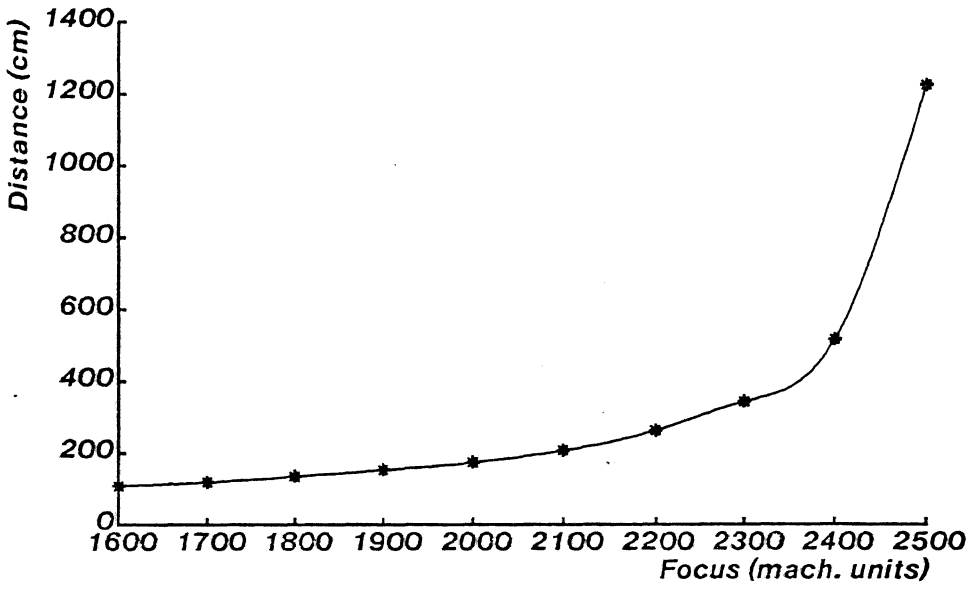


Figure 2-4: Characterization of Fred: focal length vs. the FOCUS parameter.

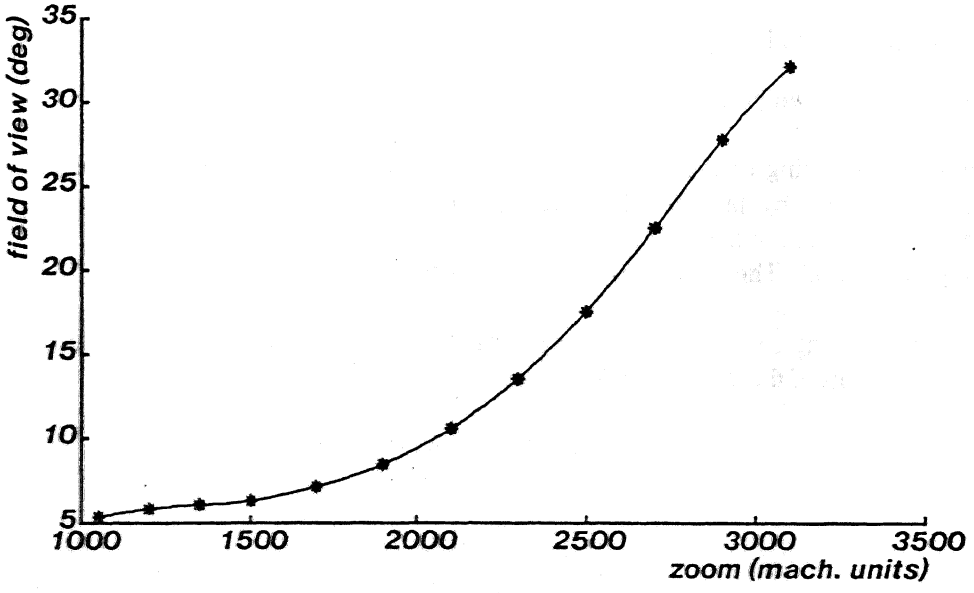


Figure 2-5: Characterization of Fred: field of view vs. the ZOOM parameter.

Chapter 3
Automatic Focusing Algorithms

The next two chapters describe the application of the POPEYE *system to the problem of automatic focusing. The introduction explains the basic concepts of automatic focusing and examines some of the issues in implementing an autofocusing system Next, a brief analysis of the effects of defocusing is given. Preprocessing operators used to reduce noise are described and analyzed. The different algorithms used for focusing are described, and finally, a brief description of some commercial autofocusing strategies is given*

3.1. Introduction

Automatic focusing is something humans do quickly and instinctively. Rarely does an eye need conscious direction from the brain. Because the act of focusing is so instinctive and because our understanding of human "hardware" is so limited, it is very difficult to understand exactly what Is happening. Consequently, it is difficult to completely formulate the focusing problem for a machine. Nevertheless, the effects of defocusing in optical systems are well understood, and so there are many algorithms that can do a reasonable job of focusing.

The general idea behind focusing is the maximization of high frequency content Almost all focusing algorithms - especially the successful ones - depend directly on the amount of high frequency information in the image. Figure 3-1 shows the intensity profiles of a well focused edge and a poorly focused edge and their resulting frequency spectra. The difference in high frequency content is obvious. This difference suggests a possible method of focusing. By computing a two-dimensional Fast Fourier Transform (FFT) over the area of interest and summing the high frequency terms, a direct measure of the high frequency content could be obtained. No other function is as obviously related to the high frequency content as the frequency spectrum.

"* Why not use an FFT-based algorithm for focusing, then? The reason is that the FFT is an overkill for the focusing problem. All that is needed is a single scalar which is monotonically related

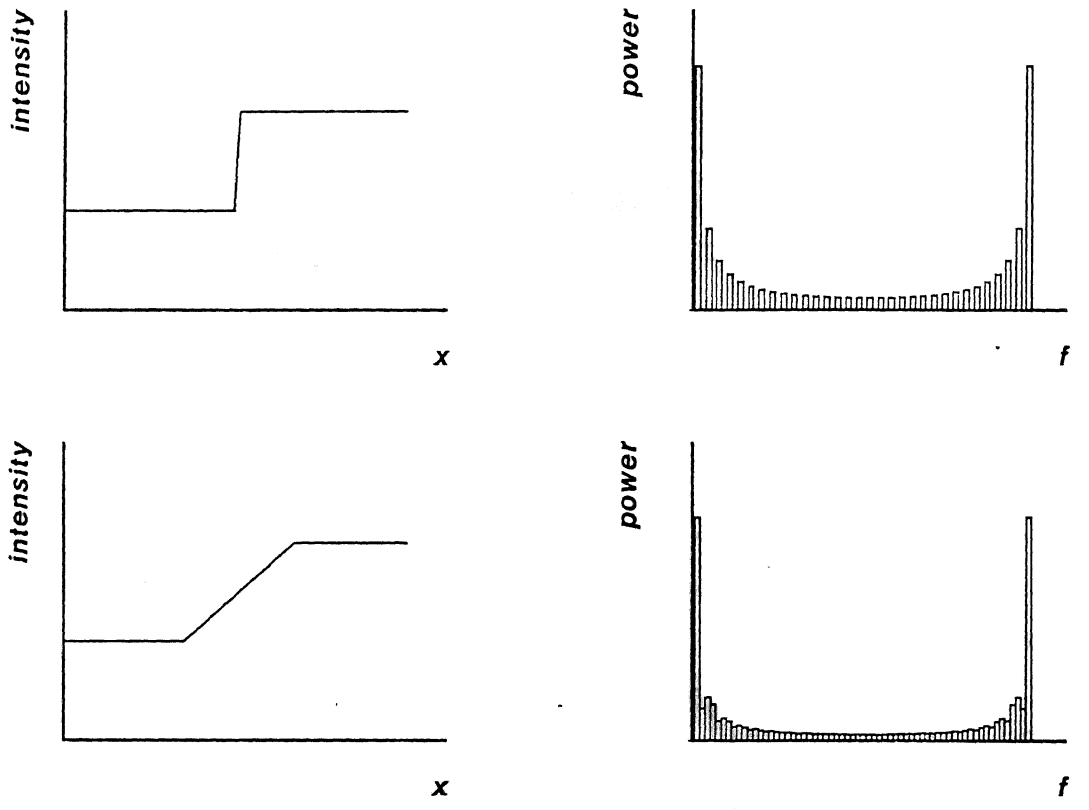


Figure 3-1: The effect of defocusing on high frequency content.

to the high frequency content in the image. Magnitude and phase information for each frequency band is unnecessary. If the extra information came for free, it could simply be ignored without complaint. Unfortunately, however, the FFT is inordinately expensive. Since an N point one-dimensional FFT requires $\frac{N}{2} \log_2 N$ *complex* multiplications, an N by N two-dimensional FFT requires $N^2 \log_2 N$ complex multiplications. (N row transforms and then N column transforms are performed.) Convolution of a k by k kernel with an N by N image requires $k^2 N^2$ *real* multiplications. Since four real multiplications are required for each complex multiplication, the next question to be answered is how does k^2 compare to $4 \log_2 N$?

A typical processing window for automatic focusing is 32 by 32 pixels, and a typical convolution kernel size is 3 by 3. This sets k^2 at 9 and $4 \log_2 N$ at 40. Not only is this an appreciable difference, but the operations required for a convolution are often only integer additions, subtractions and shifts. In contrast, the FFT usually requires floating point operations. Typical applications for automatic focusing are in the area of robotic vision where the processor should be small, inexpensive and quick. Therefore, it is advisable to avoid algorithms which require floating point computations. The focusing algorithms described later in this chapter use only integer arithmetic.

3.2, What Happens When an Image is Defocused

This section gives a brief analysis of the effects of defocusing. The symbols to be used in the subsequent equations are tabulated below and shown in Figure 3-2.

X_Q	object position
x_i	image position
f_L	lens focal length
x_f	plane of best focus position
x_d	detector plane position
d	lens diameter
a	pillbox convolution kernel diameter
F	objective quality of focus

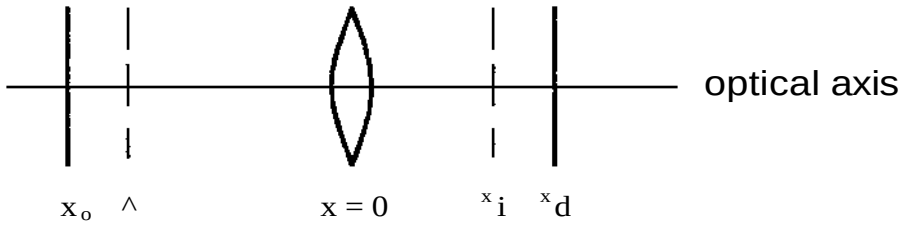


Figure 3-2: An attempt to clarify the focusing notation.

The relationship between the objective quality of focus of an image and the quality of focus as measured by an algorithm can be found in several steps. The first step is to obtain a relationship between the position of an object (relative to the lens) and the place where the image forms. For the single element system of Figure 3-2, this is given by the lens equation:

$$\frac{1}{x_o} + \frac{1}{x_i} = \frac{1}{fh}$$

or

$$x_i = \frac{f_L x_o}{x_o - f_L} \tag{3.1}$$

Next, the object position error Ax_0 is defined as the distance between the object and the plane of

best focus. Similarly, the image position error Ax_z- is defined as the distance between the image plane and the camera sensor. That is,

$$\Delta x_0 = x_0 - x_f$$
$$\Delta x_i = x_i - x_d \qquad \textbf{(3.2)}$$

The positions of the plane of best focus and the detector plane are related in the same way as the positions of the object and image planes. Hence, the image position error can be found by substitution of Equation (3.1) into Equation (3.2). The minus sign in Equation (3J) reflects the fact that the single lens system shown in Figure 3-2 is an inverting system. Henceforth, it will be suppressed.

$$\Delta x_i = x_i - x_d = \frac{f_L x_o}{x_o - f_L} - \frac{f_L x_f}{x_f - f_L} = \frac{-f_L^2 \Delta x_o}{(x_o - f_L)(x_f - f_L)} \qquad \textbf{(3.3)}$$

When an object is not situated in the plane of best focus, its image will not form on the detector. The resultant image is blurred. This blurring is explained and modeled elegantly by considering the resultant image to be the convolution of the perfect image with a blurring function. This is a result from diffraction theory. Since the lens is circular, the blurring function is a circular "pillbox", as depicted in Figure 3-3. The diameter of the pillbox is directly proportional to the lens aperture and inversely proportional to the focal length [33]:

$$a = \frac{d}{f_L} |\Delta x_i| = \frac{d f_L |\Delta x_o|}{(x_o - f_L)(x_f - f_L)} \qquad \textbf{(3.4)}$$

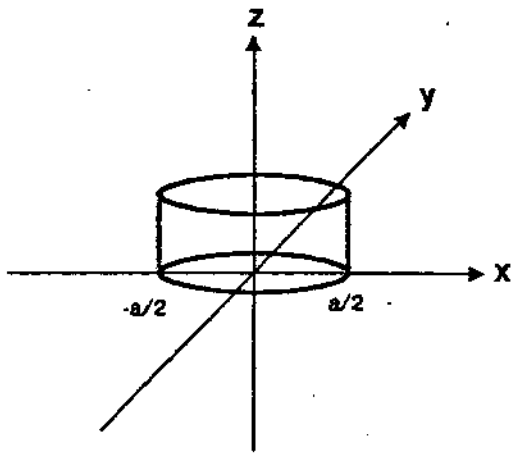


Figure 3-3: Representation of the pillbox convolution kernel

The most obvious effect of defocusing on an image is the change in edge character. The intensity profile of a single sharply focused edge is a step function, as shown in Figure 3-4. As the image is defocused, the profile aquires a transition width the size of the convolution kernel.[3]

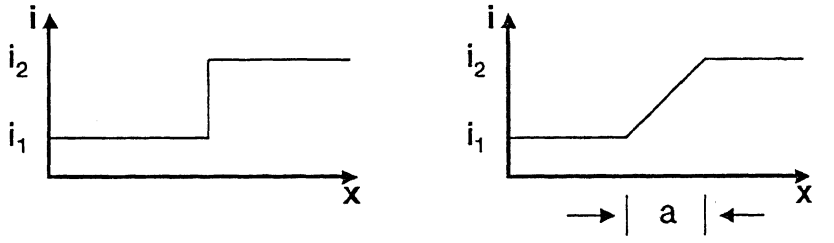


Figure 3-4: The dependence of the intensity profile on focus.

This is the conceptual point of departure for the algorithms to be described in the following sections.

3.3. Preprocessing for Noise Reduction

Most of the discussion on focusing algorithms will assume that it is possible to know the exact relationship between a scene and its image. Unfortunately, the image is always corrupted by noise. Contributions come from several sources: shot noise in the sensor, thermal noise in the amplification components, connecting cables and rf interference from nearby computers. While an analysis of the noise effects is possible in some cases, it is still a good idea to reduce the amount of noise as much as possible. This is a standard preprocessing step designed to help all the algorithms do the best they can.

The real reason for preening images before using them to focus a camera lens becomes apparent when we see how the focusing program works. The algorithms described above all have one characteristic in common: they rely on a criterion function which has a maximum or minimum at the point of best focus, as in Figure 3-5. The strategy, then, is to evaluate the function, move the lens, evaluate the function again, and see what happened. If the function decreases, we must have moved the lens the wrong way. If the function increases, we're on the right track. Eventually, the process of moving the lens to please the focusing function leads us to the top of the hill. This type of algorithm is called a hill-climbing or gradient ascent algorithm.

Now, consider what happens if the function has two humps, as in Figure 3-6. We start at the

[3] This analysis neglects second order effects such as changes in depth of field.

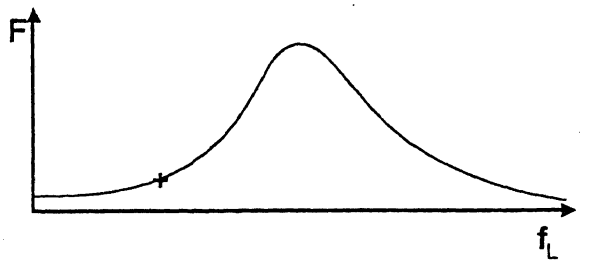


Figure 3-5: The desired form for the quality of focus vs. focal length relationship.

indicated point. We'll probably end up on top of the small hill, which is bad news if we're looking for the maximum value of the focusing function. If there's a lot of noise in the focusing function, as in Figure 3-7, things can become extremely difficult. This is why we try to smooth things out a bit before we start.

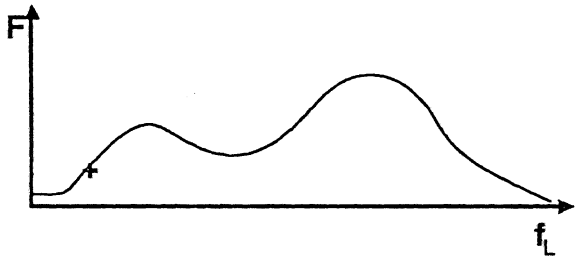


Figure 3-6: A criterion function with two humps: the hill climber's nemesis.

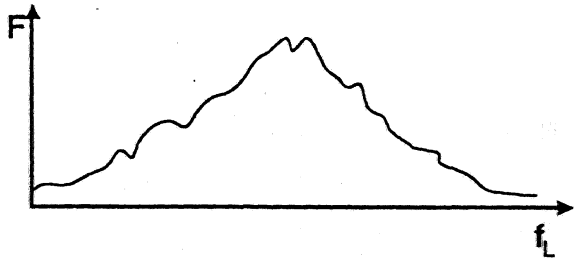


Figure 3-7: An example of a noisy criterion function.

3.3.1. Spatial Averaging

The most common technique used by image processing wizards to reduce noise in an image is spatial averaging. The two algorithms most often used are the simple four and eight pixel replacements of Equations (3.5) and (3.6), where the pixels are labelled as in Figure 3-8.

NW	N	NE
W	X	E
SW	S	SE

Figure 3-8: Pixel map for the standard spatial averaging algorithms.

$$x = \frac{N + E + S + W}{4} \tag{3.5}$$

$$x = \frac{N + NE + E + SE + S + SW + W + NW}{8} \tag{3.6}$$

The action of the spatial averaging algorithms is easily interpreted in the context of Laplace's equation. Consider the intensity of an image as a function of the two spatial variables as a surface in three dimensional space. To reduce noise, what's needed is to minimize the curvature of the surface at every point. The best we can hope for is *zero* curvature, so we set some estimate of the curvature to zero. This is exactly what Laplace's equation (Equation (3.7)) does.

$$\frac{\partial^2 i}{\partial x^2} + \frac{\partial^2 i}{\partial y^2} = 0 \tag{3.7}$$

Equation (3.8) is an acceptable approximation to the second derivative.

$$\frac{\partial^2 i}{\partial x^2}\Big|_{m,n} \approx \frac{i_{m+1,n} - 2i_{m,n} + i_{m-1,n}}{4} \tag{3.8}$$

Combination of Equations (3.7) and (3.8) yields Equation (3.5), the four pixel averaging scheme. The eight pixel scheme comes from taking into account the derivatives in the diagonal directions as well.

The principal drawback inherent in spatial averaging is its tendency to blur the image. Since the processed value of each pixel depends on the values of its neighbors as well as on its own, the energy in the image spreads out after each filtering pass.

Both algorithms act as low-pass filters, and may be analyzed as such. In the four pixel case, the

two-dimensional z-transform of Equation (3.5) yields Equation (3.9), where z_1 and z_2 are the z transform variables of m and n:

$$X(z_1,z_2) = \tfrac{1}{4}(z_1 + z_1^{-1} + z_2 + z_2^{-1})\, I(z_1,z_2)$$
$$= \tfrac{1}{2}\left(\frac{z_1 + z_1^{-1}}{2} + \frac{z_2 + z_2^{-1}}{2}\right) I(z_1,z_2) \qquad (3.9)$$

To find the frequency response, replace z by $e^{j\omega}$ to get Equation (3.10).

$$H(\omega_1,\omega_2) = \tfrac{1}{2}(\cos\omega_1 + \cos\omega_2) \qquad (3.10)$$

By incorporating some "intelligence" into the filtering algorithm, it is possible to remove noise in certain areas of the image while leaving others untouched. For example, homogeneous areas of the image could be filtered without sacrificing edge character, an operation which is clearly needed when performing edge or line detection. This type of smart filter, called an adaptive spatial averaging, or ASA filter [6], is actually two filters: one which decides which areas of the image are to be filtered, and another which performs the filtering.

3.3.2. Temporal Averaging

If the scene is cooperative enough to sit still during the imaging process, we have the opportunity to take more than one snapshot of it, and can then reduce noise by *temporal* averaging rather than spatial averaging. Each pixel is replaced by the averaged intensity i_j of the same pixel location in the previous N images:

$$y = \frac{1}{N}\sum_{j=1}^{N} i_j \qquad (3.11)$$

The longer we extend the averaging process, the larger the reduction in noise becomes.[4] Since the noise is primiarily due to shot noise from the vidicon, it can be safely assumed to be additive and Gaussian with zero mean, so the expected reduction in noise variance σ_n^2 is 1/N (Equation (3.12)). The cross terms disappear because the observations of the noise process are assumed to be independent.

[4]Temporal averaging is actually one of the tricks human eyes pull to reduce noise, with tremendous success. The exposure time for a single pixel in a discrete image is approximately 1/(60 x 65536) seconds or 250 nanoseconds, while the integration time for a human photoreceptor is approximately 35 milliseconds. The noise sources are completely different, but the sluggish response certainly helps.

$$\sigma_y^2 = \mathrm{E}[y^2] = \mathrm{E}\left[\left[\frac{1}{N}\sum_{j=1}^{N} i_j\right]^2\right]$$

$$= \frac{1}{N^2}\,\mathrm{E}\,[i_1^2 + i_2^2 + \cdots + i_N^2]$$

$$= \frac{1}{N^2}\,(N\sigma_n^2) = \frac{1}{N}\sigma_N^2 \qquad (3.12)$$

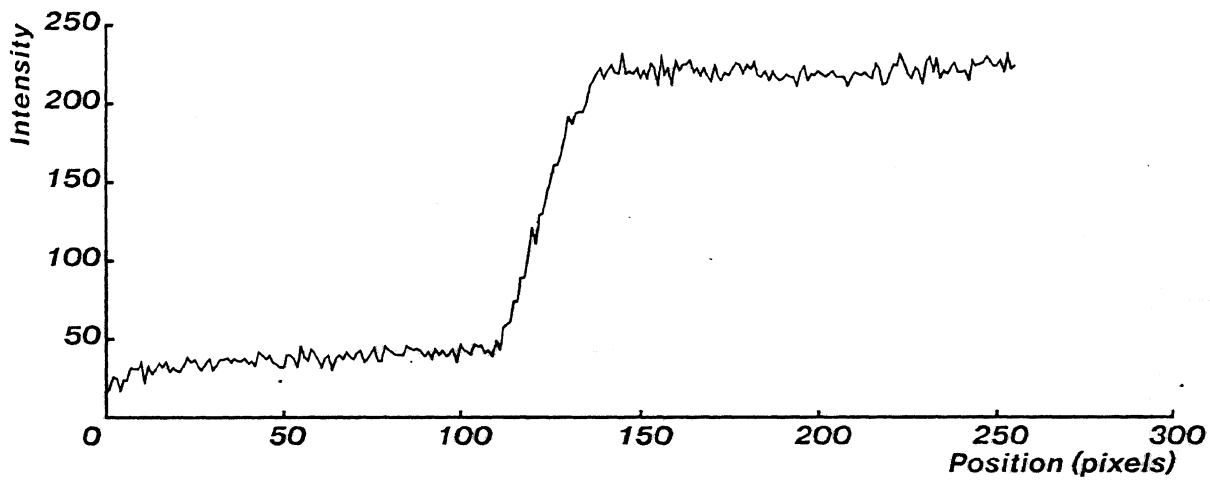


Figure 3-9: An example line scan straight from the camera.

Figure 3-9 is a typical plot of intensity vs. position for N = 1, that is, where no averaging has taken place. The amount of noise in the system is evident. Figures 3-10 through 3-14 show the results of the temporal averaging routine for increasing values of N.

Figure 3-15 is a plot of the actual noise variance obtained versus the number of frames averaged. The dotted line shows the theoretical minimum, while the solid line shows the experimental data. The variances were computed over 32 by 32 pixel windows taken from blank images obtained by leaving the lens cap on.

As long as the image is stationary, temporal averaging is a more desirable method for achieving noise reduction than standard spatial averaging. This is because temporal averaging avoids the blurring effects inherent in spatial averaging. Automatic focusing programs do well to avoid filtering algorithms which blur the image.

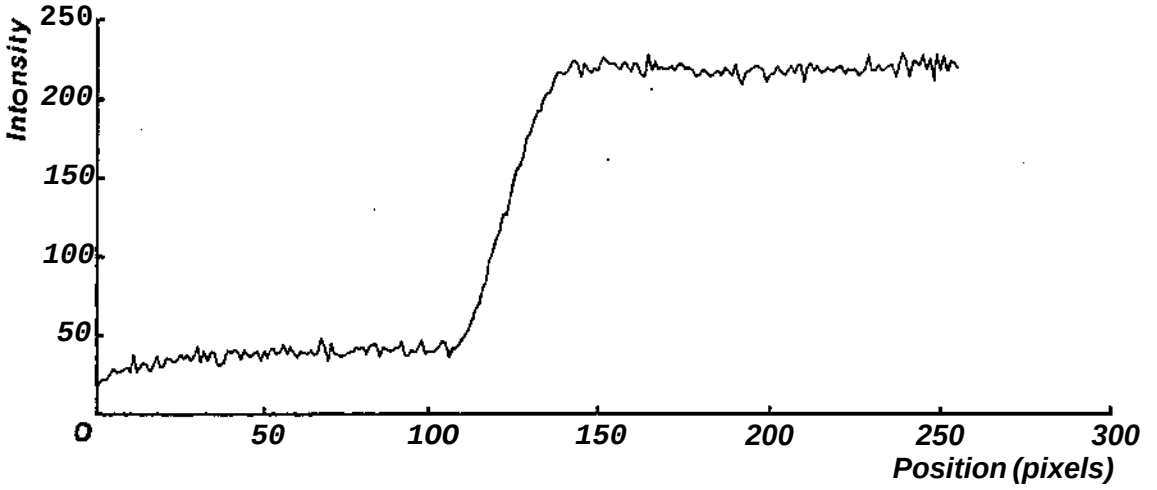


Figure 3-10: A line scan after averaging two frames.

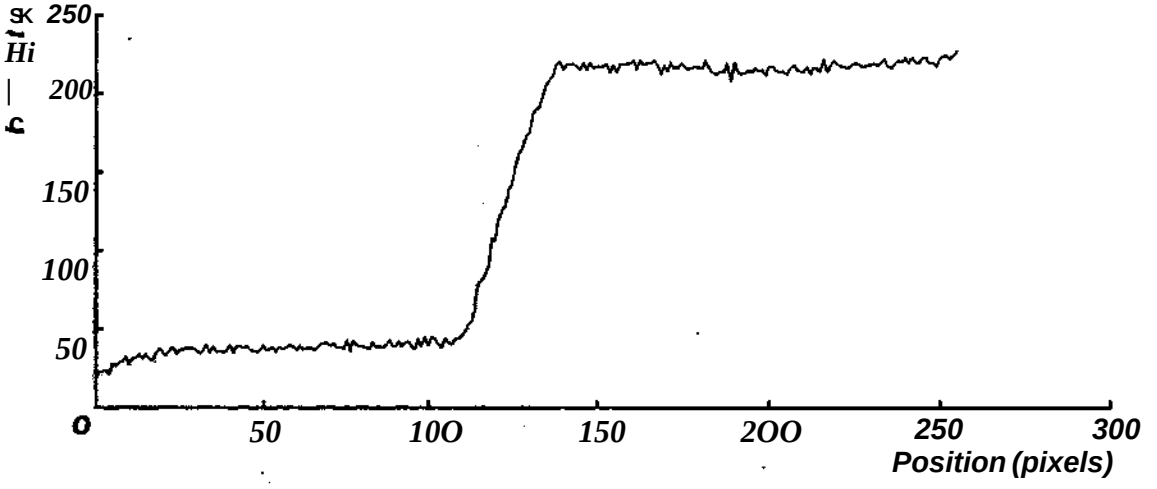


Figure 3-11: A line scan after averaging four frames.

3.4. Edge Transition Width Minimization

Section 3d described and rejected a passible frequency domain method for automatic focusing. In the image domain, the most conceptually simple method -of focusing is. to watch edge profiles. Figures 3-1 and 3-4 have shown what happens to edges when images are defocused: as the quality of focus decreases, the transition between the dark and Eght sections of the profile becomes elongated. Figure 3-16 labels this phenomenon.

Thresholds T_a and 7$ define the dart flat section of the profile. The mean of the intensity signal

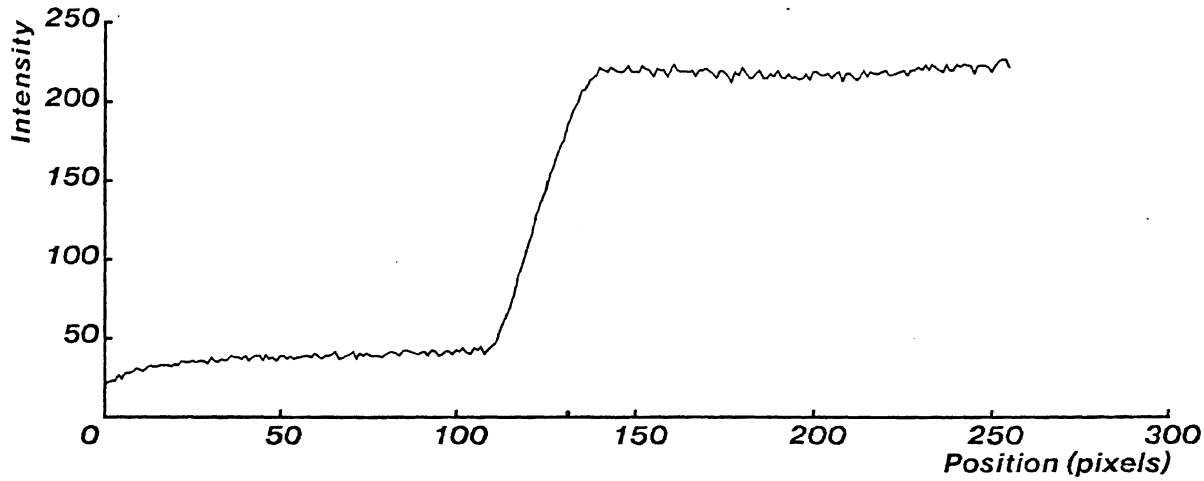


Figure 3-12: A line scan after averaging eight frames.

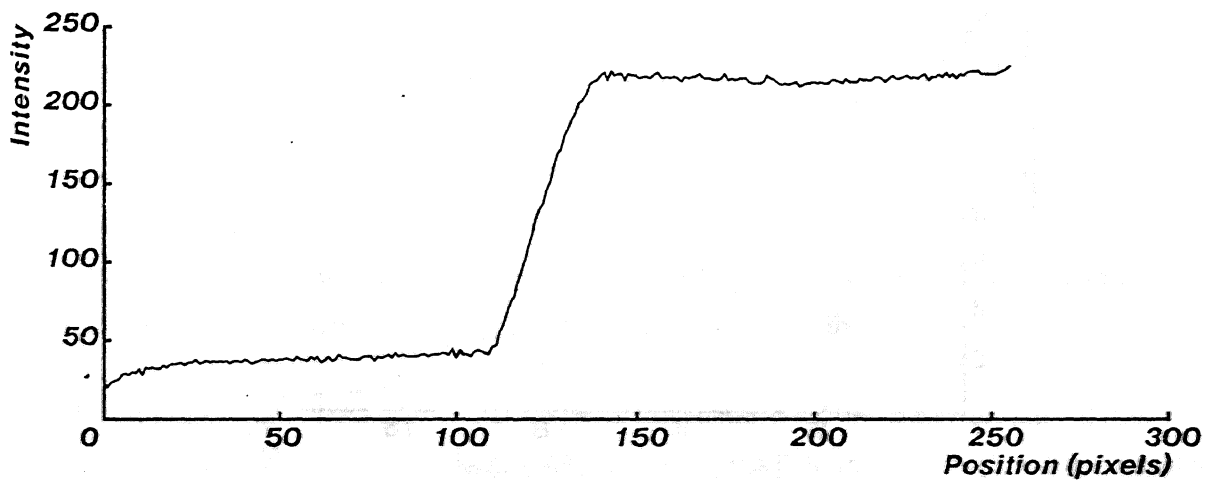


Figure 3-13: A line scan after averaging sixteen frames.

in this region is m_1. T_c and T_d similarly define the bright flat section of the profile, where the mean is m_2. The region between these sections is the transition region of width t_w. The variables d_1 and d_2 are the differences between T_b and T_a and between T_d and T_c respectively. An algorithm which could measure t_w directly would provide a fairly accurate appraisal of the quality of focus. The focusing strategy would then be to simply minimize t_w.

Threshold Monitoring

In its simplest form, edge transition time minimization boils down to monitoring the thresholds in Figure 3-16. This is a possibility which is acknowledged by everyone who writes about automatic focusing but is not taken seriously by anyone [14, 33]. The first problem is finding an edge to work

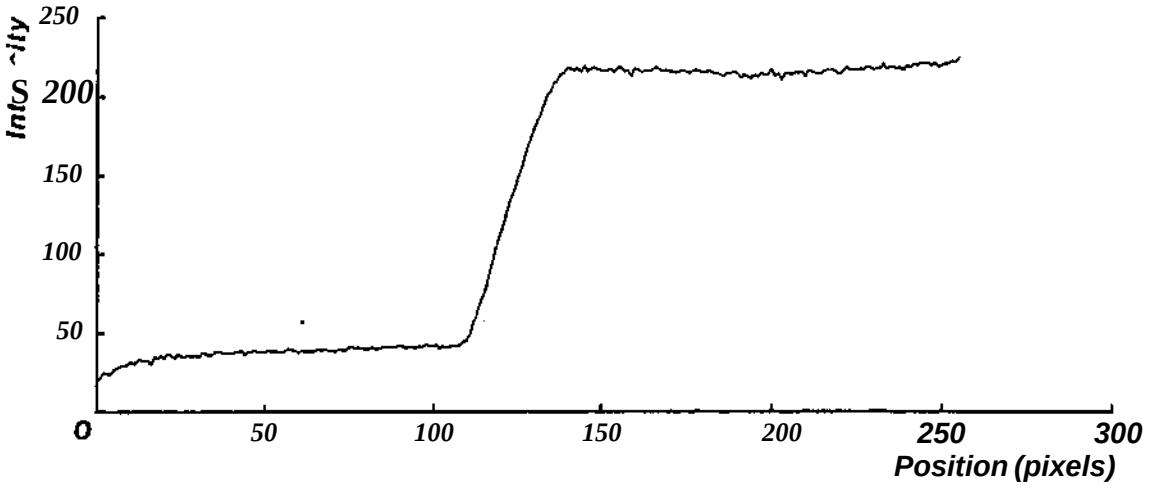


Figure 3-14: A line scan after averaging thirty-two frames.

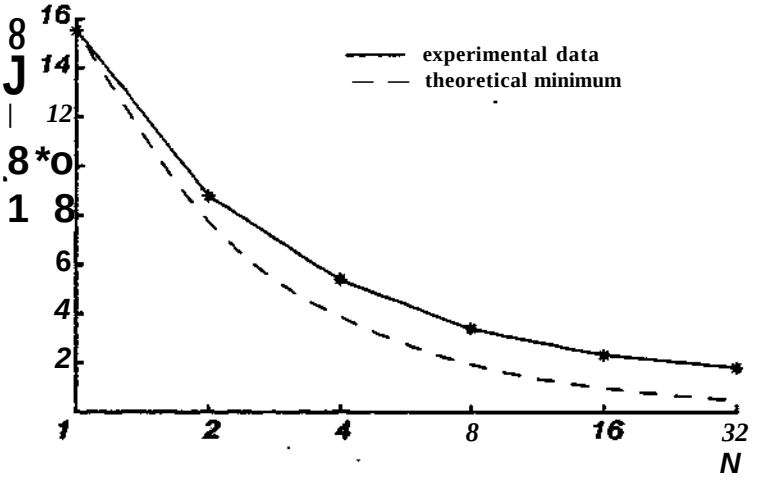


Figure 3-15: Reduction in noise variance due to temporal averaging.

with. This requires the use of some sort of edge detector, which presupposes that the image is in focus. The edge detector would have *to* work reliably even on grossly defocused images and provide both location and orientation information. If we argue that edge detection could be achieved by rcfocusmg 'the camera until the edge detector succeeds, we would be skipping to Section 3.5.

Bypassing the issue of edge detection, however, only brings up the harder problem of picking the thresholds. The transition width i_w *is* defined by T_b and T_c The differences d_x an ^ are available from noise measurements. Unfortunately, there is not enough information to anchor 7^ and T_ϵ since the regions over which the means m_t and m_2 are computed arc dependent on t^ To avoid arbitrariness in selecting the parameteis, a more intelligent approach is needed

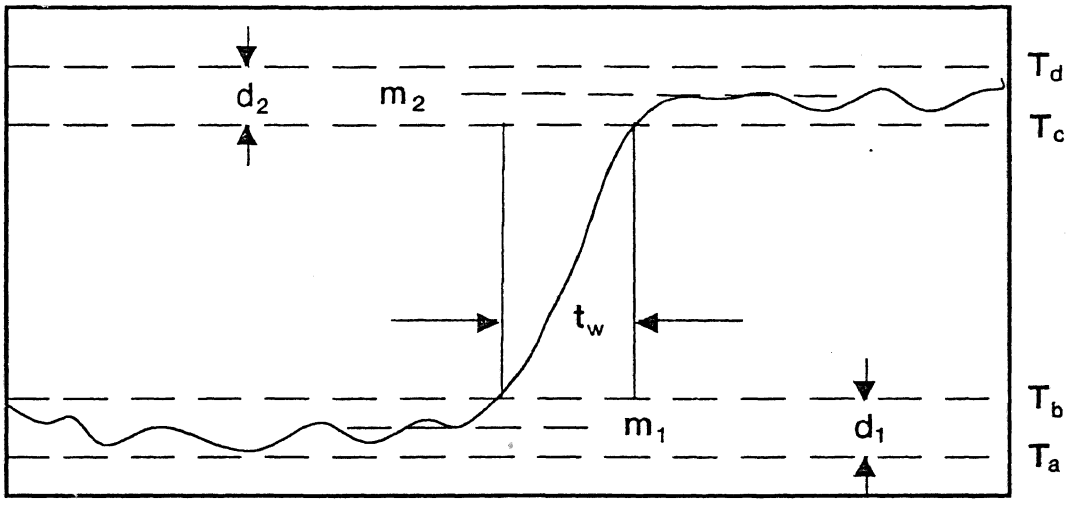


Figure 3-16: Illustration of the Edge Transition Time algorithm.

Adaptive Segmentation

An alternate way of extracting quality of focus information from an edge profile is to find the slope of the transition region. The intensity signal can be divided into three regions, each modeled by a starting point, length, mean, slope and standard deviation. The first and third regions of Figure 3-16 would both have near-zero slope but different means. The transition region would have a large slope and a mean between m_1 and m_2. The standard deviations of all three regions depend only on the amount of noise present and so would be of similar magnitude. Now, the problem can be viewed as one of segmentation where the thresholds are generated dynamically.

This approach, called Adaptive Segmentation, is summarized by Basseville [2]. It has been applied to the analysis of EEG signals [32, 28], speech signals [29] and images [3, 4, 29]. Adaptive segmentation was not applied to the focusing problem for two reasons. First, the problem of finding a good place in the image to start working is still unsolved. Second, the technique is still in the research stage. Implementation of a focusing algorithm based on it could easily have become a research project in itself and so was judged to be beyond the scope of this project.

3.5. Thresholded Gradient Magnitude Maximization

Since the quality of focus affects edge character, it is natural to use an edge detector for automatic focusing. The Thresholded Gradient Magnitude Scheme described below was used by Tenenbaum at Stanford [33]. It is informally referred to in this document as the *Tenengrad*. The implementation of the Tenengrad algorithm for this project followed the steps in Figure 3-17.

$$i_x\Big|_p \approx \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad \dot{b}\Big|_y \approx \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

$$S(p) = \sqrt{i_x^2 + i_y^2} \tag{3.13}$$

1. Set the contents of a register *R* to zero.

2. Evaluate the Sobel operator *S* (Equation (3.13)) at each pixel *p* in the processing window to get *S(p)*.

3. Compare *S(p)* to a threshold 71

4. *If S(p)* ≥ *T* add *S(p)* to *R*.

Figure 3-17: The basic Thresholded Gradient Magnitude algorithm.

After all the pixels have been processed, the register *R* contains a scalar which is used as a quality of focus measure.

The threshold *T is* useful for reducing the sensitivity of the algorithm to noise, but is also theoretically necessary. Here's why. The Sobel operator gives an estimate of the magnitude of the intensity gradient. The sum of the gradients over an edge should therefore produce the edge height, which is independent of the quality of focus. By introducing a threshold, the algorithm is made nonlinear, so only the edges with an appreciable height will contribute to the sum. In return for sacrificing tractability, we get an algorithm that works.

In practice, a slight modification is made to the algorithm in Figure 3-17 to make it run faster. Internally, the Sobel operator computes separate estimates of the gradient components in the x and y directions, squares them, adds them together and takes the square root of the sum. The square root extraction is the most computationally expensive part of the process.[5] Since die threshold *T* is constant across the processing window, the comparison is formed between $S^l(p)$ and T^2 rather than between *S(p)* and 71 The modified algorithm is detailed in Figure 3-18.

The remaining question is how to pick 71 Arbitrary thresholds are a nuisance. In practice,

[5] Texas Instruments now manufactures a chip that computes the Sobel operator in 100 nsec.

1. Set R to zero.

2. Set T' to T^2.

3. Evaluate a modified Sobel operator $S'(p)$ at each pixel p to get $S^2(p)$.

4. Compare $S'(p)$ to T'.

5. If $S'(p) \geq T'$ take the square root and add $S(p)$ to R.

Figure 3-18: The modified Thresholded Gradient Magnitude algorithm.

setting T to zero does not produce the disaster theory predicts, but a production system would be foolish to do so arbitrarily. Tenenbaum [33] treated T as an *adaptive parameter*, turning it up when possible and down when necessary. His heuristic was to set the threshold to 75% of the maximum single value obtained at the last focal length. This meant that as the image came into focus and the gradients got bigger, the threshold would track and eliminate the background texture which was also coming into focus.

If it is desired that the threshold be set only once at the beginning of a focusing run, then the value can be based on an evaluation of the noise in the gradient domain. Assume that the noise in the raw image is normal with mean zero and variance σ_n^2. Assume also that the noise variance is a known system parameter, and that the use of temporal averaging reduces it as shown in Figure 3-15. The Sobel operator first computes the partial derivative estimates i_x and i_y using the neighborhood masks. Each mask simply forms a linear combination of the neighborhood pixels, so the noise distribution in each of the partial derivative estimates is also normal. Since the variances of independent variables add as the squares of their coefficients, the noise variance in each estimate will be $\sigma_x^2 = \sigma_y^2 = 12\,\sigma_n^2$. The final value of the Sobel operator is $S(p) = \sqrt{i_x^2 + i_y^2}$. The noise distribution in the edge image, therefore, is a Rayleigh distribution with parameter σ_x [20] (page 195). The form of the Rayleigh distribution is shown in Figure 3-19. The equation of the curve is given by Equation (3.14).

$$p(z) = \frac{z}{\sigma^2} \exp\left[\frac{-z^2}{2\sigma^2}\right] \qquad \text{for } z \geq 0 \tag{3.14}$$

A possible heuristic for the location of the threshold would be $T = 3\,\sigma_x$. For example, if a four frame temporal average is used, σ_n^2 will be in the neighborhood of 6 for the POPEYE system (see

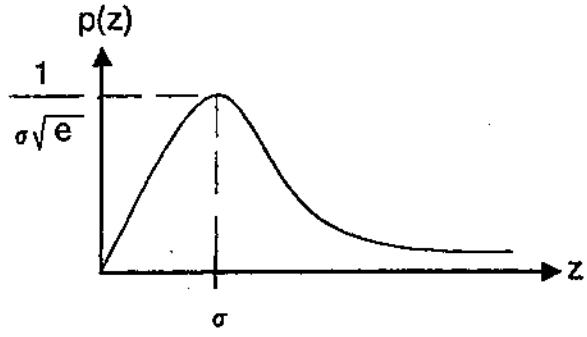


Figure 3-19: The vital statistics of the Rayleigh distribu tion.

Figure 3-15), which puts $3a_x$ at $3\sqrt{12\text{-}6}$ or 25. Alternatively, the threshold could be placed so as to avoid a given percentage of the area. Integrating the distribution and solving for the threshold yields $7= \sqrt{2\text{crj}_C{}^2\ln[1/(1\text{-}£)]}$ where k is a fraction between 0 and 1 representing the percentage of area.

That the noise distribution in the edge image is actually a Rayleigh distribution (or a close approximation) is shown by Figure 3-20. This is a histogram of edge image intensity over a 30 by 30 pixel window. No temporal averaging was used. The histogram has been smoothed with a Gaussian filter of variance 2. The nonzero value at the origin is due to both integer truncation and the smooching performed by the Gaussian filter.

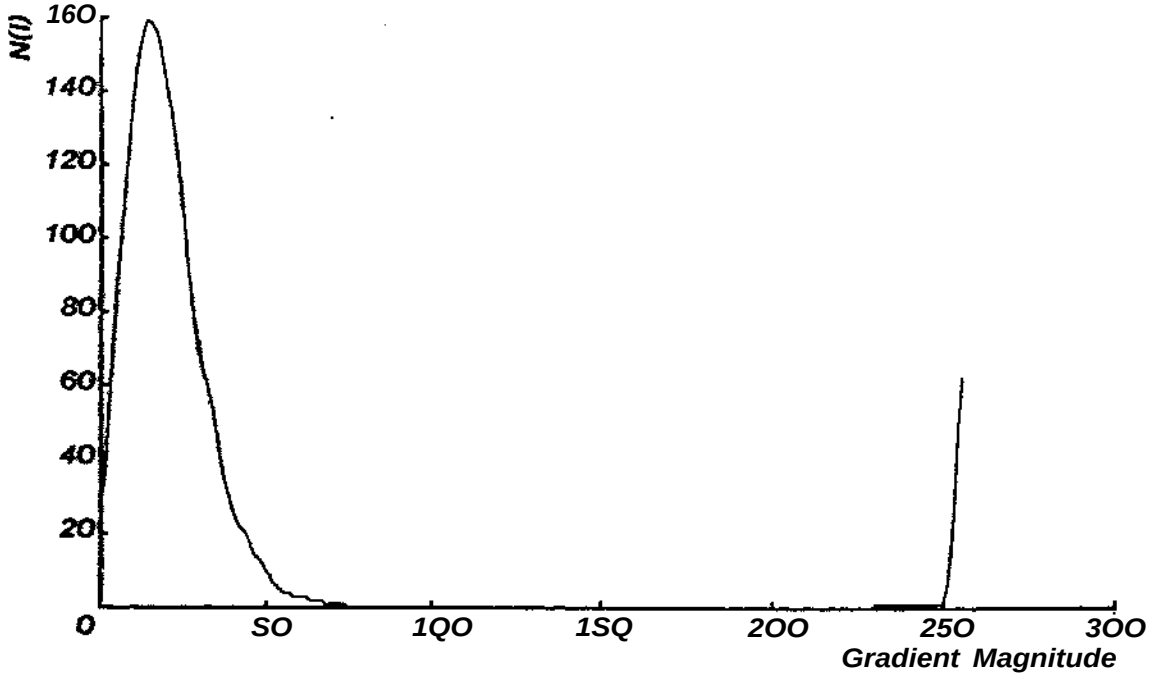


Figure 3-20: Histogram of an edge image from the Sobel operator.

Assuming that the noise variance in a single unaveraged frame is 16 (Figure 3-15 again), theory predicts that the center of the Rayleigh distribution should be $\sigma_x = \sqrt{12 \cdot 16}$ or 14. The peak in the actual histogram data is between 14 and 15.

3.6. Histogram Entropy Minimization

Minimizing the entropy of a histogram is a technique used in information theory to maximize the quality of information. A deterministic signal s will have a probability distribution $p(s)$ which consists of one or more delta functions. If s is corrupted by noise, the probability distribution will have an extended shape. The entropy of the probability distribution $p(s)$ is a measure of how random the signal is.

The histogram of an image consisting of only a single, perfectly focused edge would be two delta functions, as in Figure 3-21. In reality, the histogram is a pair of Gaussian modes due to the presence of noise.

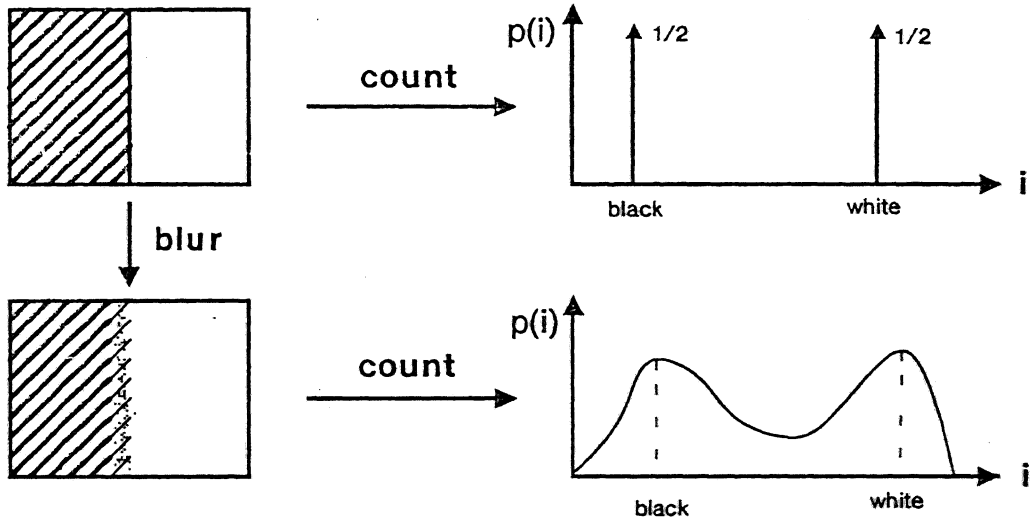


Figure 3-21: Illustration of the Histogram Entropy algorithm.

Below are the histograms of four images of a single edge taken at progressive degrees of defocusing.[6] In each, $N(i)$ is the number of pixels of a given intensity. Figure 3-22 shows the shape of a typical histogram when the image is in focus. There are two sharp peaks which correspond to the dark and light areas of the screen. As the image is defocused (Figures 3-23 and 3-24), a transition

[6] Temporal averaging was *not* used in generating these histograms. The use of temporal averaging significantly decreases the widths of the modes in the histogram.

region begins to appear in the image which gives rise to the points between the peaks (refer back to Figure 3-4 on page 19). In Figure 3-25, as the transition region width approaches the width of the processing window, the peaks become only marginally distinguishable from the flat section between them. From the information point of view, the histogram is beginning to look more like a uniform distribution, so the entropy increases. By trying to minimize the entropy, we are simply trying to push all the pixels away from the middle and back into the modes.

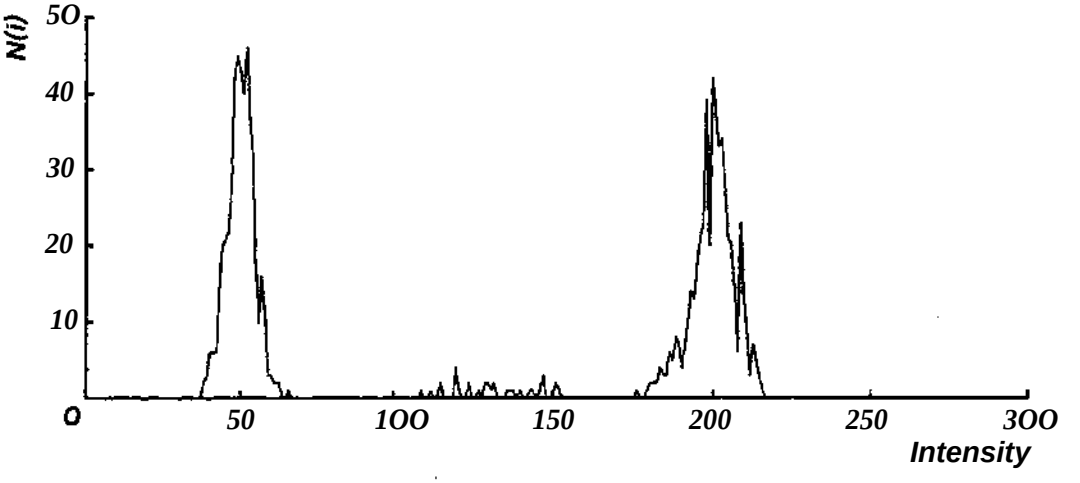


Figure 3-22: The histogram of a sharply focused edge.

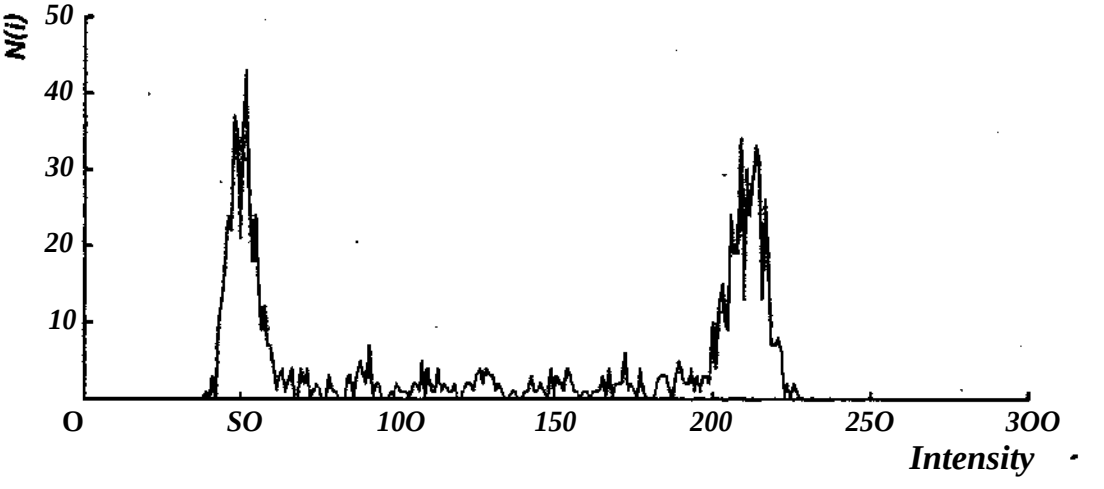


Figure 3-23: The histogram of a slightly defocused edge.

For the canonical single edge, the histogram is expressed by Equation (3.15), where w is the width of the processing window, i_Q is the average intensity, and c= i_2—$\hbar$ is the contrast. The function /w/ftjr—*y)/z]* is a rectangular window with center y and width z.

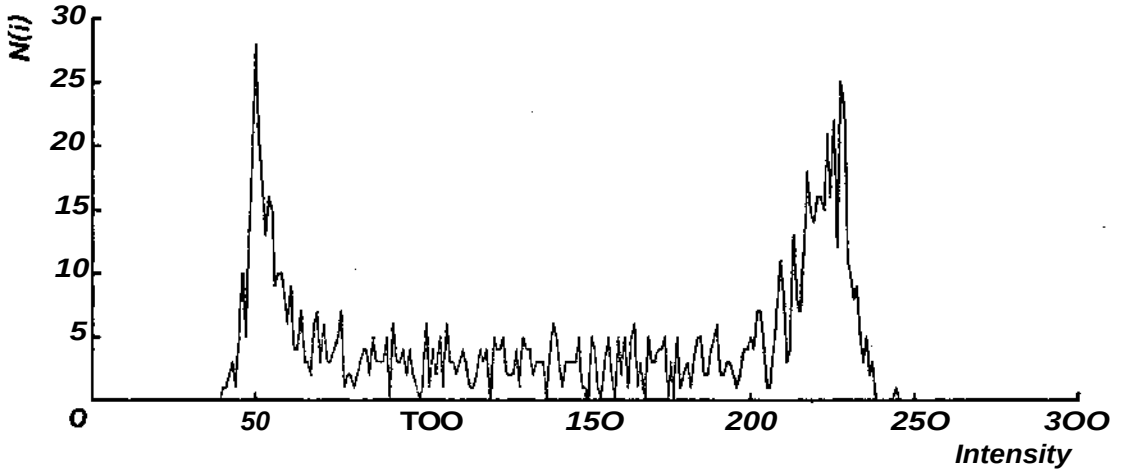


Figure 3-24: The histogram of an appreciably defocused edge.

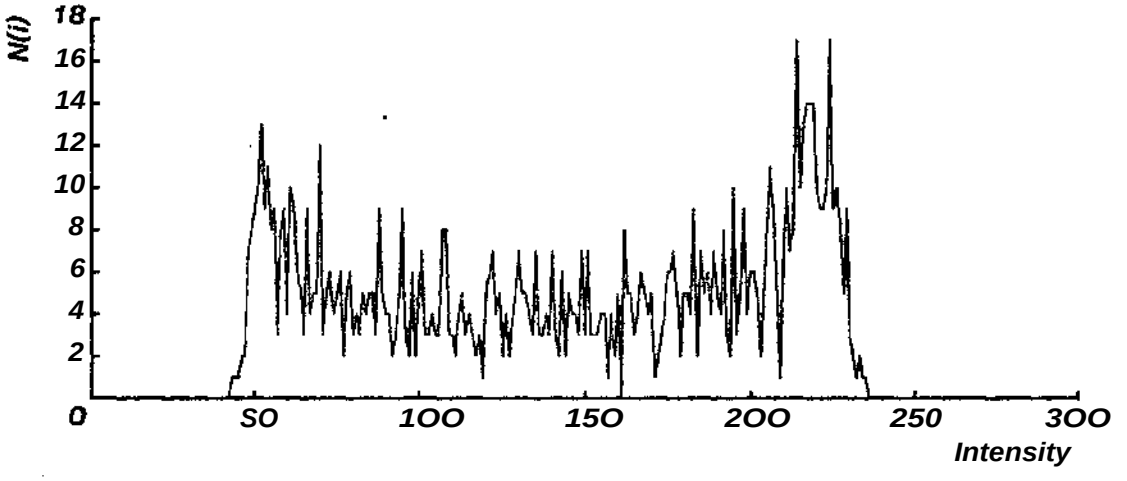


Figure 3-25: The histogram of a completely defocused edge,

$$\hat{} = pgr\text{-}\}[S(4) + mi + (-\hat{}\text{-})rect(i=\hat{}\_) \quad (3.15)$$

For a discrete histogram, the definition of entropy is given by Equation (3.16), where L is the number of intensity levels in the image:

$$E = -\sum_{i=0}^{L-1} P(i)\log_2[P(i)] \quad \textbf{(3.16)}$$

The entropy of Equation (3.15) is then given by Equation (3.17).

$$E = \left(\frac{w-a}{w}\right)\log\left(\frac{2w}{w-a}\right) + (c-1)\left(\frac{a}{w}\right)\log\left(\frac{w}{a}\right) \quad \textbf{(3.17)}$$

As usual, the implementation has added twists. The discrete histogram $P(i)$ is $N(i)/N$, where $N(i)$ is the number of pixels of intensity i and N is the total number of pixels. By using $N(i)$ rather than $P(i)$ to compute the entropy, we avoid floating point computations and obtain E', a scaled and translated version of the real thing.

$$\begin{aligned} E' &= \sum_{i=0}^{255} N(i)\log_2\Big[N(i)\Big] = \sum_{i=0}^{255} N\,P(i)\log_2\Big[N\,P(i)\Big] \\ &= N\left\{ \sum_{i=0}^{255} P(i)\log_2 N + \sum_{i=0}^{255} P(i)\log_2\Big[P(i)\Big] \right\} \\ &= N(\log_2 N - E) \end{aligned} \tag{3.18}$$

3.7. High Pass Filtering

Another way to look for high frequency content is to use a Laplacian mask. A Laplacian mask (Equation (3.19)) estimates the second derivative of the image intensity just as the Sobel operator estimates the first derivative. Two criteria of focus based on Laplacian masks are given below.

$$\textit{4 point:}\quad L(p) \approx \begin{bmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{bmatrix} \qquad \textit{8 point:}\quad L(p) \approx \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \tag{3.19}$$

Convolution Kernels

The eight point Laplacian mask was used in an algorithm similar to the Tenengrad. The magnitude of the operator is evaluated at each pixel, compared to a threshold and added to a register if the test is positive. In this case, the threshold is theoretically unnecessary but very desirable in practice since Laplacian masks are extremely sensitive to noise. Equation (3.20) shows just how sensitive. The nine neighborhood pixels in the high pass kernel are considered to be independent observations of a zero mean Gaussian noise process with variance σ_n^2.

$$\begin{aligned} \sigma_L^2 &= E[(y-\bar{y})^2] = E[y^2] = E[(8i_1 - i_2 - \cdots - i_9)^2] \\ &= E[64i_1^2] + E[i_2^2] - \cdots - E[i_9^2] = 72\,E[i_1^2] \\ &= 72\sigma_n^2 \end{aligned} \tag{3.20}$$

Again, we can set the threshold heuristically by picking a point three standard deviations above the mean or parametrically by integrating. If a four frame temporal average is used, $3\sigma_L = 3\sqrt{72 \cdot 6}$ $= 62$.

The Simple Cross

Some of the algorithms described above are extremely expensive. The Tenengrad, for example, takes approximately one second to process a 32 by 32 pixel window. If 30 samples over the range of focal length are desired, the algorithm will take half a minute to run. Often, we would like to pep things up a bit. What is needed is a focusing function which can quickly land in the general vicinity of perfect focus without taking all day. In some applications, razor sharp focus may be unnecessary or even undesirable. In these cases such a "skydiving" algorithm is appropriate and sufficient. In applications where accuracy is necessary, a more expensive operator could take over after the first one finishes, and would only need to search a small range of focal lengths.

To get the focusing process to run faster, we can try two things: we can use a less powerful operator at each point, or we can reduce the number of points processed. The Simple Cross algorithm does both. The Laplacian mask $[-1 \quad 2 \quad -1]$ is evaluated at each point along the horizontal midline of the processing window and the mask $[-1 \quad 2 \quad -1]^T$ is evaluated at each point along the vertical midline (see Figure 3-26). As before, the magnitudes are thresholded and summed.

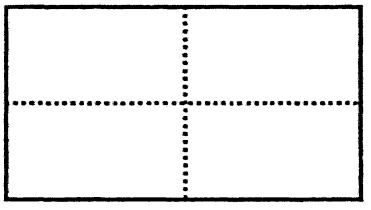


Figure 3-26: The processing lines for the Simple Cross algorithm.

Again, the threshold is theoretically unnecessary but practically beneficial. The mask gives a noise variance six times that of the image noise variance, so for $\sigma_n^2 = 6$, $T = 3\sqrt{6 \cdot 6} = 18$.

3.8. Commercial Methods

For several years, Polaroid and other companies have been marketing cameras which use sonar for automatic ranging and focusing. Late in 1981, Pentax introduced the first commercial camera which performed automatic focusing using a through-the-lens image processing strategy [12]. The Pentax solution to the focusing problem is elegant and innovative. Their camera, the ME-F, does something which human eyes and television camera tubes cannot do: it simultaneously monitors the quality of focus at two different focal lengths. By dividing the incoming light with a beamsplitter and directing the beams to photoreceptors along paths of different lengths, the camera obtains not only quality of focus information but directional information which tells it which way to move the lens.

The path lengths are planned so they straddle the path length to the film plane by equal amounts. When the image is in perfect focus on the film plane, the images on the silicon photodetectors will be out of focus by equal amounts. In all other cases, the qualities of focus measured by the detectors are unequal.

The Leitz and Honeywell approaches are less elegant but still innovative. These methods are based on the relationship between an object and its resultant light distribution which leaves the exit pupil of the lens. When the object is at the point of best focus, the intensity profile across the pupil will be constant, regardless of the angle of view. If the object is moved from the point of best focus, the distribution will change and begin to suggest the outline of the object. Both the Leitz and Honeywell systems work by monitoring the intensity distribution leaving the exit pupil and attempting to make it constant. Directional information is obtained by noting which way the distribution changes.

The strengths of these systems are the potential speed of focusing, the direct availability of direction information and the small size of the apparatus. If automatic focusing ever comes to television cameras, it may well be accomplished by using some variation on one of these methods. The weaknesses are the lack of use of color information, the occasional sensitivity to orientation due to the polarizing effect of beamsplitters and the inability to recover from gross defocus.

Chapter 4
Algorithm Evaluation Procedure and Results

This chapter describes the procedures used to evaluate the focusing criteria from Chapter 3 and shows the results in graphic form. The criteria are tested first by sampling over the complete range of focus of the lens. Focusing is then considered as a closed-loop image processing problem.

4.1. Procedure for Evaluating the Focusing Criteria

Each of the automatic focusing algorithms described in Chapter 3 produces a single scalar for each focal length at which it is evaluated. A simple test of the quality of these algorithms is to evaluate the focusing function over the range of focus of the lens. This was done for four algorithms using the lens control capabilities of the POPEYE vision system.

The data for each plot are produced by a single run of a program which is invoked with the following parameters. The default values used to produce the graphs in Section 4.2 are given in parentheses.

- The number of divisions into which the total working focal length range of the lens is to be split (25). The focusing function is evaluated once at each point.
- The number of frames to be averaged at each point (4).
- Which algorithm to use.
- What threshold to apply, if appropriate.
- Whether or not to run the program synchronously (no). The execution time of several of the algorithms is data dependent. In synchronous mode, the program will assure that evaluations of the focusing function occur at equal time intervals. This mode is used for plotting the time history of the quality of focus.
- What sampling period to use, if appropriate.

The focusing program simply moves the lens, evaluates the function, records the value and then repeats the sequence until the range of focal length is exhausted.

Several algorithms call for multiple runs at different thresholds to study the action of the threshold. In such cases, several thresholds bracketing the theoretical predictions from Chapter 3 were applied.

At the end of each run, the program would pick the maximum value obtained and move the lens to the corresponding focal length. This provided for informal subjective human evaluation of performance. In most cases other than those which used histogram entropy, the result was as good as what could have been obtained by eye.

4.2. Results

The plots in Figures 4-1 through 4-10 show the relationship between subjective quality of focus and focal length for the four algorithms tested. In all plots, the abscissas are focal lengths as measured in the machine units used to control the lens. The range of focal lengths in physical units which corresponds to the range on the plots is approximately 1 meter to infinity. (See Figure 2-4 on page 14.) The ordinates are the actual numbers obtained from the focusing functions. Multiple runs of the same algorithm at different thresholds are plotted together on the same graph.

Figure 4-1 shows two things clearly. First, with the threshold set to zero, the function is nearly flat for a simple image. This agrees with the theory. Second, the peak collapses as the threshold is raised. In Figures 4-2 and 4-3 the curves are *not* flat with the threshold at zero. Although this is actually an aid to focusing, the sources of error should be noted:

* quantization error
* noise
* change in image size due to focus ring movement
* breakdown of the simplistic assumptions about what happens when an image is defocused

It is also useful to note here that the data from the Tenegrad evaluations are basically monotonic. The success of dynamic focusing will depend on this.

The data from the histogram entropy algorithm (Figure 4-4) show that the flinction does indeed peak moootonically for a simple edge image. Unfortunately, the function rises at the edges for a mildly complicated image such as text and inverts itself completely for a very complicated image containing texture. In retrospect, we can see that this is due to the previously stated sources of error, as well as the fact that intensity distributions from adjacent edges coalesce.

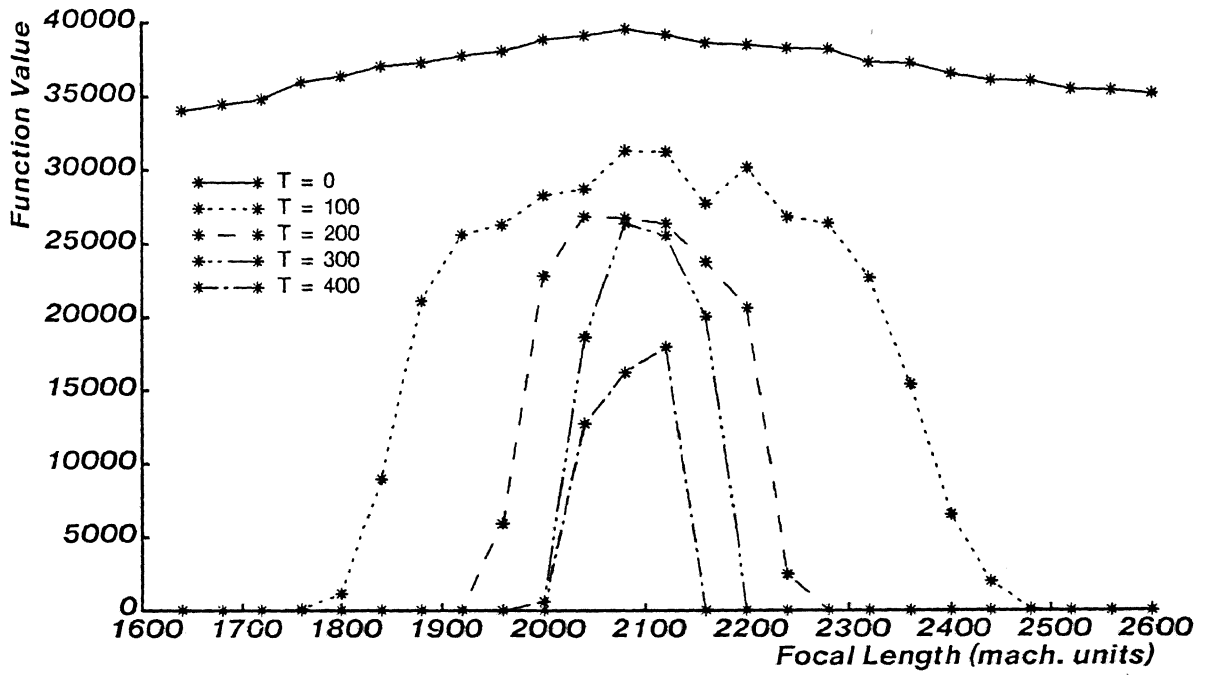


Figure 4-1: Static behavior: the Tenengrad on an edge.

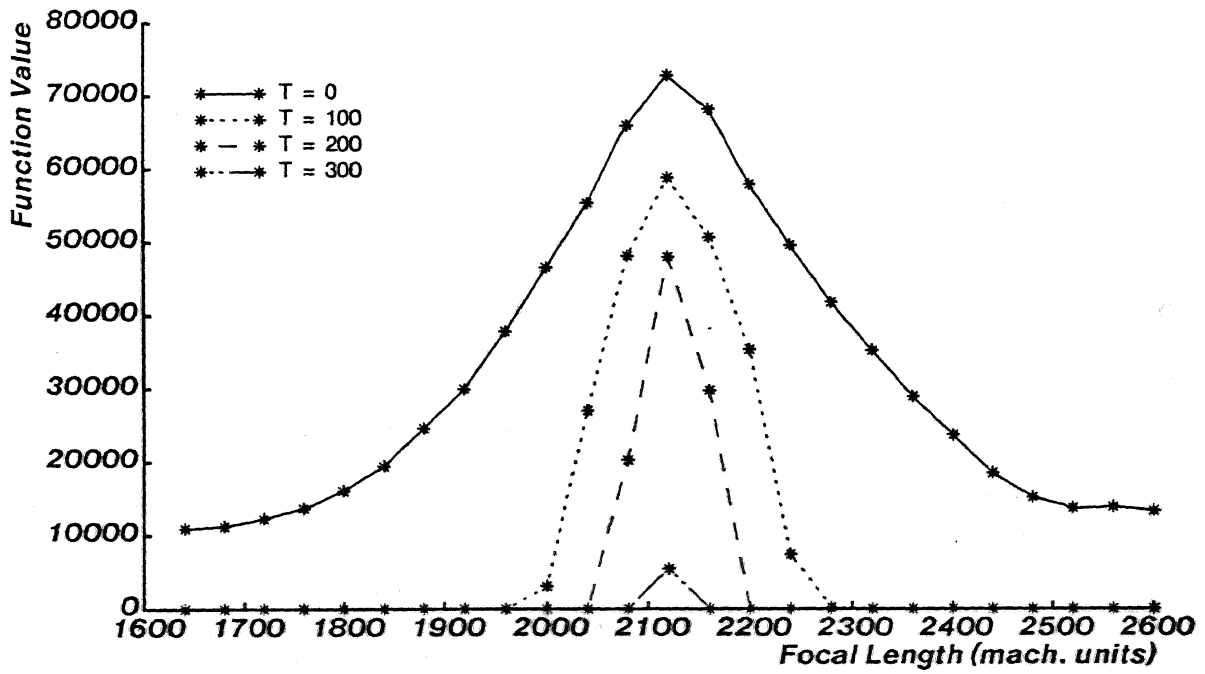


Figure 4-2: Static behavior: the Tenengrad on text.

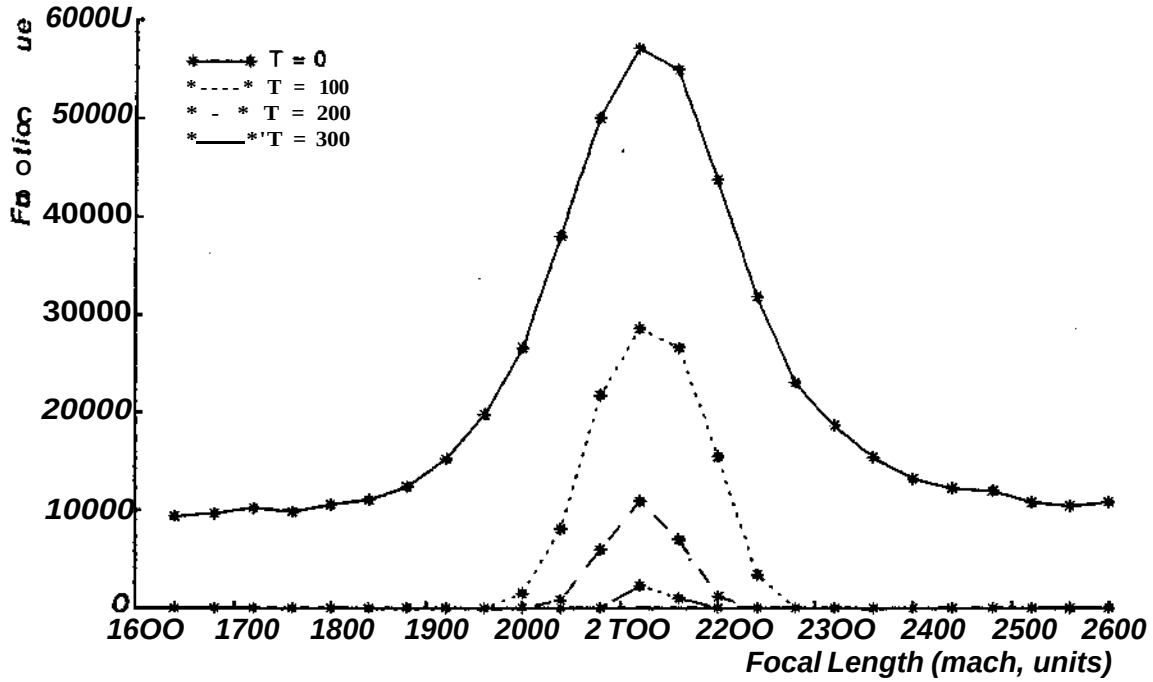


Figure 4-3: Static behavior: the Tenengrad on texture.

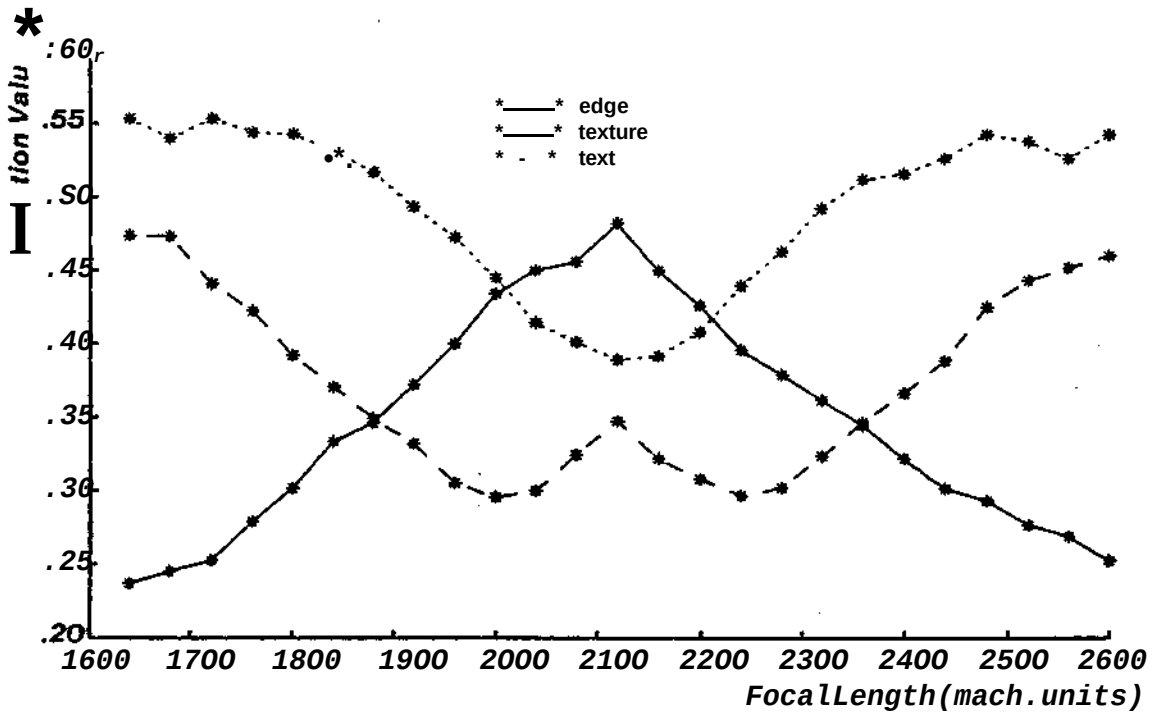


Figure 4-4: Static behavior: the Msiogram entropy function.

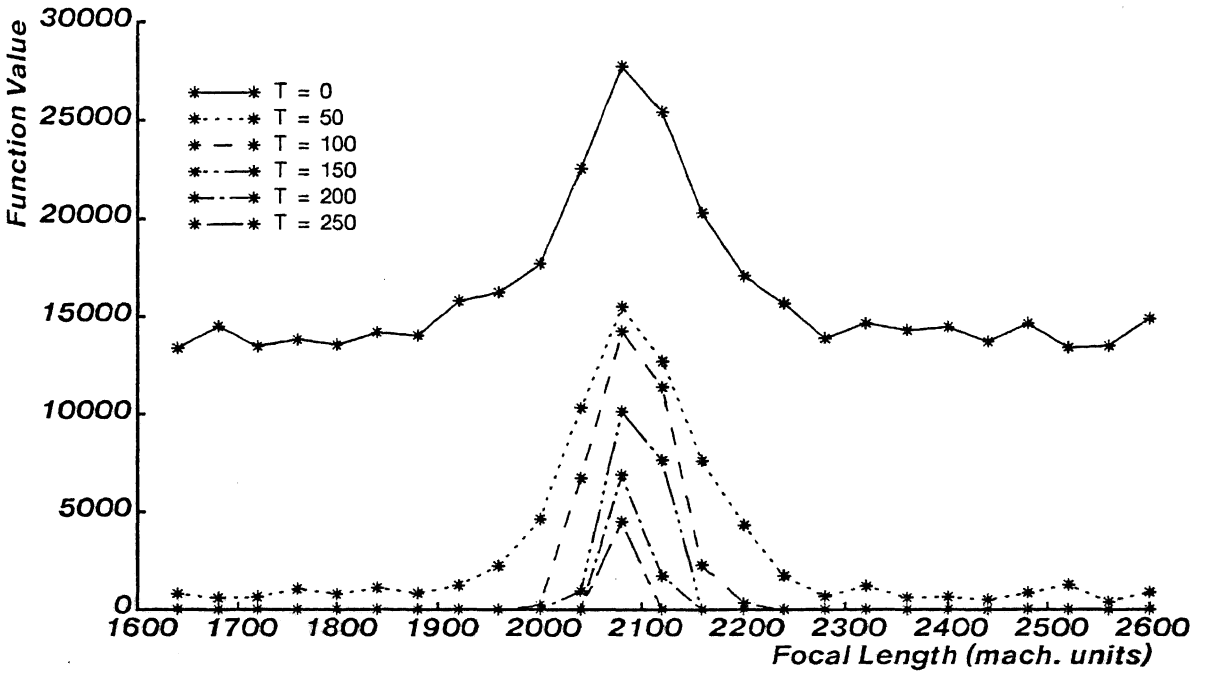


Figure 4-5: Static behavior: the high pass filter on an edge.

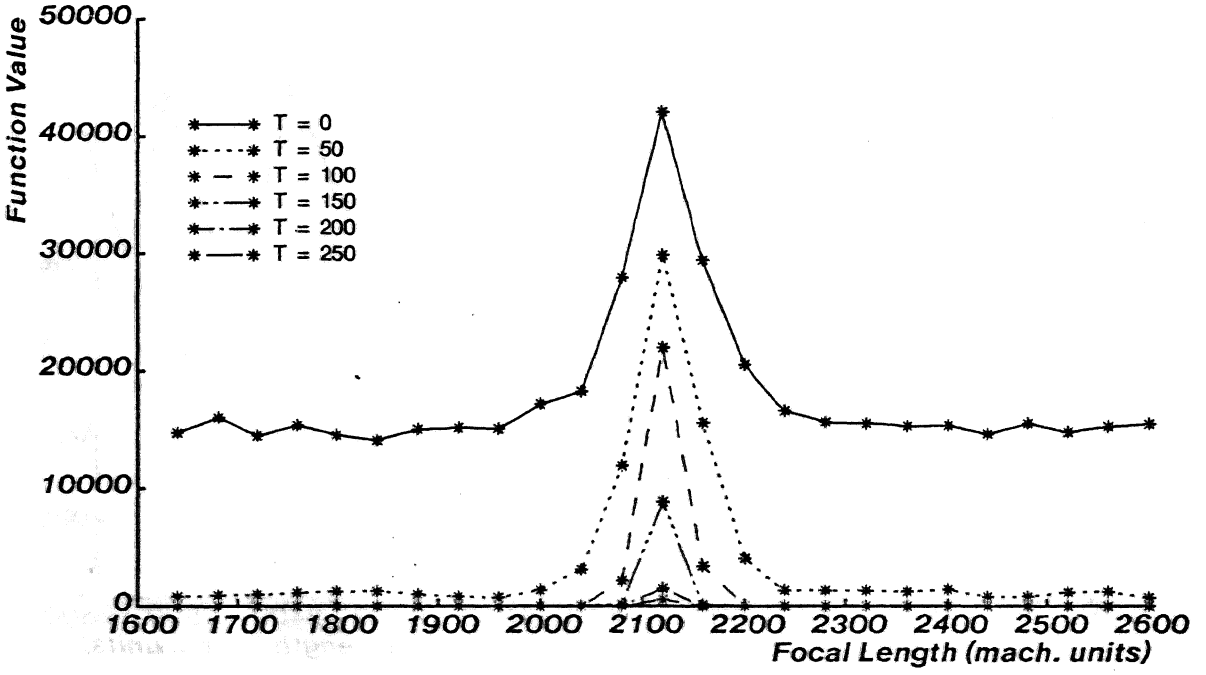


Figure 4-6: Static behavior: the high pass filter on text.

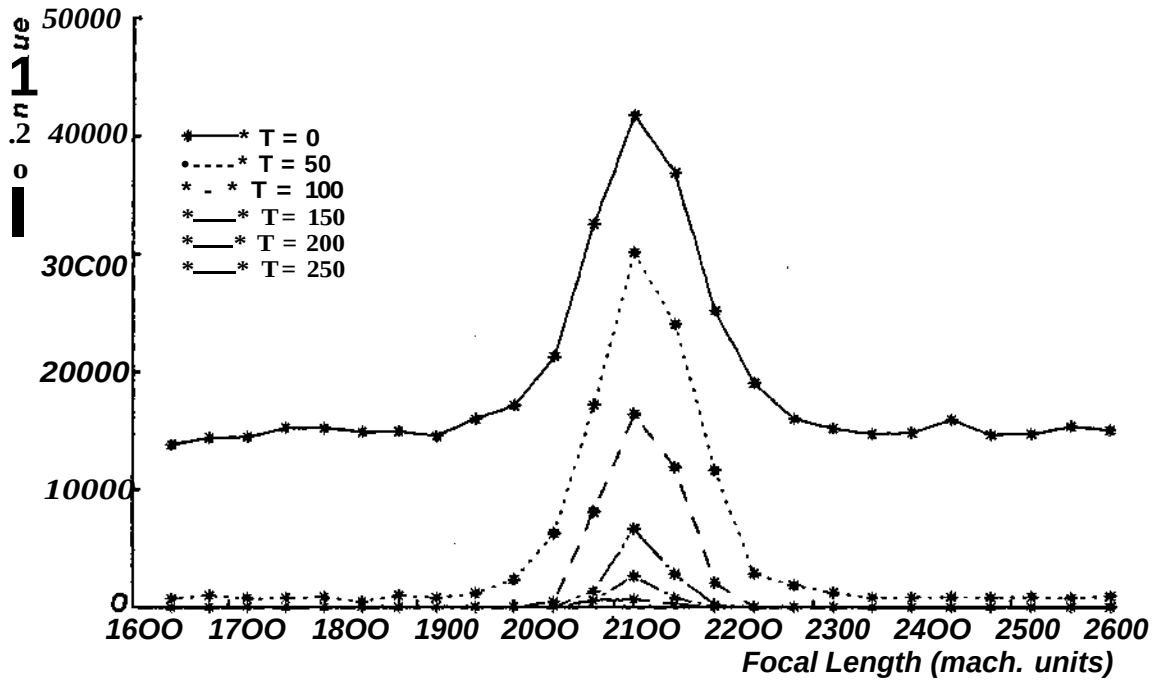


Figure 4-7: Static behavior: the high pass filter on texture.

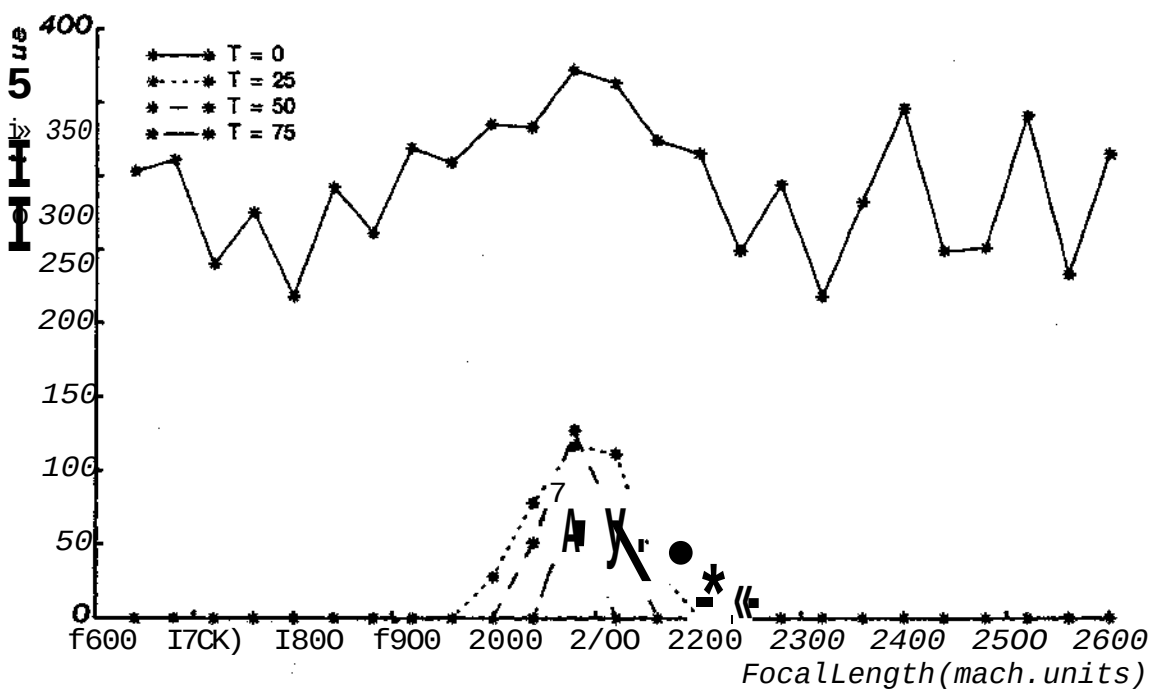


Figure 4-8: Static behavior: the simple cross on an edge.

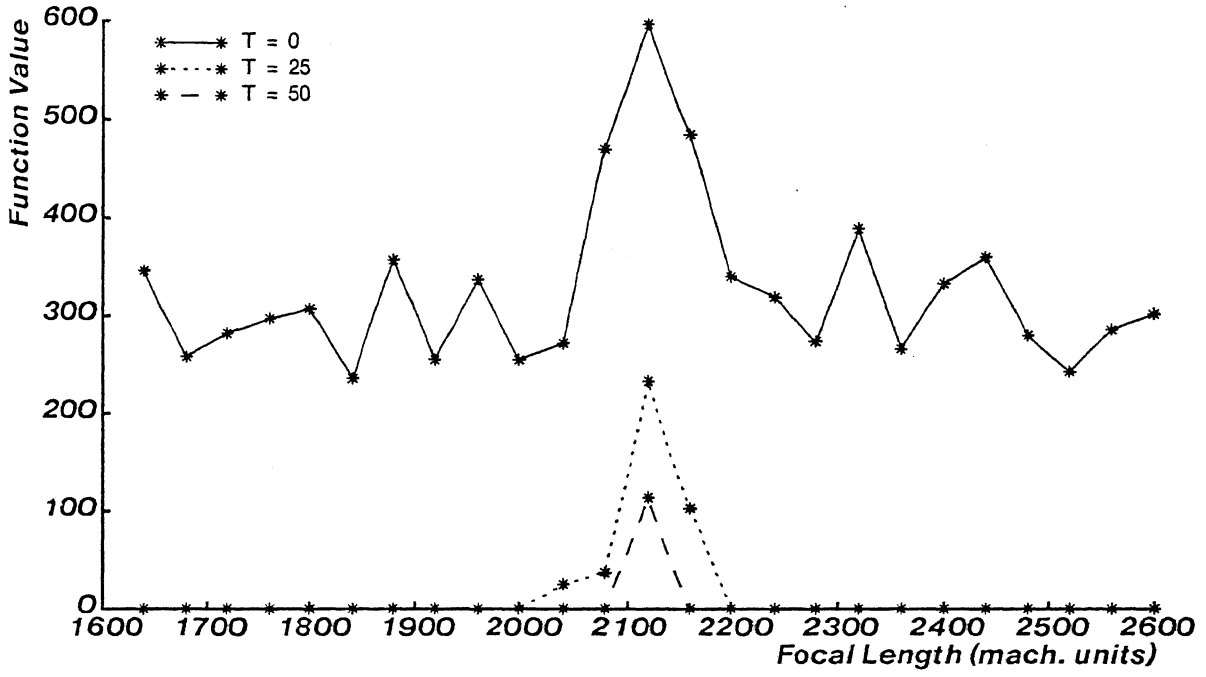


Figure 4-9: Static behavior: the simple cross on text.

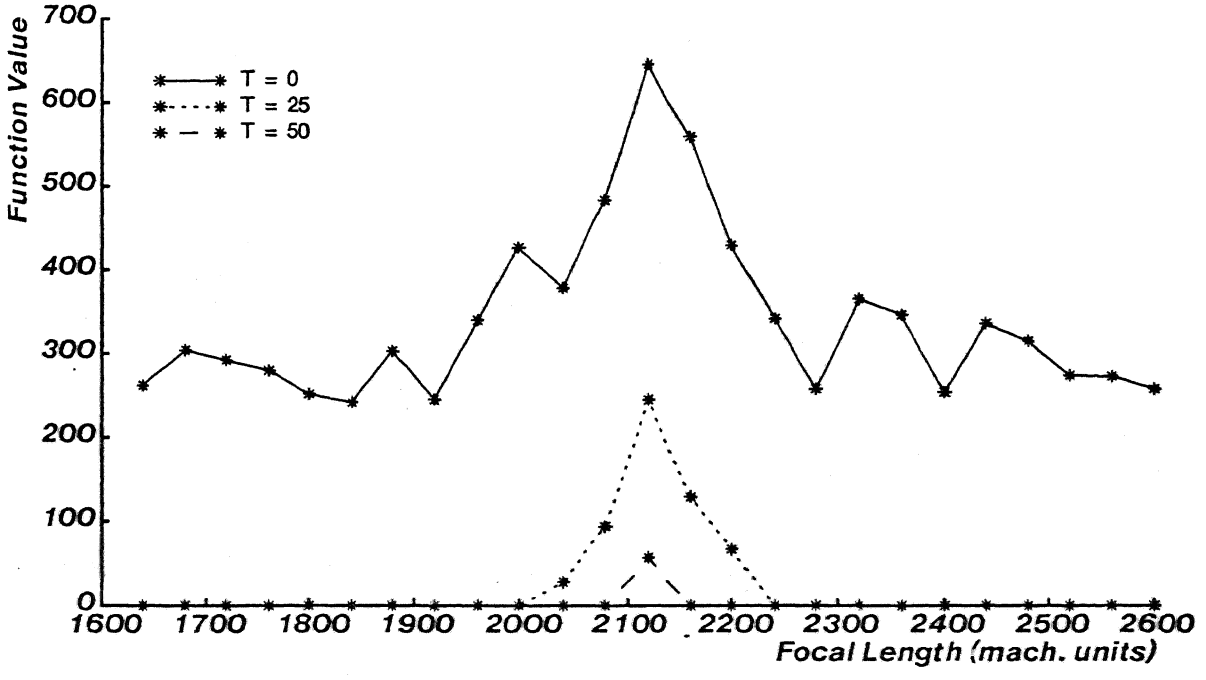


Figure 4-10: Static behavior: the simple cross on texture.

The data from the high pass filter and simple cross algorithms (Figures 4-5 through 4-10) exhibit the same basic behavior as the Tenengrad, but suffer from the noise sensitivity typical of Laplacian operators.

4.3. Introduction to Dynamic Focusing

In a production vision system, an automatic focusing subsystem would be expected to do two things: achieve and maintain good focus. The first is possibly an exploratory task, a problem with many possible solutions. The second is definitely a candidate for closed-loop control. Figure 4-11 shows the basic idea.

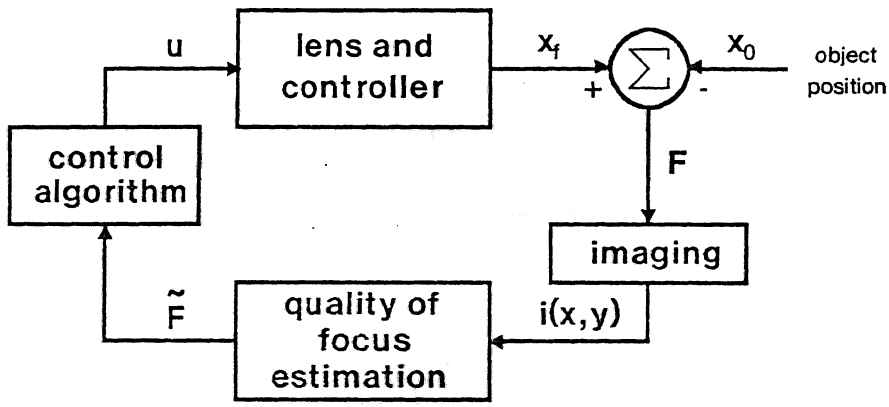


Figure 4-11: Representation of the closed-loop focusing paradigm.

The lens, in a given position, determines a point of best focus x_f (refer back to Figure 3-2 on page 17). The object to be focused on is at position x_o. The difference generates an objective quality of focus F which blurs the image $i(x,y)$. A focusing function scans the image and produces an estimate of F, or the subjective quality of focus $\tilde{F}$. A control algorithm acts on this estimate and produces a signal u which is fed to the lens motor. Finally, the motion of the lens motor moves the point of best focus x_f and the process repeats.

To analyze focus maintenance as a closed-loop control problem with this scheme, the answers to four questions would be needed:

- How does the image $i(x,y)$ depend on the objective quality of focus F? This was answered to a first approximation in Section 3.2.
- How does the estimate $\tilde{F}$ depend on $i(x,y)$? Some suggestions for this were also described in Chapter 3.

- How does the point of best focus x_f depend on u? This is a function of the mechanics and electronics of the lens motor.

- How does (or how should) the control signal u depend on $\tilde{F}$? This is the real crux of the dynamic focusing problem. Before considering this, let's take a look at some previous work.

4.4. Historical Examples of Closed-Loop Image Processing

For the purposes of this section, the term *closed-loop* means the use of results from previous processing to guide the behavior of current processing. It does not mean that the systems described can be analyzed using any or all of the traditional techniques for analyzing control systems.

An Accommodating Edge Follower

Tracking edges in grey-level images is one of the trickier problems in computer vision. Noise, focus perturbations, changes in lighting and other factors conspire to make the process difficult. Pingle, Tenenbaum and other members of the Hand-Eye Project at Stanford developed an edge tracking paradigm which attempted to recover from failures by using accommodation — changing the imaging parameters to achieve better results [10, 21, 34].

A big problem in the Hand-Eye project was hardware: only four bits of grey scale resolution could be obtained for each pixel. The quantization window could be compressed into a subwindow of the full intensity range, which provided higher resolution at the expense of dynamic range. Today's hardware eliminates this problem, but the idea of accommodation will always be timely.

In edge tracking, an edge operator such as the Sobel operator is used first to *find* an edge and then to examine adjacent pixels and follow the edge untils it ends or returns to the starting point. If noise or low contrast decreases the response of the edge detector somewhere along the edge, a simple-minded algorithm will either give up or wander off into the weeds. An accommodating algorithm will pause, evaluate the source of the difficulty, decide on a correction strategy, change the imaging parameters and attempt to restart. The accommodation could consist of opening or closing the iris, zooming in or out, refocusing, inserting a color filter or changing some electrical parameter of the camera.

The Stanford group reported success with the accommodating edge follower and offerred an eloquent rationalization [21]:

> "The performance of an edge follower can be improved by applying more sophisticated processing to a given image or by accommodating to obtain a more appropriate image.

> Most researchers have followed the former course, relying on sophisticated processing to cope with inadequate images. (...) Accommodation attacks the fundamental limitation of image inadequacy rather than the secondary problems caused by it."

Human Face Recognition

Computer recognition of human faces is an example of complicated image processing which depends on structure. Conceptually, the structure of the image guides the processing, rather than the processing determining the structure. In the implementation, however, a program for such a task will often incorporate a feedback mechanism, whereby failure at one stage will lead to the resetting of program parameters and retrial, and success can be used to sharpen the performance of previous stages.

Sakai, Nagao and Kanade developed a face recognition program at the University of Kyoto, Japan, which used a feedback strategy to recover from failures in analysis [27]. Again, this was not a control system in the classical sense, but showed the benefits which accrue from accepting the fact that everything doesn't have to be done perfectly the first time around.

Image-Based Visual Servo Control

More recently, Sanderson and Weiss at Carnegie-Mellon University have developed a means of computing control signals for a robot from a relational graph representation of an image [30]. Figure 4-12 shows the whole story, where the control signal generation takes place in the last stage. The input to the control stage is an attributed relational graph and the output is a vector of control signals[7].

Sanderson and Weiss consider simple polyhedra stationed on a rotary table observed by a fixed camera. The control signal is a voltage applied to an armature controller DC motor which rotates the table. Alternatively, the camera could be mounted on a robot arm. In either case, the control signals cause the relational graph to change, hopefully driving it toward a desired reference position.

The generation of error signals is formulated specifically as a problem in adaptive control, making this one of the first abstractions to combine the techniques of image processing and control theory.

---

[7] The early stages of the process which develop the relational graph are described in [5].

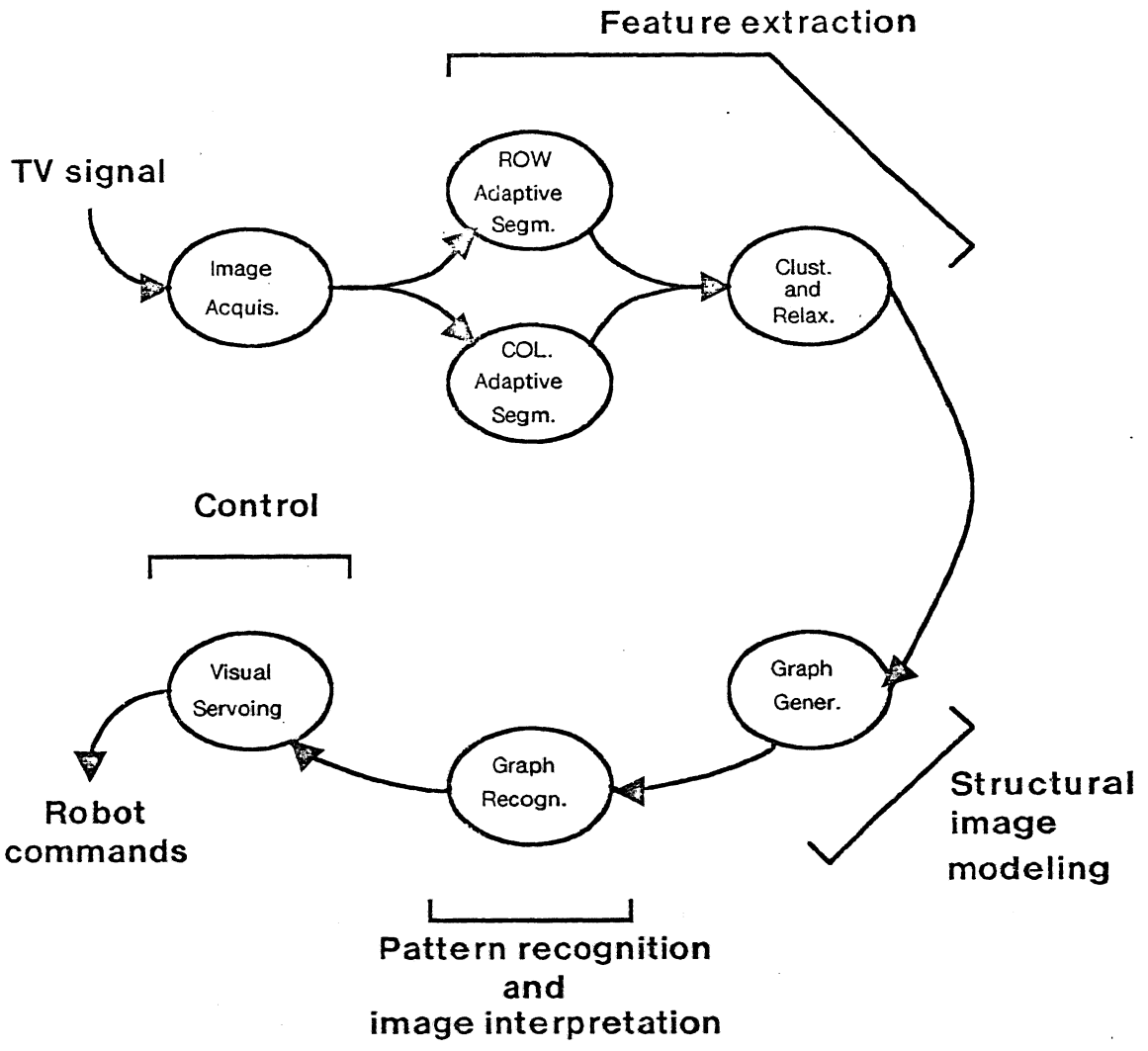


Figure 4-12: A complete closed-loop image processing paradigm.

4.5. Dynamic Focusing Procedure: Hill-Climbing Revisited

We now return to focusing. An important point is that none of the algorithms described in Chapter 3 and evaluated in Section 4.2 provide directional information. From one measurement, it is impossible to decide which way to move the lens to improve the quality of focus. To get directional information, it is necessary to take two measurements and differentiate. In terms of Figure 4-11, this suggests a control algorithm

$$u_n = k_d(\tilde{F}_n - \tilde{F}_{n-1}) \tag{4.1}$$

where $\tilde{F}_n$ and $\tilde{F}_{n-1}$ are the current and previous estimates of F, u_n is the current value of the control signal and k_d is the derivative gain constant. The focusing strategy then degenerates to the hill-

climbing strategy described back in Section 3.3. The dynamic focusing program developed for this project uses a constant step size rather than a gain factor. That is, after each evaluation of the focusing function, the lens is moved a fixed distance in one direction or the other. This is done to avoid the derivative kick typical of derivative control algorithms.

Other factors which impede or preclude the development of dynamic focusing as a control system in the classical sense include noise in the input, time delay in controlling the lens, quantization error, and, most importantly, the intrinsic nonlinearity of the focusing algorithms.

The program which demonstrates dynamic focusing is invoked with the following parameters. Again, the default values used to produce the graphs are shown in parentheses.

- The number of frames to be averaged at each point (4).
- Which algorithm to use (Tenengrad). The only data which indicate possible success with a hill-climbing algorithm are the data from the Tenengrad evaluation (Figures 4-2 and 4-3). The others all contain too much noise.
- What threshold to use, if appropriate (0).
- Whether to run synchronously or not (yes).
- What sampling period to use, if appropriate (2 sec).
- What stepsize to change the focal length by in climbing the hill (20 machine units).

4.6. Results

Figures 4-14 and 4-15 show the data obtained from the dynamic focusing trials. These results suggest at least two things. First, in a production system, the constant control algorithm (fixed step size) should be replaced by the derivative control law (Equation (4.1)) with an appropriate constant, and perhaps incorporate clipping of the difference signal to avoid derivative kick. Making the control signal variable would eliminate the oscillations about the point of best focus seen in Figure 4-15. The problem is that the peak in the focusing function is so sharp that even small movements of the lens produce significant changes in the function.

Second, for focusing on a motionless object, dynamic focusing is unneccesary. A better scheme would be to use a simple but quick focusing function such as the simple cross to land in the general vicinity of best focus, and then to evaluate a more powerful function over a narrow range of focus to finish the job. Chapter 6 will have more to say about implementing a production focusing system.

Another important point is the length of time taken by the hill-climbing routine to reach the vicinity of the point of best focus. This is due to the expense of the Sobel operator.

A loose handle on the computational costs of the focusing algorithms can be obtained a priori by simply collecting operation counts. On a per pixel basis, the Sobel operator used in the Tenengrad algorithm (page 28) requires 4 shifts, 11 adds and 2 multiplications before taking the square root. (Recall that the multiplications by 2 can be accomplished by shifting.) For the Motorola 68000, a shift is approximately the same cost as an addition, and a 16 by 16 bit multiplication is approximately ten times the cost of both. This results in an approximate unit cost of 35. The eight point Laplacian mask (page 34) requires 1 shift and 8 adds, or 9 units, while the one dimensional masks used for the simple cross tally to 1 shift and 2 adds, or 3 units. These numbers are summarized in Figure 4-13.

Operator	Unit Cost
Sobel operator	35 + square root
8 pt. Laplacian	9
2 pt. Laplacian	3

Figure 4-13: Basic per pixel computational cost of the focusing operators.

These numbers can be considered only as rough estimates for two reasons. First, simply accessing the pixels and moving them in and out of registers represents a substantial portion of the computation time. In the Laplacian filter cases, this load can actually outweigh the other operations. Second, if the squared magnitude of the Sobel operator exceeds the threshold, the square root must be taken before adding the result to the sum. This extraction dominates the calculation of the operator.

In terms of performance, the bottom line is the quality of the image when the focusing algorithm is through. Given that two algorithms both produce satisfactory results, the faster of the two is certainly to be preferred. As mentioned previously, all of the derivative-based algorithms behave satisfactorily for static scenes even without smoothing the data. The only remaining question is sensitivity to noise, which is answered succinctly by the fact that the Tenengrad algorithm is the only one that worked without smoothing.

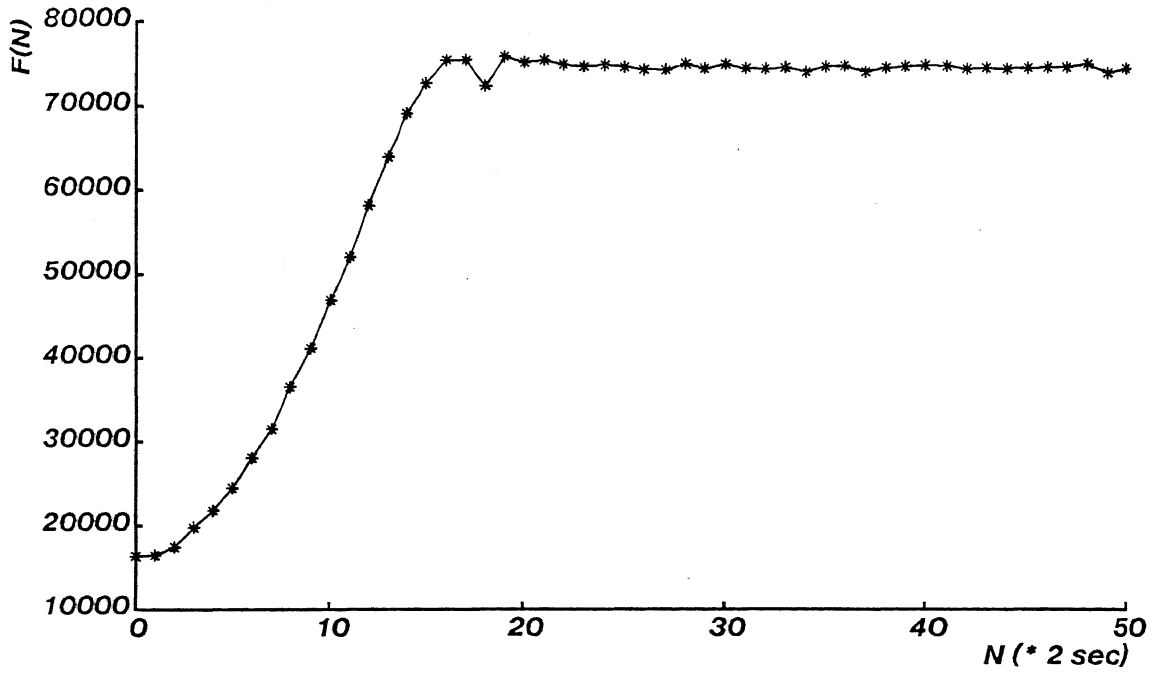


Figure 4-14: Dynamic Behavior: the Tenengrad on texture.

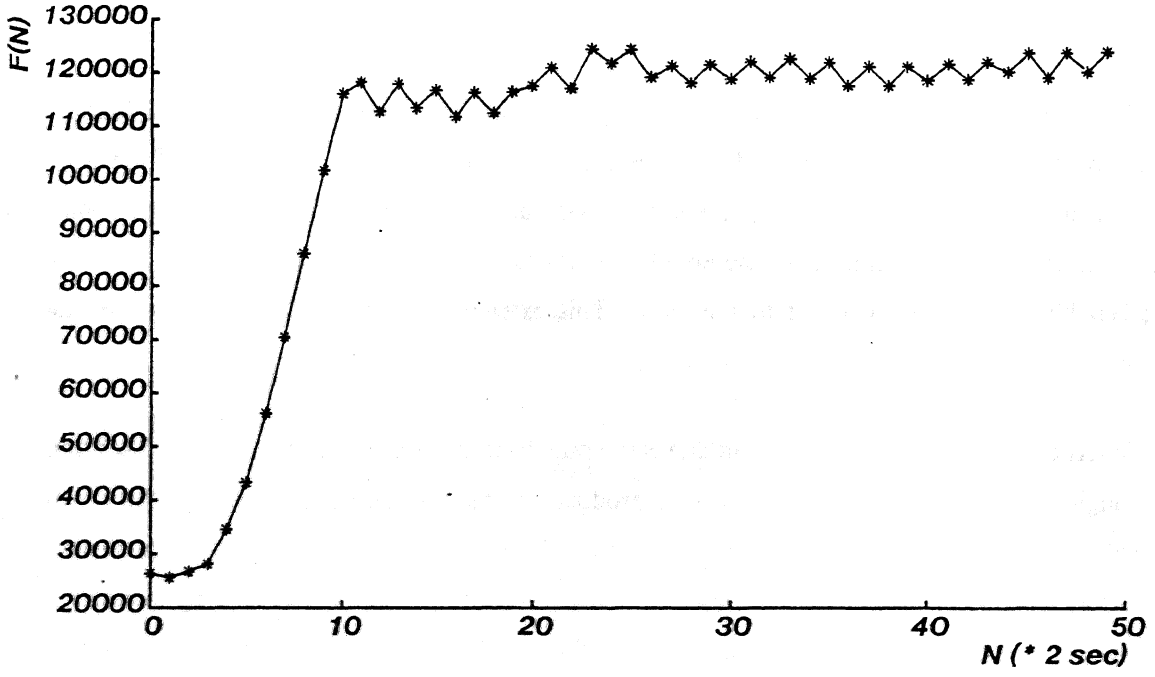


Figure 4-15: Dynamic Behavior: the Tenengrad on a circuit board.

Chapter 5
Industrial Applications

This chapter describes the application of the POPEYE *system to the problem of automated fluorescent lamp inspection. After an introduction giving some background on the lamp project, the problems of filament inspection, flare inspection and polaroscopy are described. The work done to solve these problems is then described in detail.*

5.1. Introduction

The project under which the activities described in this chapter have taken place is called the *Factory of the Future*, an attempt to usher in the third industrial revolution – Robotics. There are three major components in the Factory of the Future project: process modeling and control system design [25], intelligent sensor development [13, 16, 8] and the Intelligent Management System [11].

Process modeling is concerned in part with figuring out exactly how all the machines work and how they affect the finished product. Hypotheses are formed, experiments are carried out and data are collected and analyzed. Sensors are needed to watch what the machines do, and *intelligent* sensors are needed to figure out why they're doing it. The Intelligent Management System takes the diagnostic information provided by the sensors, organizes it in a meaningful way, and then either gives it to a human manager for perusal, makes its own decisions, or both. The process models, the sensors, the IMS and the human managers, together with feedback lines to control the machines keep things running at maximum productivity. This is the goal of the Factory of the Future project, and one of the first testing grounds was to be the Westinghouse fluorescent lamp factory in Fairmont, West Virginia.[8]

Several intelligent sensor systems have been developed so far [13, 16, 8]. The work described below is a continuation of Maddox' investigations [16].

---

[8]The Westinghouse fluorescent lamp division has since been sold to North American Phillips Corporation. At the time of writing, it was unclear whether the project would continue. The remainder of this chapter ignores this development.

Figure 5-1 is a line drawing of a fluorescent lamp mount. This is a necessary but invisible part of every successful fluorescent lamp. Straight lamps have one of these in each end to excite the gas inside the tube. Any defect in a mount will doom to destruction the lamp of which it becomes a part This is the point of inception for mount inspection.

The set of possible defects is cleaved in three: electrical defects, filament defects and glass defects. Electrical defects such as problems with the lead wires and filament connections are left to "hard" automation (inflexible machines). Filament and glass defects are described in the following sections.

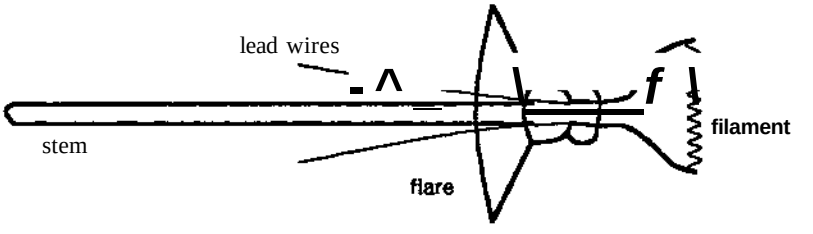


Figure 5-1: A line'drawing of a fluorescent lamp mount

5.2. Filament Inspection

The fluorescent lamp filaments are wire wound coils which are dipped into an emission material.[9] After that step in the process, several problems can arise. If the emission material deposited on the coil is too thick or too thin, the lamp into which the filament is inserted will die an early death. Figure 5-2 shows a binarized image of a lamp filament sporting an extra drop of emission material. The drop forms under the influence of gravity before the liquid dries. (The coils are inverted before dipping.) The presence of two coils or no coils on the mount structure will also cause a lamp to expire. The problem, then, is to look at a filament structure, decide whether it is acceptable or not, and if not, to determine the type of defect

Maddox tackled this problem with a binary vision system and conjectured that perhaps binary vision was not enough. To aid the decision making* the FOPEYE system described previously was used to see what extra classification performance a grey-level vision system could provide.

The type of defect shown in Figure 5-2 was precisely the type that confused the binary vision

[9] The majority of the wort on filament inspection was done by Maddox. For a detailed discussion of filament aad g t e Inspection, the reader is referred to [16J,

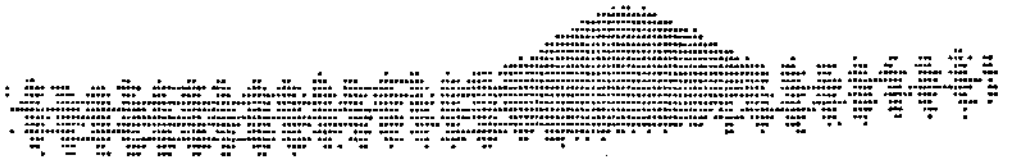


Figure 5-2: A binarized image of a fluorescent lamp filament.

system. Here's why. Two of the parameters used in the classification of defects were the width and height of the bounding box of the filament, so the machine was very sensitive to small anomalies in shape. A bounding box with a large height could be due to the type of defect shown in Figure 5-2, or it could be due to the presence of two coils on the same mount structure. Since it was not possible to reprogram the binary system, we turned to the POPEYE system for both its grey scale capability and its versatility.

The first algorithm implemented was a height vs. horizontal position extractor which worked by binarizing the image and finding the height of the filament section at each point along the horizontal axis. Small bumps on the filament would then show up only as bumps in the height vs. position graph, rather than perturbing the entire bounding box estimate. Figure 5-3 shows the plot of height vs. position for the filament in Figure 5-2. Note that the grey level capability of the POPEYE system was *not* used in this algorithm — only its programmability.

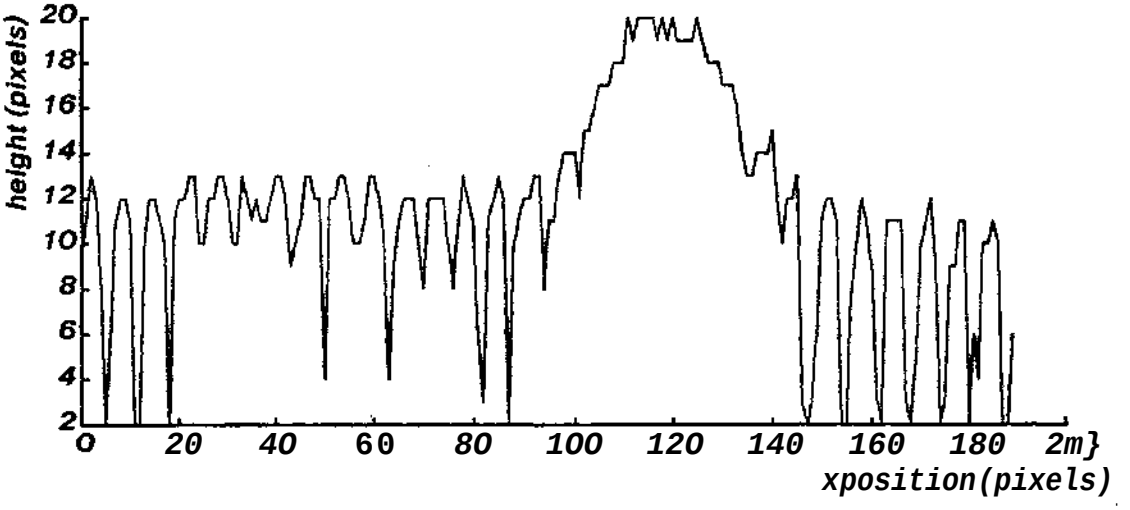


Figure 5-3: The running height of the filament in Figure 5-2.

There are two advantages to this algorithm. First, by using the ratio of maximum to miaimum or maximum to average height rather than the height of the bounding box as a feature, classification performance is improved. This feature is coupled tightly to the occurrence of the type of defect

shown in Figure 5-2. Second, computation of the filament height at each position along the horizontal axis removes the effect of curvature from the feature calculation. The filament in Figure 5-2 is pretty straight from end to end, but if a filament with significant curvature shows up, the bounding box estimate will be too large. Subsequently, the classification may fail.

The second step in filament inspection was to look at the texture of the filament. Since the filaments consist of a dark wire wound coil dipped in a bright emission material, there is usually a periodic intensity variation over the surface. In the interstices between adjacent loops of the coil, the emission material fills in the cracks, and so the intensity is high. Near the loops, the dark wire shows through a bit, and so the intensity is lower (refer again to Figure 5-2). If an extra drop of emission material remains on the coil, or if the emission material is too thick, the periodic intensity variation will disappear. Figure 5-4 shows a plot of the intensity along the midline of the filament in Figure 5-2. The disappearance of the periodic intensity variation coincides with the location of the lump.

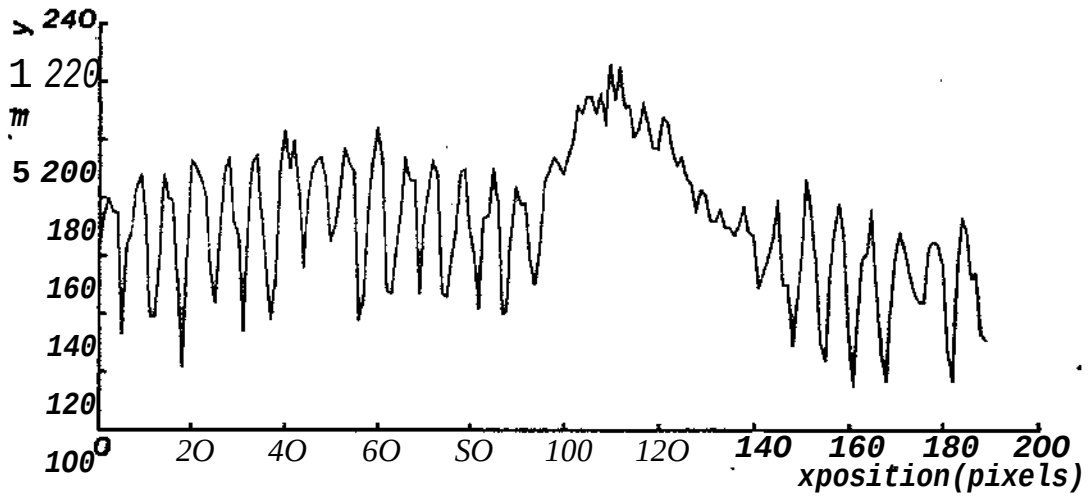


Figure 5-4: The intensity along the midline of the filament blob.

The absence of the periodic intensity variation can be emphasized by taking the variance of the intensity function over a sliding window. This function is plotted in Figure 5-5. The variance can be seen to drop where the lump occurs. An eight pixel window was chosen as a convenient and reasonable compromise between accuracy of estimation and good tracking of the trend toward decreased variance.

An advantage of this scheme is that the extra emission material does not have to form a large blob to be noticed, but only has to obscure the periodic intensity variation. On the assembly line this is occasionally the case, and indicates that worse things are soon to come. By noticing such a defect

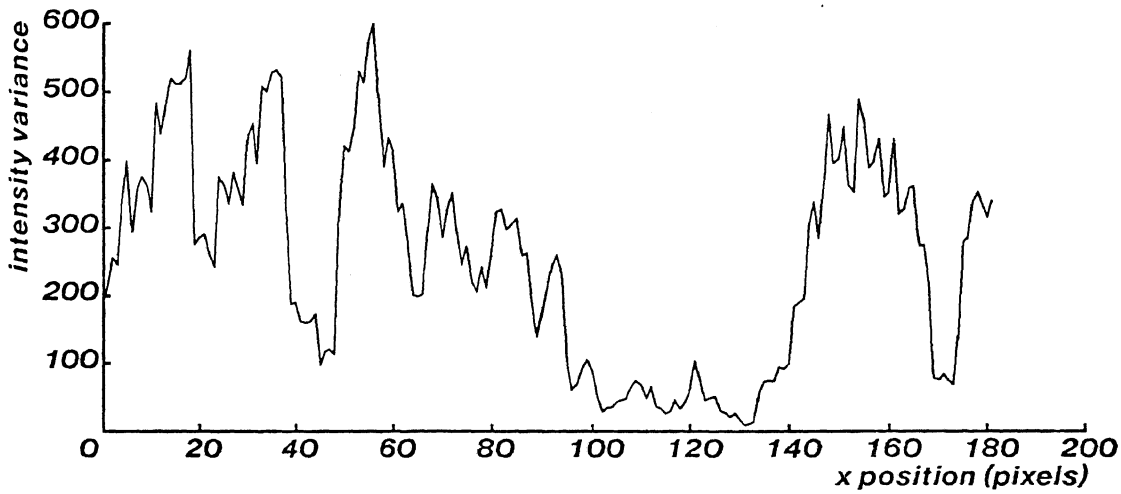


Figure 5-5: The variance of the intensity values using an 8 pixel window.

and informing a central data facility of its occurrence, it would be possible to change the parameters on the assembly line before a large amount of defective filaments were generated. This is one small example of the potential interaction between a process, an intelligent sensor and the IMS.

The filament inspection story is important in the big picture of this project since it was the first applications package to be written on the POPEYE system. In addition to demonstrating the feasibility of the industrial inspection concepts to the funding organization, this project was the dominant factor behind the development of many of the amenities now present on the system. Several of the routines written for this project found their way into other places, both in later inspection projects and in theoretical work.

The conclusion in the case of filament defects was that a programmable binary vision system could do an adequate job on the assembly line, and so the work of specifying and ordering a system began.

5.3. Flare Inspection

At the start of the mount inspection project, the goal was to use a single vision system to look for all the different things that can go wrong with a mount: electrical defects, filament defects, and glass defects. This lofty goal was promptly subdivided into smaller goals. As mentioned earlier, the lead wire defects were left to hard automation. The previous section together with Maddox' work [16] dealt with filament defects, and this section deals with glass defects. In the case of glass defects, it became evident that grey scale imaging was necessary, and so work continued on the POPEYE system.

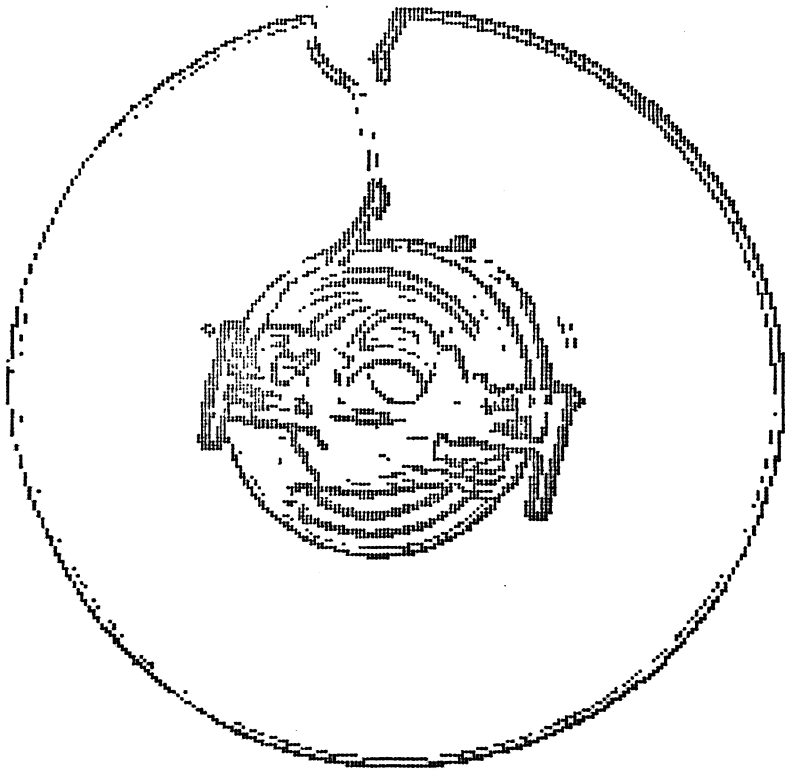


Figure 5-6: A top view of a glass mount showing both a chip and a crack.

Figure 5-6 shows a top down view of a fluorescent lamp mount. The filament assembly is now in the center of the image, and is essentially noise to be ignored.

Glass defects come in three flavors: chips, cracks and ringouts. All of these affect the edge structure of the image. In a normal mount, there is a blotch due to the filament assembly in the center, surrounded by a circular contour that delineates the outer radius of the glass flare. The dark edge arises from destructive interference between light rays passing through and around the glass. In a chipped or cracked mount, spurious edges arise inside the outer radius. In the special case of ringouts, an entire outer portion of the flare falls away, leaving a jagged contour which lies completely inside what would have been the outer radius of the flare. The problem of detecting these defects, then, is one of edge detection. Classification of the defects is a more complicated issue which is beyond the scope of this project and has not yet been treated.

The task of examining a mount image for bogus edges can be decomposed into two pieces: figuring out which points to examine, and then examining each point. Since the flares are circular when viewed from the top, it makes sense to examine an annular region of the image. The outer radius of the annulus lies just inside the outer radius of the flare, so the outside edge won't be

confused with a defect, while the inner radius lies just outside the filament area. If any edges are found in this annular region, the mount can be pronounced defective. In the first implementation of this idea, the annular region was generated by using a circle algorithm to construct a series of concentric circles of appropriate radii.

Once we decide which pixels to examine, we need a way of deciding whether a crack intersects the pixels. This is easily done with an edge detector. Several of the multitude of edge detection algorithms are described below, in order of decreasing computational complexity [22].

- THE SOBHL EDGE OPERATOR. One of the best (and the slowest) in the business, the Sobel edge operator estimates the magnitude of the intensity gradient at each point The masks shown in Equation (5.1) are used to estimate the directional gradients at each pixel. The magnitude of the resultant gradient is computed from these estimates via Equation (5.2).

$$i_x\Big|_p \approx \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1. \end{bmatrix} \qquad i_y\Big|_{\tilde{P}} \approx \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{5.1}$$

$$S(p) = \sqrt{i_x^2 + £^2} \tag{5.2}$$

  Of particular value is the fact that its output is relatively independent of the orientation of the edge. (Figure 5-6 was actually produced by edge enhancing a grey scale image using the Sobel operator and binarizing the result.) In practice, the square root is used only for display purposes. When comparing the output of the operator to a threshold, it is often less expensive to square the threshold.

- THE ROBERTS CROSS OPERATOR uses simpler estimates for the directional gradients (Equation (5.3)) but computes the resultant gradient using the same formula as the Sobel operator (Equation (5.2)). The gradients are computed in the diagonal directions s and / rather than the vertical and horizontal directions *x* and *y*. The current pixel is taken to be in the upper left corner rather than in the center.

$$i_s\Big|_p \approx \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \qquad i_t\Big|_p \approx \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{53}$$

- HIGH PASS CONVOLUTION KERNELS. Since an edge gives rise to high frequency components in the image, high pass filters can be used to find them. The two most popular convolution masks used to find edges are shown below. The output of each is used directly.

$$4\ point: \quad L(p) \approx \begin{bmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{bmatrix} \qquad 8\ point: \quad Up)* \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \tag{5.4}$$

- ONE DIMENSIONAL DIFFERENCING. This is one of the cheapest and quickest ways of looking for edges. The masks used are one dimensional instead of two dimensional and

are therefore sensitive only to edges perpendicular to **the** direction of differencing. For example, the horizontal mask [-1 2 - 1] will respond only to edges with reasonable slope. One possible use for this method is in vision systems which perform horizontal and vertical analysis separately [5].

The flare inspection demonstration program provided a choice of algorithms, making it possible to trade quality of edge detection for execution speed.

At each point in the annular region, the edge detector returns a number related to the edge strength at diat pixel. The edge strength is compared to a threshold based on the noise present in the image. A count is made of the number of pixels whose edge strength meets or exceeds the threshold. Lastly, the edge count is compared to another threshold to decide whether the glass flare is defective or not. Figure 5-7 is a scatter diagram made by plotting the edge count for each sample and labelling the data points to show which samples were cracks, chips, or ring-outs. As expected, the good samples separate from the defective ones, but the three classes of defects fail to separate from each other.

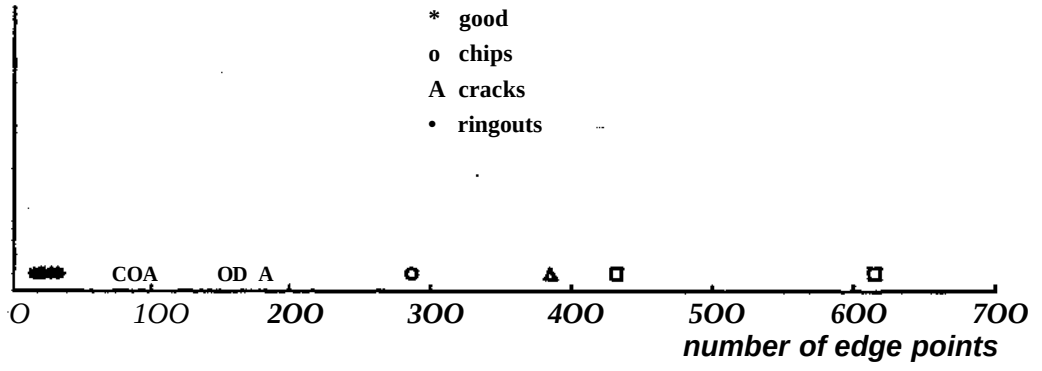


Figure 5-7: A scatter diagram of the data obtained from edge point counting.

Before discussing possible methods of helping the separation, let's look at some of the implementation details that complicate the edge point counting scheme. First, commercial frame buffers typically display their images using Cartesian coordinates. This mathematical tyranny is desirable in most applications and tolerable in many othere, IH the mount situation, however, the objcxts of interest are circular, so we're feced with either looking for a polar coordinate display or making due with what we have by using rectangular to polar coordinate transformations. The extra processing time involved in coordinate transformation would make the inspection process much too slow.

Second, in cheaper frame buffer systems, the pixels are rectangular. The MATROX unit divides

the screen into 256 lines of 256 dots each. Common sense says the display area should be square, but this is not always so. The aspect ratio of the standard television monitor is 4:3, which means that the screen is 3 as wide as it is tall. To accommodate this fact, cheap frame buffers deform the pixels accordingly. Consequently, the circles drawn on the screen by even a mathematically perfect circle generator come out looking elliptical. This means that circles won't exactly fit inside the area of the glass flares. Figure 5-8 illustrates this problem. Since the pixels are horizontal rectangles, the "circle" comes out as a horizontal ellipse. The small chip in the upper right section of the glass is missed completely, and the minimum size of the ellipse is limited by the filament blotch in the center.

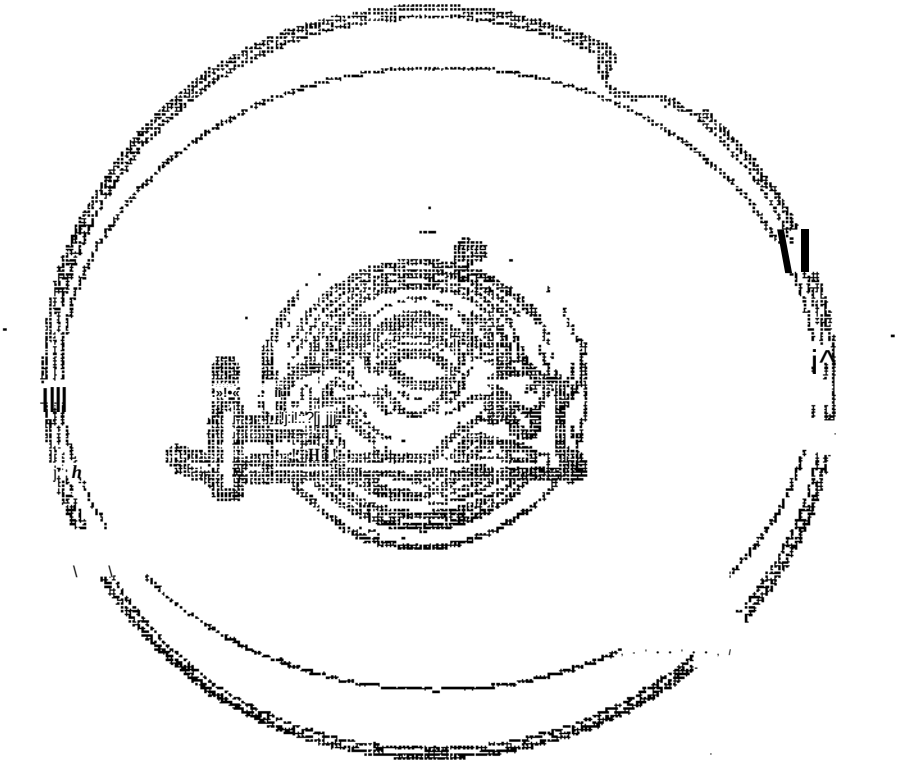


Figure 5-8: Dlustration of the ellipse problem in glass inspection.

In general, when a problem arises, it can either be solved or avoided. To solve the rectangular pixel problem, a polar coordinate frame buffer system would be needed. Since this is not available, we avoid the problem instead. The application is specific enough to guarantee both the placement and the orientation of the flare to within one pixel, so we have a fairly specific idea of which pixels to search for edges. By generating a map such as the one shown in Figure 5-9 and storing it in a compacted form such as ran length coding, we skirt the problem of imperfect circle generation. Each pixel posidon in the map is still examined by the edge detector, so only the method of generating the points has changed. The method of examination remains the same.

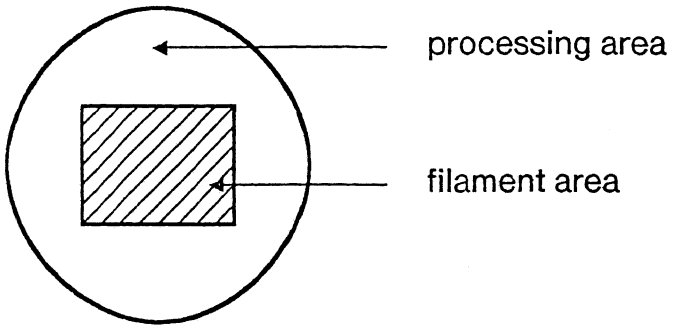


Figure 5-9: The template necessary for examining glass flares.

Before concluding the discussion on glass inspection a few words on classification are in order. As mentioned earlier, the problem of classifying defects is more involved than simply detecting them. Detection is a good start, and provides the assembly line with the information necessary to reject bad parts, but if we want to stop the *production* of bad parts, we need to classify. To classify, we need something more revealing than an edge point count. Although edge point counting gives us a fairly good idea of the length of the edges, it tells us nothing about their location or structure.

The edge structure of a chipped flare is different from that of a cracked one and the structure of a ring-out is different from both. To differentiate between the three classes, we need not just an acknowledgment of the edges, but a structural description. Once an edge description has been produced, classification can be attempted. Mathematically, the change is from *statistical* to *syntactic* pattern recognition.

Edge description is not without its problems, however. First, it is expensive and therefore slow with current processing technology. On an assembly line, even a description of defects whose pattern of occurrence is random is of questionable value. Second, edge tracking is extremely sensitive to lighting conditions. This makes the job even more difficult.

During the course of the flare inspection project, it was decided that since the flare production process is difficult or impossible to model, energy spent on classification of flare defects would be wasted. Hence, algorithm development for the project was terminated after a reasonable detection scheme was demonstrated.

5.4. Polaroscopy

This section is included for historical, rather than academic completeness. At the beginning of this project, one of the aims of the Westinghouse Lamp Project was to build a system which could extract information from glass mounts using polaroscopic inspection. During the project, other concerns were deemed more important. The results from these "digressions" have been presented in the previous sections. Only toward the end of the project did attention return to polaroscopy.

Often, the structure of a glass or plastic object will be under considerable stress. If the object is heated or handled indelicately, it may break. Or, it may wait a while and break from fatigue. It is suspected that a large amount of shrinkage (loss) in fluorescent lamp and light bulb production in general is due to stress. The signs of stress, however, are not usually visible under ordinary illumination. The way to see stress in a transparent object is to position the object between parallel plates of polarizing material whose axes of polarization are set at right angles to one another. Illumination comes from behind the rear plate, so the only light which penetrates the front plate is light which has been repolarized by anomalies in the refractive index of the object.

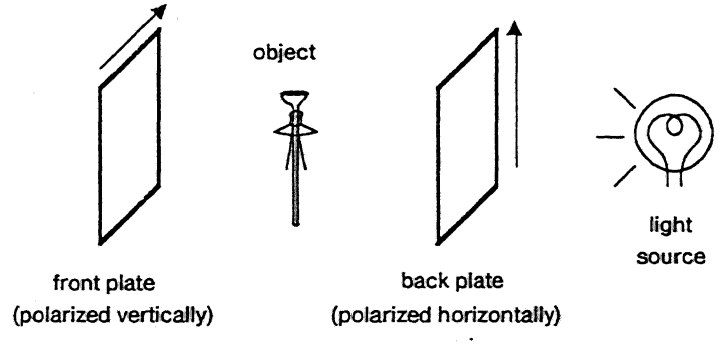


Figure 5-10: The configuration of a basic polaroscope.

A crude prototype polaroscope was fabricated using two 2′ by 1′ sheets of polarizing plastic normally used for photography and a fluorescent light box for illumination. This was sufficient to demonstrate the concept for objects made of injection-molded plastic, where the stress is often large, but was inadequate for revealing the subtle stresses more characteristic of glass.

Several improvements upon the configuration in Figure 5-10 are necessary. First, the entire assembly must be enclosed in a light-proof box so that the effects of ambient light can be eliminated. Second, high quality polarizing material should be used, and third, the object under scrutiny should be submerged in a fluid whose refractive index is approximately equal to that of glass. This eliminates artifacts which arise because of variations in the thickness of the glass.

Two sheets of Polaroid high quality polarizing plastic were purchased for incorporation in the second polaroscope. At the time of writing, however, work has been halted on the project.

Chapter 6
Conclusions and Suggestions for Future Work

This final chapter appraises the results of the project in light of the original objectives given in Chapter 1. First, the usefulness of the POPEYE *computer vision system is proven by example. Conclusions are drawn from the work on automatic focusing and suggestions are made for possible extensions to the work. Some complaints are raised about current state of the art in computer vision hardware and finally, a request is made for an 'ideal' sensor.*

6.1. Evaluation of the POPEYE System as a Research Tool

The usefulness of the POPEYE vision system is best attested to by an enumeration of the applications which have used it.

- FLUORESCENT LAMP MOUNT INSPECTION. This was described in Chapter 5.
- AUTOMATIC FOCUSING ALGORITHMS. These were described in Chapters 3 and 4.
- GRADIENT SEGMENTATION. Basic research in image segmentation based on gradient rather than intensity magnitude is being carried out by Sanderson and Bracho [31]. Theoretical predictions have been made concerning the shading across basic Lambertian surfaces and compared with data from the POPEYE system.
- FLEXIBLE ASSEMBLY. Binary connectivity algorithms similar to those of the MIC vision module have been implemented as part of the Westinghouse-CMU flexible assembly project. The package of routines is available to any application program and has also been ported successfully to the Vax. As a front end to connectivity analysis, histogram features are used to automate the choice of a binarization threshold. The adaptive binarization and connectivity are part of a larger effort to determine the orientation of transistors prior to insertion in a circuit board.
- HYBRID PATTERN RECOGNITION. A new project in hybrid syntactic/statistical pattern recognition has recently begun which extracts primitives from images in a bottom-up manner and builds an attributed relational graph intended for parsing by a parser for a two dimensional grammar. Especially useful in this context is the grey scale capability of the POPEYE system, which enables the implementation of a robust front end which can handle noisy images.

The principal users of POPEYE, who were also its principal developers, have found the system so convenient and easy to use that energy has been devoted to specifying and implementing second generation software. Some of the new projects planned or underway are:

- A completely new monitor for the 68000-based processor board which exploits the user/supervisor discrimination and tracing capabilities of the 68000. This will make the system more reliable and significantly improve debugging.

- Reimplementation and extension of the object level support layer. Several dubious design decisions from the first generation package have been revamped and additional support for high level object types has been added.

- Reimplementation of the standard command interpreter.

- A new standard shell program which provides interactive access to the capabilities of the object level support layer.

In summary, the POPEYE system provides a convenient base for the rapid design and testing of computer vision algorithms for research and development applications. By virtue of its grey scale capability, programmability, modularity and embedment in a large resource network, it makes possible the solutions to problems which confound currently available commercial vision systems.

6.2. Suggestions for a Production Focusing System

The implementation of automatic focusing algorithms described in this report has resulted in a significant enhancement in the capability of the POPEYE system and has provided a starting point for the development of a production quality focusing system. This section provides some suggestions for such a system.

Ideally, the user of a computer vision system should never have to worry about focusing a camera. The system should insure that that the camera is in focus at all times. To achieve this in the general case, a focusing subsystem needs input information. A higher-level problem solving algorithm, be it software or human, must tell the system what to focus on. In simple planar scenes the point is moot, since everything is at the same depth. This is generally not the case, however. In many industrial inspection settings the scene is complex from a focusing standpoint, even after being constrained for lighting and geometric arrangement considerations. The work on automatic focusing described in previous chapters has simply avoided the issue by requiring the user to define a processing window. A production system would pass a definition of a processing window down to the focusing module.

Another issue in focusing is how powerful the algorithm should be. Succinctly, the lesson learned from this work is that optimality is not a consideration. There are cheap algorithms that can do a fine job. Some situations may require more expensive operators than others, such as subtle texture in a low contrast image, so several operators should be made available. As to the type of operator, high pass filters (Laplacian masks) and edge detectors seem to be the way to go. The information in histograms is inadequate and the information in frequency transforms is redundant and expensive to compute. In addition, if more than one operator is available, the focusing module has the option of switching from one to another in midstream.

A consideration closely related to the power of the operator is which pixels to process. Often it is unnecessary to process every pixel in the window to get satisfactory results. The user should be given a choice of attacks, perhaps consisting of the following.

- Process every pixel in the window.
- Decimate the window by an integer factor. That is, process every n^{th} pixel.
- Process the horizontal and vertical midlines of the window.
- Process the diagonals of the window.

This type of approach can be generalized by considering one module of the focusing subsystem to be a point generator. The generator would be initialized with the window origin, dimensions and sampling style, and would return the coordinates of the next pixel on each invocation.

Lastly, we have behavior. How should the system act on its way to achieving good focus? Ideally, when a user issues a command to focus, the lens should start from wherever it is and seek directly to the point of best focus limited only by the speed of the servomotor. Using the cheapest algorithm on the POPEYE system (the simple cross), the speed constraint comes close to being satisfied. To attain similar speed with more thorough algorithms, fancy hardware would be needed. The direct seek constraint presents problems which require more intelligence to solve.

It is evident from the graphs in Section 4.2 that the only reliable point along the range of focus is the maximum point, which means that the entire range must be searched to find it. Few of the graphs are monotonic on both sides of the maximum. Those who see the focusing demonstration program run never fail to ask why it continues through the "obvious" point of best focus to the opposite end of the range. This is certainly a valid criticism. To appease these people, which amounts to satisfying the direct seek constraint, a production focusing system should smooth the data over a window wide enough to threaten monotonicity but narrow enough to keep the amount of extra processing negligible.

At the time of writing, there are plans to implement a focusing package as part of the standard software support of the POPEYE system.

6.3. Suggestions for Hardware Improvements

During the course of this document, several complaints about hardware inadequacy have been made. It is now time to suggest some improvements.

The most infuriating feature of the POPEYE hardware is the lack of a memory mapped frame buffer. To access any pixel, two registers which address the rows and columns of the buffer must be loaded before storing or retrieving the pixel value. By mapping the complete frame into the memory space of the processor, acquisition and display of image data could be quickened by at least a factor of three. Such hardware is now available, and there is no reason why a second generation hardware implementation of the POPEYE system could not use it.

Another whole set of annoying hardware problems came from the various pieces of VICON hardware. Both the lens and the pan/tilt head were originally intended for surveillance applications where accuracy and speed are not critical. Hence, the hardware used in constructing them, particularly the motors, is of low quality. A better system would use stepping motors rather than servomotors. Also, corners were cut in designing the feedback systems which control the motors. In the pan/tilt circuitry, account is taken of the high inertia of the moving system not by compensation with integral or derivative control, but by killing the control signals to the motors when the position is a fixed distance away from a desired point and letting the apparatus coast to a halt This results in positioning errors which confound the interface software. The worst problem is that the pan/tilt motors have only two speeds: slow and stop. As usual better performance can be had for indulgence in capital outlay.

Lastly, there is the ubiquitous noise. After going to the trouble of concatenating two four bit frame buffer boards to get eight bits of grey-scale resolution, it is annoying to realize that only five bits are really data. Noise reduction schemes improve this, of course^ but what's really needed is prevention Instead of cure. Ekonbt, a manufacturer of image acquisition equipment, expresses the idea well in Its advertising:

> "Eikonix learned long ago that it Is far better to take a little longer at the front end and collect quality data, than to try and correct for noisy and Inaccurate data later in die image processing."

6.4. A Request for a Smart Sensor: The VLSI Retina

Often, naive observers of computer vision research will ask why computer programs for vision and image processing are so slow. The answer is that all the processing is done sequentially, one pixel at a time, whereas human vision is achieved through massive parallelism. The limitations in computer programs are often imposed not by the algorithm but by the machine. Most computer vision research endeavors utilize standard general purpose computers simply because of their availability. Ironically, many of the standard image processing operators can be expressed as parallel local computations. Because of this, several projects have developed special hardware to perform parallel computations on image data [23].

Hardware for parallel computation typically consists of an array of cells, each of which computes local neighborhood transformations. Since the arrays are two dimensional, highly regular and require only neighborhood connections, they are prime candidates for integration to VLSI. What follows is a loose specification for a sensor chip which would perform the early stages of visual processing in a highly parallel fashion. Hence, it is called the *VLSI Retina**

The first stage in vision is the acquisition of light. A two dimensional cartesian array of photoreceptors should gather incoming light and convert the incident intensity to a voltage or current The integration time of the photoreceptors should be variable, allowing the operator to trade speed for signal quality. A range of integration time from one microsecond to one millisecond should be adequate. The signals should be made available at the back of the array. This stage alone would provide images vastly superior in signal-to-noise ratio over images from conventional sensors.

Subsequent layers of the sensor should consist of arrays of processing cells which receive inputs from all the neighboring cells of the previous stage. Each stage would then have enough information to perform spatial averaging (noise reduction), spatial differencing (spot, edge or line detection), or, with delays, temporal averaging or differencing (movement detection).

It should be possible to concatenate several stages to perform heirarchical processing. At the final stage, the output information would have to be passed sequentially to a processing stage which would convert from an array representation of the information to an abstract representation such as a relational graph.

Appendix A
The Master's Project Proposal

For the sake of convenience, perspective and perhaps contrast, the Master's project proposal submitted to the faculty of the Electrical Engineering Department of Carnegie-Mellon University is reproduced on the following pages. Minor changes informal have been made to allow the inclusion of the document in this project report No changes in content have been made since the submission date. Page 74 is a list of references for this appendix, not for the entire project report.

A CONTROLLABLE CAMERA SYSTEM FOR COMPUTER VISION RESEARCH

MASTER'S PROJECT PROPOSAL

John F. Schlag
Department of Electrical Engineering and Robotics Institute
Carnegie-Mellon University, Pittsburgh, PA 15213

January 12, 1982

In the fall of 1979, various members of CMU faculty, staff, and local industrial organizations combined their resources to form a new research group. The purpose of this group, now referred to as the Robotics Institute, is to conduct advanced research and development in the area of sensing, thinking machines in order to increase national industrial productivity. Two of the primary research efforts in the Robotics Institute concentrate on robotic manipulator control and computer vision. This proposal outlines a master's project aimed at linking these two areas through the construction of a computer controlled camera system.

Robotic manipulator control encompasses such issues as movement algorithms, collision avoidance algorithms, feedback control system design and multi-processing architectures, each of which is currently under investigation here at CMU [1-4]. Computer vision research includes object and pattern recognition, image segmentation, three dimensional shape determination and scene representation. To an extent, the areas of manipulator control and computer vision have been connected. For example, Sanderson and Weiss [1] use relational graph error signals while Hunt [2] uses first moment calculations extracted from images to generate control signals for a robot arm. These visual feedback strategies are needed to construct robots which can reach out and grasp moving objects.

Previous vision systems at CMU have been constrained by the static characteristics of the camera mounts [2,5,6]. These systems were intended to be aimed and focused once, after which they were largely forgotten. While this type of configuration may be satisfactory for industrial inspection applications, it is inadequate for vision research. To relax the camera constraints and give the image processing computer more control over the incoming image, it is proposed to construct a system in which the computer controls the position of the camera, the orientation of the camera and the three basic lens parameters: focus, aperture opening and zoom. Position and orientation control will be achieved by mounting the lens and camera directly on a robot arm (a Unimation PUMA), and by providing a software interface between the computer and arm controller. The position can be determined in cartesian coordinates (x, y, z), and two angular coordinates (tilt and pan angles) can specify

the orientation. Control over the lens parameters will be achieved by interfacing the computer to a commercial motorized television camera lens. Figure A-1 shows the organization of the proposed system.

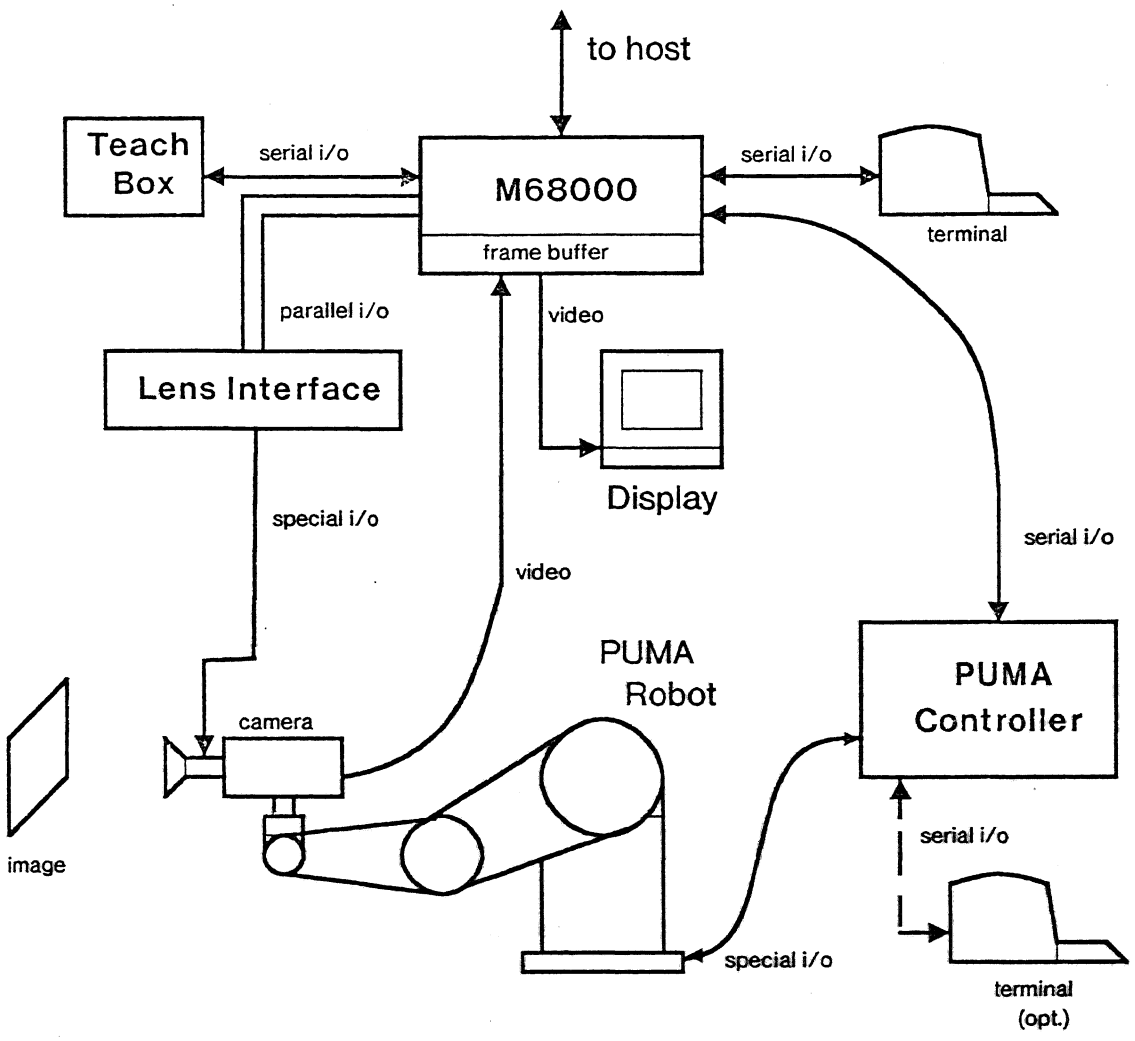


Figure A-1: Block diagram of the proposed camera system.

If weight considerations at the end of the arm become important, a fiber optic image guide to link the lens to the camera will be installed. In such a configuration, the arm need only support the weight of the lens, servo motor and protective housing, thereby freeing the camera to be mounted elsewhere.

The advantages of a computer controllable camera system are several:

- Many different avenues of research would be open to present and future Robotics Institute personnel. Examples include object tracking, image stabilization, automatic feature inspection, three dimensional shape extraction strategies and flexible assembly.

- Investigation of any one parameter in the system -- focus control, for example -- requires only that the others be constrained.

- The solution of various industrial inspection problems would be greatly facilitated. In a development mode, the system could be used to test ideas for numerous prototype systems before actual construction. Thus, there are economic benefits as well.

Part of the motivation behind this proposal is the desire to obtain experience in both hardware and software. Consequently, the tasks to be accomplished during the project fall into both categories. An outline of these tasks follows.

- Implementation of frame grabbing capability. The image processing computer for this project will be one of the Motorola M68000 microcomputer-based systems developed here at CMU. Frame grabbing and display capabilities will be provided by adding a commercial MULTI-BUS compatible frame grabbing unit to this system. Optimized low level software will be provided for the transfer of image information from the frame buffer to memory and vice versa.

- Construction of a hardware interface. The drive mechanisms of the lens will be interfaced to the image processing computer in such a way that the focus, zoom and aperture opening can be changed independently and at variable speed.

- Implementation of a software interface. A low level software interface package will be developed that provides independent control of each of the system parameters. Part of this package will drive the lens interface, while part will communicate with the robot controller.

- Provision for manual control. The Puma robot comes with a manual control unit, called a teach box, which will be interfaced to the M68000 system over a serial line. A manual controller is also available for the lens, and will be modified to provide computer control.

- Construction of the camera mount. A scheme to mount the camera on the arm will be devised, a design specified, parts machined, if necessary, and construction completed. To minimize accidental damage to the camera and lens, a protective housing will be provided.

- Implementation of image processing and control algorithms to achieve the specific theoretical and practical objectives detailed below.

Since these tasks are relatively independent, it is expected that work will proceed on each concurrently. Once the separate pieces are ready (by February or March 1982), the work of integrating them into a coherent system can begin.

Since it is hoped that this project will have both theoretical and practical value, it is proposed that the system described above be used to address the problem of automatic focusing on simple planar scenes. Known control algorithms will be implemented (possibly in firmware) and their

behaviors compared [7,8]. Possible algorithms include edge transition time minimization and high frequency content maximization. The end product should be an approximation to the procedure a television cameraperson uses while focusing an image, a process which is believed to be carried out through the use of edge inspection.

As a practical benefit, the system will be applied to the problem of detecting stress/strain patterns in glass objects through the use of polarized light. An automated solution to this problem is of enormous value to manufacturers of glass products. In particular, part of the Robotics Institute's interface with industry includes a cooperative project with Westinghouse aimed at automating the manufacture of fluorescent lamps. It is believed that a signifigant amount of the losses in this process are due to stress/strain defects in the glass components. By illuminating these components with polarized light, defects can be detected by the color patterns they create. Equipped with suitable filters, the vision system described above should be able to sense these patterns.

At the end of this project, the controllable camera system described above will be operating autonomously for research and development purposes. In addition, it is intended that the system will later be dismantled and various parts used as a front end for a much more ambitious system designed to control manipulators in real time based on video input. This system is described in detail in [4] and is depicted in Figure A-2.

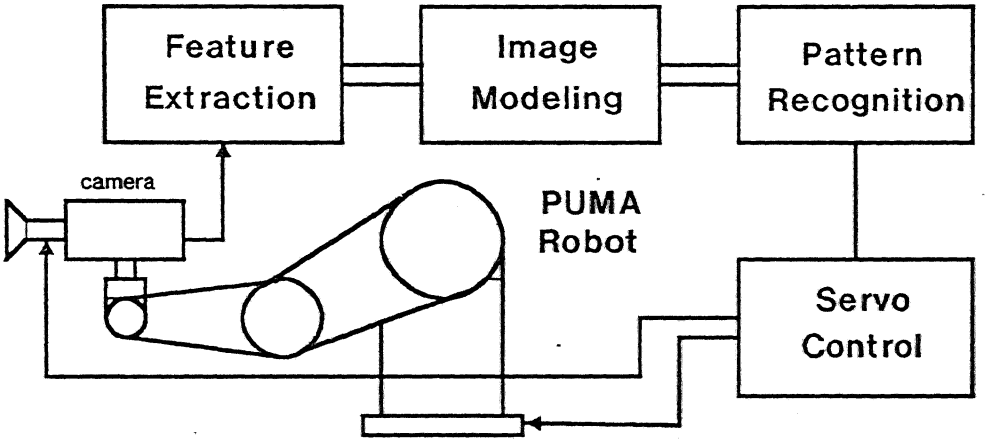


Figure A-2: Schematic Representation of the RIPl System.

In summary, it is hoped that the system described above will provide researchers with a valuable tool for computer vision research, provide a test bed for the demonstration of industrial inspection concepts, provide the author with more solid backgrounds in both hardware and software, and eventually be integrated with the ongoing efforts of the Robotics Institute.

REFERENCES

1. Arthur Sanderson and Lee Weiss, *Image Based Visual Servo Control Using Relational Graph Error Signals*, Proc. Int'l Conf. on Cybernetics and Society, pp 32-35 (Oct., 1980).

2. Allison Hunt, *Kalman Filtering Applications in Robotic Tracking*, Master's Project Report, Carnegie-Mellon University Electrical Engineering Department (in preparation).

3. John Myers, *A Supervisory Collision Avoidance System for Robot Controllers*, Master's Project Report, Carnegie-Mellon University Electrical Engineering Department (Dec., 1981).

4. Rafael Bracho and Arthur Sanderson, *RIPI: An Image Processor for Robotics*, Robotics Institute Tech. Report, Carnegie-Mellon University (in preparation).

5. Mark Handelsman, *Automated Coating Inspection of Flourescent Light Bulbs*, Master's Project Report, Carnegie-Mellon University Electrical Engineering Department (in preparation).

6. Anthony Maddox, *Automated Visual Inspection of Flourescent Lamp Components*, Master's Project Report, Carnegie-Mellon University Electrical Engineering Department (in preparation).

7. Jay M. Tenenbaum, *Accommodation in Computer Vision*, Stanford Artificial Intelligence Memo AIM-134, Stanford University (Nov., 1970).

8. Berthold Horn, *Focusing*, MIT Project MAC, Artificial Intelligence Memo No. 160 (May, 1968).

Appendix B
Interface Hardware Description

This Appendix gives the gory details on the special purpose interface hardware. For convenience, this contraption is referred to as Gertrude, *Fred's companion.*

The function of the lens controller interface is to emulate the EAROM chip in the VICON controller. Figure B-1 shows how this is done.

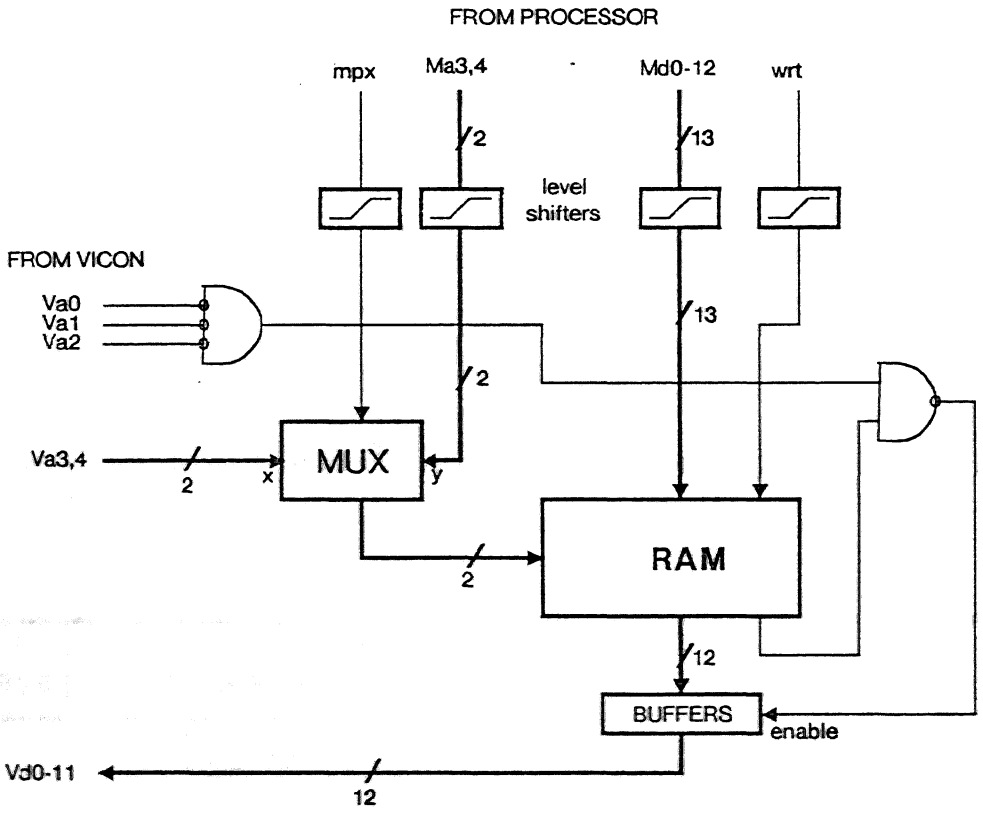


Figure B-1: Block diagram of the lens interface hardware.

Signals Md0 through Md11 are direct replacements for the data bits normally stored in the ROM. These are level shifted from TTL to CMOS and shown to the RAM. Md12 is intended to be

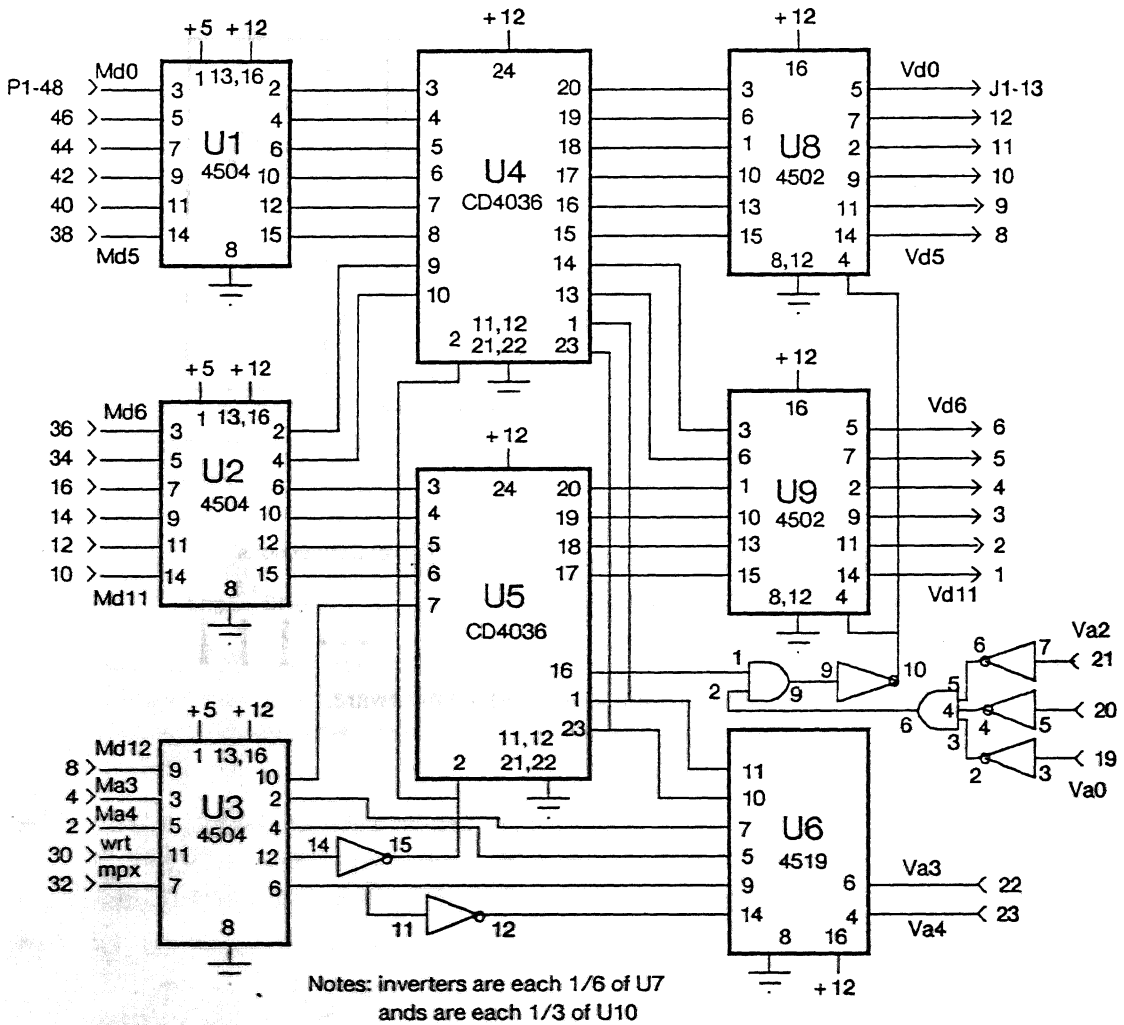
an *active parameter flag* which enables the substitution of the RAM contents for the EAROM 'contents. Address bits VaO through Va2 determine which of eight presets is selected on the controller. These are complemented and anded to produce a signal which is true only when preset 8 is selected. This signal is anded with the active parameter flag Mdl2 to produce the enable signal for the tri-state buffers which protect the RAM from the controller data bus and vice-versa. Address bits Va3 and Va4 determine which of the four parameters is to be accessed, with the following truth table.

a4	a3	parameter
0	0	focus
0	1	pan
1	0	tilt
1	1	zoom

The processor address bits Ma3 and Ma4 can be substituted for Va3 and Va4 under the control of the *mpx* bit from the processor (0 = processor control). The *wrt* bit from the processor causes the data bits MdO through Mdl2 to be written into the RAM (0 = write). From the software, the sequence of steps for writing a new value to the RAM is:

1. Put the desired data onto MdO through Mdl2,
2. Take *mpx* low to grab control of the parameter address bits.
3. Take *wrt* low to write the data into the RAM.
4. Bring *wrt* back up.
5. Bring *mpx* back up to relinquish control of the address bits*

Figure B-2 shows the complete schematic of the interface. Figure B-3 shows the chip layout



PROJECT	DESIGNER	TITLE	DATE
Popeye	John F. Schlag	Lens Interface Hardware	12/16/82

Figure B-2: Schematic of the lens interface hardware.

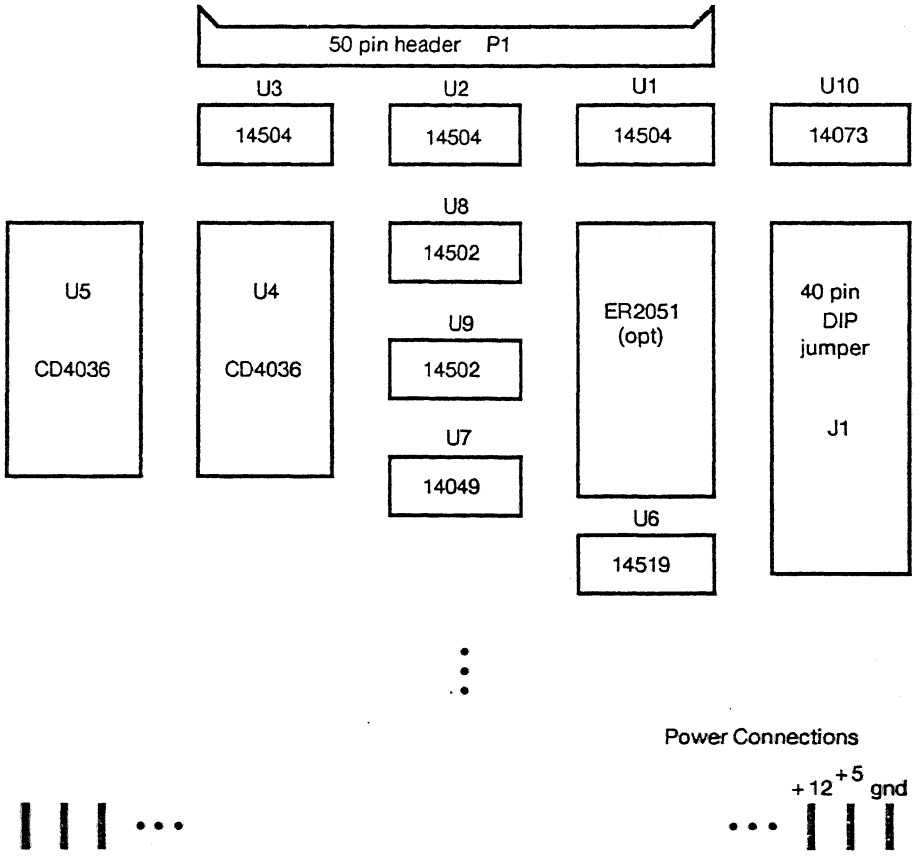


Figure B-3: Chip layout of the lens interface hardware.

References

[1] Ballard, Dana H. and Christopher C Brown.
Computer Vision.
Prentice-Hall, 1982.

[2] Basscville, M. and A. Benveniste.
Changes in Statistical Models: Various Approaches in Automatic Control and Statistics.
Rapport de Recherche de 1'INRIA 145, Universite de Rennes, Rennes, France, March, 1981.

[3] Basseville M. et al.
Edge detection using sequential methods for change in level-Part I.
IEEE trans, on ACOUSL, Speech andSig. Proa ASSP-29(1):24-31, February, 1981.

[4] Basseville, M.
Edge detection using sequential methods for change in level-Pan II.
IEEE trans, on ACOUSL, Speech andSig. Proc. ASSP-29(l):32-50, February, 1981.

[5] Bracho, Rafael and Arthur Sanderson.
Design Study for RIP I: An Image Processor for Robotics.
Technical Report CMU-RI-TR-82-3, Carnegie-Mellon University Robotics Institute, May, 1981

[6] Bracho, Rafael, John Schlag and Arthur Sanderson.
POPEYE: A Gray-Level Vision System for Robotics Applications.
IEEE Transactions on Pattern Analysis and Machine Intelligence, (in review).

[7] Brooks, Frederick P., Jr.
The Mythical Man Month.
Addlson-Wesley, 1975.

[8] Christ, James and Arthur Sanderson.
A Prototype Tactile Sensor Army.
Technical Report CMU-RJ-TR-82-R Carnegie-Mellon University Robotics Institute, September, 1981

[9J Duda, Richard 0. and Peter E Hart
Pattern Classification and Scene Analysis.
Wiley-InterScience, 1973.

[10] Feldman, J-A^etaL
The Stanford Hand-Eye Project
In *UCAI /*, pages 521~526a. 1969.

[11] Fox, Mark S.
The Intelligent Management System: An Overview.
Technical Report CMU-RI-TR-81-4, Carnegie-Mellon University Robotics Institute, December, 1982.

[12] Goldberg, Norman.
Inside Autofocus: How the Magic Works.
Popular Photography :77-83, February, 1982.

[13] Handlesman, Mark.
Automated Coating Inspection of Fluorescent Light Bulbs.
Master's thesis, Carnegie-Mellon University Electrical Engineering Department, (In preparation).

[14] Horn, Berthold.
Focusing.
MIT Project MAC, A. I. Memo 160, Massachussets Institute of Technology, May, 1968.

[15] Hunt, Alison.
Vision-Based Predictive Robotic Tracking of a Moving Target.
Master's thesis, Carnegie-Mellon University Electrical Engineering Department, January, 1982.

[16] Maddox, Anthony B.
Automated Visual Inspection of Fluorescent Lamp Components.
Master's thesis, Carnegie-Mellon University Electrical Engineering Department, April, 1982.

[17] Myers, John.
A Supervisory Collision Avoidance System for Robot Controllers.
Master's thesis, Carnegie-Mellon University Electrical Engineering Department, December, 1981.

[18] Newman, William M. and Robert F. Sproull.
Computer Science Series: Principles of Interactive Computer Graphics.
McGraw-Hill, 1979.

[19] Oppenheim, Alan V. and Ronald W. Schafer.
Digital Signal Processing.
Prentice-Hall, 1975.

[20] Papoulis, Athanasios.
McGraw-Hill Series in Systems Science: Probability, Random Variables, and Stochastic Processes.
McGraw-Hill, 1965.

[21] Pingle, Karl K. and Jay M. Tenenbaum.
An Accommodating Edge Follower.
In *IJCAI II*, pages 1-7. 1971.

[22] Pratt, William K.
Digital Image Processing.
Wiley-InterScience, 1978.

[23] Preston, Kendall et al.
Basics of Cellular Logic with Some Applications in Medical Image Processing.
Proc. of the IEEE :826-856, May, 1979.

[24] Rabiner, L. R. and B. Gold.
Signal Processing Series: Theory and Applications of Digital Signal Processing.
Prentice-Hall, Englewood Cliffs, N.J., 1975.

[25] Rauh, Michael J.
Statistical Modeling of a Fluorescent Tube Coating Process.
Master's thesis, Carnegie-Mellon University Electrical Engineering Department, December, 1982.

[26]
Robotics Institute Biographies.
A Publication of the Carnegie-Mellon University Robotics Institute.
Available from the Institute.

[27] Sakai, Toshiyuki, Makoto Nagao and Takeo Kanade.
Computer Analysis and Classification of Photographs of Human Faces.
In *First USA-JAPAN Computer Conference*, pages 55-62. 1972.

[28] Sanderson, A. C., J. Segen and E. Richey.
Hierarchical Modeling of EEG Signals.
IEEE trans. on Pattern Analysis and Machine Intelligence PAMI-2(5):405-414, September, 1980.

[29] Sanderson, A. C. and J. Segen.
A pattern-directed approach to signal processing.
In *Proceedings of the 5th. International Conference on Pattern Recognition*, pages 613-617. Miami, FL, December, 1980.

[30] Sanderson, Arthur and Lee Weiss.
Image Based Visual Servo Control Using Relational Graph Error Signals.
In *Proceedings of the International Conference on Cybernetics and Society*, pages 32-35. October, 1980.

[31] Sanderson, Arthur and Rafael Bracho.
Segmentation and Compression of Gray Level Images Using Piecewise Gradient Models.
Proceedings of 26th. SPIE Technical Symposium , August, 1982.

[32] Segen, J. and A. C. Sanderson.
Detecting Change in a Time-Series.
IEEE trans. on Inf. Theory IT-26(2):249-255, March, 1980.

[33] Tenenbaum, Jay M.
Accommodation in Computer Vision.
PhD thesis, Stanford University, November, 1970.

[34] Tenenbaum, Jay M. et al.
A Laboratory for Hand-Eye Research.
In *IFIPS*, pages 206-210. 1971.

[35] Wimberly, F.
CMU-Westinghouse Project on Fluorescent Lamp Manufacture.
Technical Report CMU-RI-81-1, Carnegie-Mellon University, Feb., 1981.

[36] Winston, Patrick H., ed.
Computer Science Series: The Psychology of Computer Vision.
McGraw-Hill, 1975.